Enforcing Resilience in Cyber-physical Systems via Equilibrium Verification at Runtime

MATTEO CAMILLI and RAFFAELA MIRANDOLA, Politecnico di Milano, Italy
PATRIZIA SCANDURRA, University of Bergamo, Italy

Cyber-physical systems often operate in dynamic environments where unexpected events should be managed while guaranteeing acceptable behavior. Providing comprehensive evidence of their dependability under change represents a major open challenge. In this article, we exploit the notion of equilibrium, that is, the ability of the system to maintain an acceptable behavior within its multidimensional viability zone and propose RUNE^2 (RUNtime Equilibrium verification and Enforcement), an approach able to verify at runtime the equilibrium condition and to enforce the system to stay in its viability zone. RUNE^2 includes (i) a system specification that takes into account the uncertainties related to partial knowledge and possible changes; (ii) the computation of the equilibrium condition to define the boundaries of the viability zone; (iii) a runtime equilibrium verification method that leverages Bayesian inference to reason about the ability of the system to remain viable; and (iv) a resilience enforcement mechanism that exploits the posterior knowledge to steer the execution of the system inside the viability zone. We demonstrate both benefits and costs of the proposed approach by conducting an empirical evaluation using two case studies and 24 systems synthetically generated from pseudo-random models with increasing structural complexity.

CCS Concepts: • **Software and its engineering** → *Software verification and validation*; *Extra-functional properties*; • **Computer systems organization** → **Architectures**; *Embedded and cyber-physical systems;*

Additional Key Words and Phrases: Cyber-physical systems, runtime verification, resilience, viability zone, equilibrium, uncertainty

ACM Reference format:
Matteo Camilli, Raffaela Mirandola, and Patrizia Scandurra. 2023. Enforcing Resilience in Cyber-physical Systems via Equilibrium Verification at Runtime. *ACM Trans. Autonom. Adapt. Syst.* 18, 3, Article 12 (September 2023), 32 pages.
https://doi.org/10.1145/3584364

1 INTRODUCTION

Cyber-physical systems (CPSs) are software-intensive systems that are embedded in the physical world. They monitor, control, and coordinate processes in both the physical and the digital world. They are characterized by the need to interact with both humans and changing environments as well as the need to be able to deal with unexpected events and uncertainties that permeate today's world [1]. To this end, CPSs are often augmented with self-adaptive capabilities, in which

Authors' addresses: M. Camilli and R. Mirandola, Politecnico di Milano, Via Golgi 42, Milano, Italy, 20133; emails: {matteo.camilli, raffaela.mirandola}@polimi.it; P. Scandurra, University of Bergamo, Viale Marconi 5, Dalmine, Italy, 24044; email: patrizia.scandurra@unibg.it.

1556-4665/2023/09-ART12 $15.00
https://doi.org/10.1145/3584364

a feedback control loop helps the system to adapt to new situations (e.g., the MAPE-K loop [2, 3]). CPSs are widespread, with applications in several domains, such as health care, automotive, and aircraft avionics systems [4–6]. For this reason, the ability to design and engineer CPSs that are *resilient* [7, 8], being able to function even during changes and/or unexpected events, is of paramount importance.

In the following, we conform to Laprie's definition [9] of resilience: the *persistence of dependability when facing changes*. Thus, the notion of dependability represents a key concept elaborated by the dependable computing community [9] as *the ability to deliver service that can justifiably be trusted*.

Ideally, the resilience of CPSs should be verified by assessing dependability properties across all possible environmental scenarios and configurations. However, exhaustive exploration of all of these dimensions to guarantee resilience by design is often unfeasible [7]. This entails taking into account (and enumerating), for instance, internal systems sources of uncertainty, such as different configurations or different software or hardware faults and failures, as well as external sources of uncertainty, such as uncertain and changing environments or unexpected situations related to human behavior [10]. Additionally, due to the sensing limitations of their hardware, CPSs need to reason and respond with only partial knowledge of the environment, adding another dimension of uncertainty.

For this reason, a resilient CPS must be able to recognize and (possibly) mitigate unexpected behaviors arising from any of these sources of uncertainties. Borrowing the terminology adopted in [7], a system should be in *equilibrium*, able to maintain a behavior within its multidimensional viability zone. This last concept is defined as the set of possible states in which the system operation is not compromised [11], that is, the set of states in which the system's dependability requirements are satisfied.

In this article, we introduce RUNE^2 (RUNtime Equilibrium verification and Enforcement), an approach that extends our prior work presented in [12]. RUNE^2 aims to verify at runtime whether a system satisfies the equilibrium property and to enforce the system to stay in its viability zone.

The approach exploits the well-known conceptual framework about the *world* and the *machine* introduced in [13] and more recently applied in the context of dependability assurance [14]. The system (or the machine) can provide the solution to the problem that exists in the environment (or the world). This is feasible only because there exists an interaction between the machine and the world through participation in the so-called shared phenomena. The specification S of the shared phenomena will satisfy the requirements R under the domain knowledge D that formalizes the assumptions made on the environment behavior. Formally, the following entailment relation must be satisfied:

$$S, D \models R. \tag{1}$$

The proposed method builds on this conceptual framework and augments the domain knowledge by considering uncertainty and the notion of the system's multidimensional viability zone explicitly. This last concept is defined as the set of possible states in which the dependability requirements R are satisfied. This is characterized in terms of relevant context attributes and corresponding desired values [15] for the variables of interest of the variability dimensions. Since the viability zone of a target system varies with and along adaptation dimensions, changes in the environment may take the system outside its viability zone. To verify that a newly entered state is still within the boundaries of the system viability zone (possibly after a trajectory of transient states that crosses these boundaries), a finite set C of *equilibrium constraints* (derived from R) that characterize the boundaries of such a viability zone are necessary. We formalize these notions by means of Equation (2), henceforth referred to as the *main equilibrium argument*:

$$S, D_{\langle\Delta\rangle} \models C(R), \tag{2}$$

indicating whether S with the domain knowledge D satisfies the equilibrium constraints C in the presence of possible changes Δ affecting the assumptions D.

RUNE^2 includes the three main activities of RUNE [12] and adds a resilience enforcement mechanism as the fourth one. The main steps of RUNE^2 are then as follows: (i) specification of the shared phenomena between environment and implementation using parametric Markov decision processes that factor in the uncertainties related to partial knowledge and possible changes; (ii) computation of the equilibrium condition that accounts for dependability requirements to define the boundaries of the viability zone; (iii) runtime equilibrium verification that leverages Bayesian inference to reduce the uncertainty under the required threshold and quantitatively reason on the system's ability to remain inside the boundaries of the viability zone; and (iv) a resilience enforcement mechanism that exploits the posterior knowledge to steer the execution of the system by maximizing the probability of enforcing the boundaries, thus staying inside the viability zone.

To show the applicability and benefits of RUNE^2, we have conducted an empirical evaluation to study feasibility, effectiveness, and costs using two different established case studies from the literature: the first from the robotics domain [16] and the second dealing with an autonomous team of Unmanned Aerial Vehicles [17, 18]. We synthesize 24 systems from pseudo-random specifications having increasing structural complexity to study the scalability of our approach.
This article extends our previous work, presented in [12], as follows.

(1) We enhance the runtime verification framework with the introduction of an enforcement mechanism that constrains the system in its viability zone.
(2) The empirical evaluation has been completely redesigned by including two additional research questions, one additional case study, and a scalability study involving synthetic specifications.
(3) We extend the related work section by including a qualitative comparison with existing approaches.
(4) We include a discussion of the major strengths and limitations of the proposed method.

The rest of this article is structured as follows. In Section 2, we introduce a running example in the robotics domain (the search and rescue robot) as well as a preview of our approach. In Section 3, we introduce background notions we use in the rest of the article. All stages of RUNE^2 are then detailed in Section 4. In Section 5, we discuss our empirical evaluation. We present related work in Section 6. In Section 7, we discuss the major strengths and limitations of our approach as well as threats to validity. We present our conclusions in Section 8.

2 PRELIMINARIES

This section introduces preliminary concepts detailed in the following sections. We introduce a running example (Section 2.1) and then a preview of our approach RUNE^2 (Section 2.2).

2.1 A Running Example: The Search and Rescue Robot

To illustrate our approach, we use a self-adaptive search-and-rescue robotic system inspired by an existing case study [16]. The system aims at supporting emergency circumstances such as fire, hurricanes, or earthquakes. Figure 1 shows a high-level overview of its main architecture components using informal notation. The managed software includes proper abstractions of the physical interfaces (sensors and actuators), and key functions to carry out the rescue tasks, such as navigation as well as obstacle/human detection. The sensors (e.g., camera, LiDAR, ultrasonic range finder)

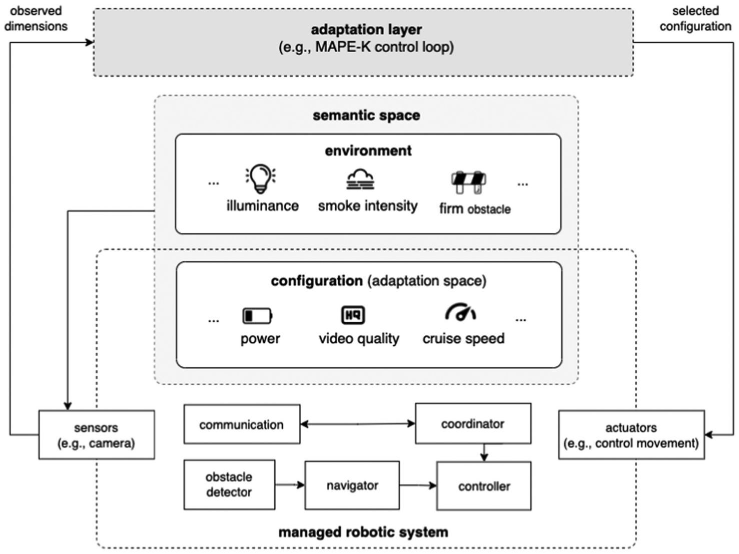


Fig. 1. Main components of the self-adaptive robotic system and its semantic space.

feed the component *obstacle detector*, which detects and classifies obstacles or human beings. This information is used by the *navigator* to plan the steering angle, acceleration, and braking, which are then put into effect by the *controller*.

The main components of the robotic software system can be configured at runtime to adapt the mode of operation in the presence of uncertain changes in its environment (e.g., available bandwidth) and resource variability (e.g., estimated battery life). These configuration changes are enacted by engineering the controller with an *adaptation layer* embedding a feedback control architecture responsible for changing the system's operation mode and meeting dependability requirements at runtime. The configuration space of the system includes multiple dimensions. For instance, *power* is a discrete configuration dimension for the controller that may operate from 0 (energy-saving mode) to 100 (full-power mode). The *cruise speed* represents a continuous configuration dimension for the navigator. The set of configuration and environment dimensions is referred to as *semantic space*. For our illustrative example, they are roughly depicted in Figure 1, while Table 1 lists all the dimensions that collectively compose the semantic space in our case study. The semantic space is defined by a set of variables, each having a space (either configuration or environment), type, and domain. For instance, *power* is an integer configuration variable that ranges in the interval $[0, 100]$ percentage level. *Illuminance* represents instead a continuous environment variable ranging from 40 lux (for sunset/sunrise) to $120k$ lux (for brightest sunlight).

The behavior of such a search-and-rescue robot comprises different execution scenarios. Let us consider the navigation scenario in which the robot must stop close to a still human body and then send an alarm message. System-level dependability requirements for this scenario may include, for instance, R_1 stating that a violation of the protective human–robot distance will happen in less than 5% of the runs. Nonetheless, changes in the semantic space affect the ability of the system to stay viable, that is, meet the equilibrium constraint derived from R_1. Low video quality, as well as high cruise speed, can result in issues for the visual perception component that might lead to invalidating R_1 in the case of hostile environmental conditions such as heavy rain or the presence of smoke.

Table 1. Semantic Space of the Search-and-Rescue Robotic System

Variable	Space	Type	Domain
Power	Configuration	Discrete	[0, 100] %
Cruise speed	Configuration	Continuous	[0, 5] m/s
Bandwidth	Configuration	Continuous	[10, 50] Mbit/s
Quality	Configuration	Categorical	{*low*, *mid*, *high*}
Illuminance	Environment	Continuous	[40, 120000] lux
Smoke intensity	Environment	Categorical	{*none*, *thin*, *thick*}
Obstacle size	Environment	Continuous	[0, 120] ft^3
Obstacle distance	Environment	Continuous	[0, 10] m
Moving obstacle	Environment	Boolean	Yes/No

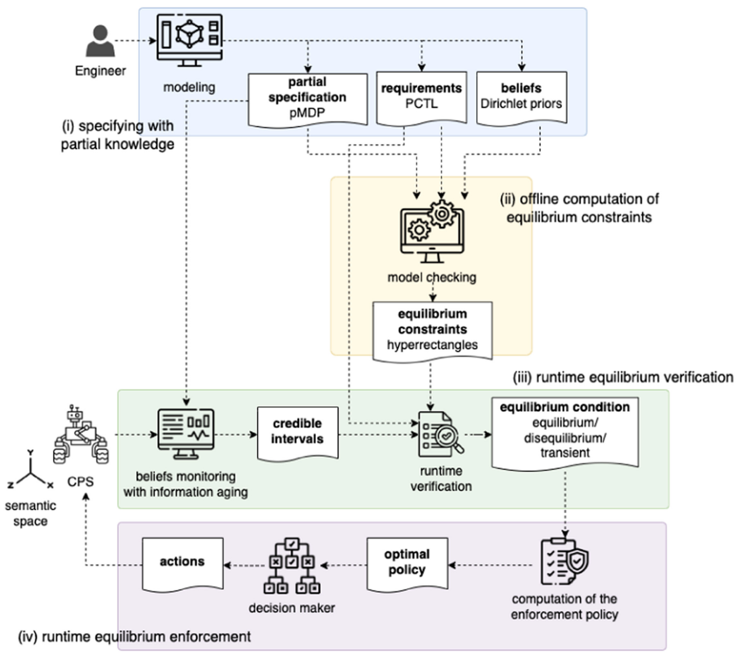

Engineer
modeling
partial specification pMDP
requirements PCTL
beliefs Dirichlet priors
(i) specifying with partial knowledge
(ii) offline computation of equilibrium constraints
model checking
equilibrium constraints hyperrectangles
(iii) runtime equilibrium verification
CPS
semantic space
beliefs monitoring with information aging
credible intervals
runtime verification
equilibrium condition equilibrium/ disequilibrium/ transient
actions
decision maker
optimal policy
computation of the enforcement policy
(iv) runtime equilibrium enforcement

Fig. 2. RUNE^2 approach overview.

2.2 Preview of the Approach

In this work, we deal with the verification and enforcement of the main equilibrium argument. We developed a specific interpretation of this abstract framework using the parametric Markov Decision Process (pMDP) formalism [19] to specify with partial knowledge the *shared world-machine phenomena* [13, 14] and Probabilistic Computation Tree Logic (PCTL) [20] (briefly described in Section 3) to formalize system-level dependability requirements from which equilibrium constraints are derived.

The approach we propose follows four steps as illustrated in Figure 2. In stage (i), the engineer specifies the shared phenomena of interest as well as the system-level requirements. Stage (ii) is a partial evaluation step. It takes as input the partial specification, the set of desired requirements, and a set of variables with initial beliefs. Variables are model parameters whose value is uncertain and becomes known at runtime, which may change over time due to changes in the semantic space. The output is a partially evaluated set of equilibrium constraints that represents a verification condition identifying the viability zone based on the requirements. In this stage, our approach uses parametric model checking algorithms known to be computationally expensive since they require an exhaustive exploration of the state space to analyze arbitrarily complex properties [21]. Since the computational cost of model checking may be prohibitive for online usage, we relegate this phase to design time and apply computationally inexpensive operations at runtime. The constraints are then evaluated in stage (*iii*) by collecting evidence and binding concrete values to variables. Here, we use Bayesian inference to quantify the uncertainty and spot violations of the main equilibrium argument caused by changes in the semantic space. The outcome of the verification step is the equilibrium condition indicating whether the constraints are satisfied or not along with the variables responsible for the negative outcome. With this information, step (*iv*) calculates the so-called optimal equilibrium policy that is then enacted by the decision maker. This latter component makes on-the-fly choices of actions according to the optimal policy whose aim is to avoid the regions violating the constraints, thus enforcing the equilibrium property.

3 BACKGROUND

This section explains the necessary background concepts and notation used in the rest of the article. The theoretical background includes the pMDP formalism (Section 3.1), the PCTL language (Section 3.2), and Bayesian inference (Section 3.3).

3.1 Parametric Markov Decision Processes

Given a set of variables θ and the set of all rational-coefficient polynomial functions (i.e., a sum of terms in which each term is given by a coefficient and a monomial) $\mathbb{Q}[\theta]$, a pMDP [19] $\mathcal{M}$ is a tuple $(S, \theta, A, s_0, \delta, AP, L)$, where S is a (finite) set of states; θ is a finite set of parameters; A is an alphabet of actions; $s_0 \in S$ is the initial state; and $\delta : S \times A \times S \to \mathbb{Q}[\theta] \cup [0, 1]$ is the partial probabilistic transition function; AP is a set of atomic propositions; $L : S \to 2^{AP}$ is a labeling function that associates to each state the set of atomic propositions that are true in that state. State transitions occur in two steps: a nondeterministic choice among available actions; and a stochastic choice of the successor state according to δ. In the rest of the article, the notation $p^a_{i,j}$ will be used as the short form for $\delta(s_i, a, s_j)$. The function $A(s_i)$ is used to denote the actions in A available from the state s_i.

Note that a parameter-free pMDP coincides with the standard MDP as defined in [19]. An MDP can be obtained from a pMDP by assigning values to parameters. Formally, we need to create an *instantiation* function $val : \theta \to \mathbb{R}$ such that the δ is well defined, that is, $\sum_{s_j \in S} p^a_{i,j} = 1$ for all $s_i \in S$ and $a \in A(s_i)$. In the following, we use $\mathcal{M}[val]$ to denote the MDP obtained from the pMDP $\mathcal{M}$ with instantiation val.

Both MDP and pMDP models can be augmented with *reward*s to quantify a benefit (or loss) due to the occurrence of certain transitions. A reward usually represents nonfunctional aspects such as average execution time, power consumption, or usability. Rewards are formally specified by using the notion of *reward structure*, that is, a function $r : S \times A \times S \to \mathbb{R}$. Given a standard MDP and a reward structure r, a deterministic policy π specifies for each state s_i the action $\pi(s_i) \in A(s_i)$ chosen by a decision maker to solve nondeterminism.

The decision maker takes actions using π and receives rewards according to r. The *value function* V^π is defined as the expectation of the cumulative reward D^π given that the decision maker acts according to that policy:

$$V^{\pi}(s_i) = \mathbb{E}[D^{\pi}(s_i)] = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s_i, a_t = \pi(s_t)\right], \tag{3}$$

where $\gamma \in [0,1]$ is a discount factor that alleviates the contribution of future (long-term) rewards in favor of present (short-term) rewards.

Using the linearity of the expected value, this expression can be formulated in a recursive form, known as the *Bellman equation* [22]:

$$V^{\pi}(s_i) = \sum_j p^a_{i,j} \cdot (r^a_{i,j} + \gamma V^{\pi}(s_j)). \tag{4}$$

The notion of optimal policy π^* is defined as the policy that maximizes the value function at the initial state, as follows:

$$\pi^* = \arg\max_{\pi} V^{\pi}(s_0). \tag{5}$$

For any standard MDP, there exists an optimal policy π^* (no worse than any other policy for that MDP) that maximizes the expected cumulative reward over a potentially infinite horizon. The value of the optimal policy V^* satisfies the Bellman optimality equation and the optimal policy π^* can be always computed using, for instance, dynamic programming, as described in [19].

3.2 Probabilistic Computation Tree Logic

Requirements of interest can be defined by using the PCTL language [20]. The syntax supports the definition of state formulas ϕ and path formulas ψ, which are evaluated over states and paths of a pMDP, respectively. Formally, a formula is defined as follows:

$$\phi ::= \ \mathit{true} \mid a \mid \phi \wedge \phi \mid \neg\phi \mid \mathcal{P}{\bowtie}p[\psi]; \ \psi ::= \ \mathsf{X}\phi \mid \phi \,\mathsf{U}\, \phi, \tag{6}$$

where $a \in AP$ and a path formula ψ is used as the parameter of the *probabilistic path operator* $\mathcal{P}_{\bowtie p}[\psi]$, such that $\bowtie \in \{\leq, <, \geq, >\}$ and $p \in [0,1]$ is a probability bound. The symbol X represents the *next* operator; U is the *until* operator. The operators G (i.e., *globally*) and F (i.e., *eventually*) can be derived from the previous ones as for CTL. A state $s \in S$ satisfies $\mathcal{P}_{\bowtie p}[\psi]$ if, under any nondeterministic choice, the probability of taking a path from s satisfying ψ is in the interval specified by $\bowtie p$.

Parametric model checking [20] is a verification technique able to analyze the parametric variability of a pMDP model $\mathcal{M}$ and determine how such a variability affects the satisfaction of a set of target PCTL properties. Formally, the outcome of the model checker is a mapping between *hyper-rectangles* and truth values, where a hyper-rectangle is a multidimensional rectangle $h = \bigtimes_{x \in \theta}[l_x, u_x]$ with $l_x, u_x \in \mathbb{R}$ lower- and upper-bound for parameter x, respectively. Intuitively, for each *true* hyper-rectangle h, the model $\mathcal{M}[val]$ satisfies the properties iff $val(x) \in [l_x, u_x]$ for all $x \in \theta$.

3.3 Bayesian Inference

The main goal of Bayesian inference [23] is to learn one or more uncertain/unknown parameters θ affecting the behavior of a stochastic phenomenon of interest. The prior knowledge (i.e., initial hypothesis or belief) of θ is incrementally updated based on a collected data sample $y = (y_1, y_2, \ldots, y_n)$ describing the actual behavior of the target phenomenon. By using Bayes's theorem, we obtain the posterior distribution $f(\theta|y)$ describing the best knowledge of θ given the evidence y.

$$f(\theta|y) \propto f(\theta) \cdot f(y|\theta) \tag{7}$$

The density $f(y|\theta)$ is referred to as the likelihood function and represents the compatibility of the data with the hypothesis. The hypothesis is often available from external sources such as expert information based on past experience or previous studies. This information is encoded by the prior $f(\theta)$.

The posterior distribution can be used, in turn, to perform either point or interval estimation. Point estimation is typically addressed in the multivariate case by summarizing the distribution through the mean, as follows:

$$\mathbb{E}[f(\theta|y)] = \int \theta \cdot f(\theta|y)\, d\theta. \tag{8}$$

In this case, the parameters θ are estimated through individual values by discarding the notion of confidence encoded into the posterior distribution. Interval estimation provides further information by accounting for the notion of confidence. In Bayesian statistics, this is achieved by calculating the highest density region (HDR), that is, the region of the sample space that contains $100(1-\alpha)\%$ of the posterior distribution. Thus, $\text{HDR}[f(\theta|y)] = C$, such that

$$P(\theta \in C|y) = \int_C f(\theta|y)\, d\theta = 1 - \alpha \tag{9}$$

typically using $\alpha = 0.05$. If $f(\theta|y)$ is multivariate, the HDR encodes a set of *credible intervals*. According to the Bayesian perspective, each interval represents the region within which each variable value falls with probability $1-\alpha$. As described in [24], the magnitude of the HDR yields the highest possible accuracy in the estimation of θ and is usually adopted as a measure of confidence in the inference process (i.e., the smaller the region, the higher the confidence).

4 THE RUNE^2 APPROACH

In this section, we detail the main steps of the RUNE^2 approach: (i) specifying with partial knowledge in Section 4.1, (ii) offline computation of equilibrium constraints in Section 4.2, (iii) runtime equilibrium verification in Section 4.3, and (iv) runtime equilibrium enforcement in Section 4.4.

4.1 Specifying with Partial Knowledge

As shown in Figure 2, our methodology leverages the design-time partial specification of the shared phenomena of interest of the target CPS by using a pMDP. Such a modeling formalism is widely accepted for specification and verification for software system dependability [19]. Its adoption here is justified by the need for modeling both nondeterminism (i.e., stimuli or events) and stochastic behavior (i.e., outcomes in response to the events). Nondeterministic stimuli are initiated either by the environment or the system itself. In the former case, events are *observable*. In the latter case, we have instead *controllable* events. In our modeling approach, we partition the set of states into observable and controllable according to the type of available stimuli, as follows:

$$S = \{s \in S : \forall a \in A(s), a \text{ observable}\} \sqcup \{s \in S : \forall a \in A(s), a \text{ controllable}\} \tag{10}$$

Given the current state of the system s_i and a stimulus a (either controllable or observable), the system evolves into another shared state s_j according to $p^a_{i,j}$. This probability value may be underspecified by means of a model variable. As anticipated in Section 3, the set of all model variables θ represents the uncertain parameters that affect the shared phenomena of interest.

Example 1 (Partial pMDP Specifications). The pMDP in Figure 3 specifies the robotic system introduced in Section 2. While navigating, the robot can encounter still human bodies or other obstacles (observable event). Here, depending on the environment conditions and the internal configuration of the system, the visual perception may succeed or fail according to probabilities

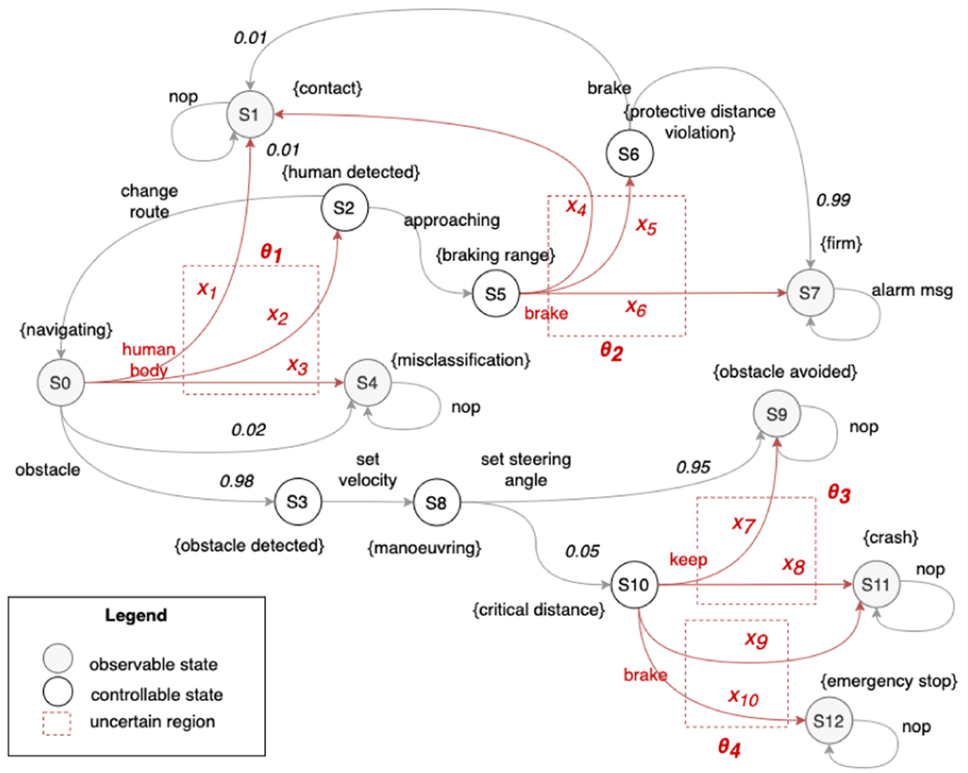


Fig. 3. Partial pMDP specifications for the search-and-rescue robotic system describing human detection and obstacle avoidance scenarios.

that may be hard to specify due to internal/external uncertainty. In this case, we leverage variables. For instance, x_3 is the probability of misclassification, whereas x_1 is the likelihood of a contact due to detection failures that leads, in turn, to safety issues.

Definition 1 (Uncertain Model Region). Given a state–action pair (s, a) in the pMDP specification, an *uncertain model region* θ_i is defined as the set of variables attached to edges (s, a, s') with $\delta(s, a, s') > 0$. The region θ_i is modeled by a k-dimensional *categorical* distribution, where k is the number of states reachable from s when a is selected.

Example 2 (Uncertain Model Region). The region θ_1 in Figure 3 is modeled by the categorical density function $Cat(x_1, x_2, x_3)$ that describes the uncertain model parameters $p^a_{0,1}$, $p^a_{0,2}$, and $p^a_{0,4}$. These parameters govern the probability of observing a contact, a detection, or a misclassification event, respectively, where the stimulus a initiated by the environment represents the presence of a *human body* within the visual perception site.

The modeler can express prior knowledge (i.e., beliefs) of transition probabilities by means of the natural conjugate prior[1] of the categorical distribution, that is, a *Dirichlet* prior density function, or simply prior $f(\theta_i)$, as described in [25]:

$$f(\theta_i) \sim Dir(\alpha_1, \ldots, \alpha_k), \tag{11}$$

[1]In Bayesian statistics, if both the prior and the posterior are in the same probability distribution family, they are called *conjugate distributions*. The prior is called a *conjugate prior* [24].

Table 2. System-Level Dependability Requirements of the Search-and-Rescue Robot

Requirement	Scenario	Natural Language Statement	PCTL Property
R_1	Human detection	Violations of the protective distance between the robot and human beings will occur in less than 5% of the runs.	$P_{<0.05}[\mathsf{F}\ \textit{protective distance violation}]$
R_2	Human detection	Contacts between the robot and human beings will occur in less than 1% of the runs.	$P_{<0.01}[\mathsf{F}\ \textit{contact}]$
R_3	Obstacle avoidance	Whenever an obstacle has been detected, a crash between the robot and the obstacle will occur in less than 1% of the runs.	$P_{\geq 1}[\mathsf{G}(\textit{obstacle detected} \rightarrow P_{\geq 0.99}[\neg \textit{crash}\ \mathsf{U}\ \textit{obstacle avoided}])]$
R_4	Obstacle avoidance	Whenever an obstacle has been detected, it will be avoided without reaching a critical distance in more than 95% of the runs.	$P_{\geq 1}[\mathsf{G}(\textit{obstacle detected} \rightarrow P_{>0.95}[\neg \textit{critical distance}\ \mathsf{U}\ \textit{obstacle avoided}])]$

with $\alpha_i > 0$ concentration parameters, each one for each category in the corresponding categorical distribution. As described in [24], a prior can be either determined from past information, such as previous observations of the phenomena of interest, or crafted from the purely subjective assessment of a domain expert. When no information is available, an *uninformative* prior can be created to reflect a balance among outcomes. In this latter case, the modeler can adopt concentration parameters $\alpha_i = 0.5$ for all $i = 1, \ldots, k$.

As anticipated in Section 3, to formalize dependability requirements of interest we make use of the PCTL syntax. This choice is motivated by the practical need for verifying properties of interest upon the pMDP specification by using off-the-shelf model checking tools, such as PRISM [26]. With the aid of this language, engineers can express properties about the system that are typically domain dependent. In our context, we express properties that can be instanced by using either unbounded or constrained reachability properties adapted to the PCTL syntax.

Example 3. PCTL dependability requirements for the human detection scenario are reported in Table 2. R_1 and R_2 are expressed in terms of simple unbounded reachability properties preceded by the P operator. which is used to reason about the probability of an undesired event. In this scenario, we verify that the probability of eventually reaching a failure state (i.e., either *distance violation* or human–robot *contact*) is acceptable (i.e., below specific thresholds).

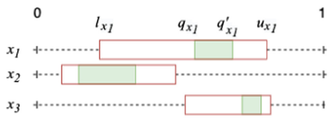
4.2 Offline Computation of Equilibrium Constraints

This stage aims at studying the parameter space of the pMDP model in order to pre-compute the *equilibrium constraints* that guarantee the satisfaction of dependability requirements under changes in the semantic space. Since the specification admits parametric variability of the uncertain regions, this pre-computation is necessary to build the basis for efficient runtime verification methods. To achieve this goal, we exploit the parametric model checking functionality of PRISM to analyze how the actual values of θ parameters affect the satisfaction of PCTL requirements (e.g., dependability properties in Table 2). Formally, given a PCTL property p, the outcome of the model checker is a mapping between *hyper-rectangles* $\mathcal{H}_p$ and truth values $\{\mathtt{True}, \mathtt{False}\}$, where a hyper-rectangle h is a multidimensional rectangle defined as follows:

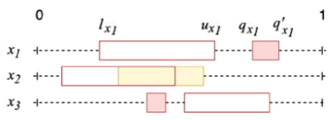
$$h = \prod_{x \in \theta} [l_x, u_x], \tag{12}$$

with $l_x, u_x \in [0, 1]$ lower- and upper-bound for the model parameter x, respectively. Intuitively, given an instantiation function *val* and a hyper-rectangle h mapping to $\mathtt{True}$, the MDP $\mathcal{M}[val]$ satisfies p iff $val(x) \in [l_x, u_x]$ for all $x \in \theta$.

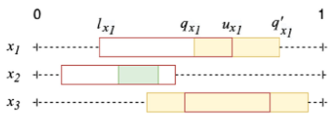
We define the ability to meet dependability requirements under changes in the semantic space by instantiating the notion of equilibrium constraints $C(R)$ introduced in Equation (2).



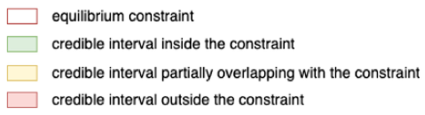
(a) Equilibrium condition.



(b) Disequilibrium condition.



(c) Transient condition.


equilibrium constraint
credible interval inside the constraint
credible interval partially overlapping with the constraint
credible interval outside the constraint

(d) Legend.

Fig. 4. Visualization of the equilibrium conditions for the uncertain region θ_1.

Definition 2 (Equilibrium Constraints). Given a pMDP specification $\mathcal{M}$, a number of PCTL requirements $p_1, \ldots, p_m$, and the set of *true* hyper-rectangles $\mathcal{H}_{p_i}^{\top}$ for the requirement p_i, the equilibrium constraints are defined as a set of hyper-rectangles $\mathcal{H}^*$ as follows:

$$\mathcal{H}^* = \left\{ h_1 \cap \cdots \cap h_m : (h_1, \ldots, h_m) \in \prod_i \mathcal{H}_{p_i}^{\top} \right\}. \tag{13}$$

The constraint $\mathcal{H}^*$ defines all possible *val* functions, such that $\mathcal{M}[val]$ satisfies the requirements $p_1, \ldots, p_m$.

Example 4 (Equilibrium Constraints). Figure 4 represents the equilibrium constraints. In this example, the constraints yield the boundaries for the variables in θ_1. Essentially, they encode verification conditions that will hold in order to meet the target dependability properties R_1 and R_2, even under changes in the semantic space.

It is worth noting that model checking can be computationally expensive [21]. Since its online usage may be prohibitive, we keep this pre-computation separated as an offline phase, where we can execute demanding activities without interfering with the system's operation.

4.3 Runtime Equilibrium Verification

In this stage, we aim to verity at runtime the behavior of the target CPS embedded in a closed-loop setting with the surroundings. We check whether the system in operation is able to keep equilibrium in order to meet the required dependability properties under change.

Example 5 (Execution Context). A concrete execution context for our robotic system can be defined by means of a valid assignment to semantic space variables in Table 1 (e.g., *power* = *full*, *cruise speed* = *5*, *illuminance* = 1, 000, *smoke intensity* = *none*). Sudden changes in this context might occur. For instance, very high luminous flux incidents on the camera sensor combined with unexpected thick smoke intensity could lead to fooling the visual perception component of the robotic software. Even under such changes, the system will meet the dependability requirements (e.g., R_1 and R_2) to achieve resilience.

As shown in Figure 2, to quantitatively reason about the effect of changes, we collect runtime evidence to conduct *belief monitoring*. We apply an *adaptive observation aging* mechanism to filter out old information when changes occur to alleviate the negative effect of historical data. Belief monitoring is carried out by using *Bayesian inference* in order to incrementally update the prior knowledge expressed by the Dirichlet distribution for all uncertain model regions θ_i. Formally, given the prior in Equation (11), the posterior $f(\theta_i|y)$, with $y = n_1, \ldots, n_k$, can be obtained by applying the following very efficient updating rule:

$$f(\theta_i|y) \sim Dir(\alpha_1 + n_1, \ldots, \alpha_k + n_k), \tag{14}$$

where n_j is the number of occurrences of the associated model transitions in the region θ_i (i.e., categories in the corresponding categorical distribution).

The posterior distributions are then exploited to summarize the updated knowledge for each region θ_i, by computing the region HDR reported in Equation (9). The HDR encodes the set of credible intervals for all variable $x \in \theta$ as follows:

$$\prod_{x \in \theta} [q_x, q'_x] : P(x \in [q_x, q'_x]) \geq 1 - \alpha. \tag{15}$$

Compared with point estimates, the credible intervals yield quantified uncertainty (or estimation errors). As reported in [27], with point estimates, the uncertainty propagates in ways that are unknown but likely to be significant for the verification outcomes. This raises concerns about the validity of decisions based on this latter approach. Thus, we leverage the credible intervals in Equation (15) to overcome this major limitation. The HDR magnitude allows our verification procedure to determine whether the collected evidence is enough to reduce the uncertainty under acceptable levels and produce valid outcomes, as detailed in the following.

The outcome of the runtime verification procedure is based on the two mutually exclusive conditions that make the main equilibrium argument operational. Essentially, we distinguish between success and failure (i.e., Equation (2) holds or not) by means of the notion of *equilibrium* and *disequilibrium*, respectively.

Definition 3 (Equilibrium). Given the posterior $f(\theta_i|y)$ for each θ_i and the equilibrium constraints $\mathcal{H}^*$, we say that the equilibrium condition holds iff:

$$\exists \prod_{x \in \theta} [l_x, u_x] \in \mathcal{H}^* : \forall x \in \theta,\ [q_x, q'_x] \subseteq [l_x, u_x]. \tag{16}$$

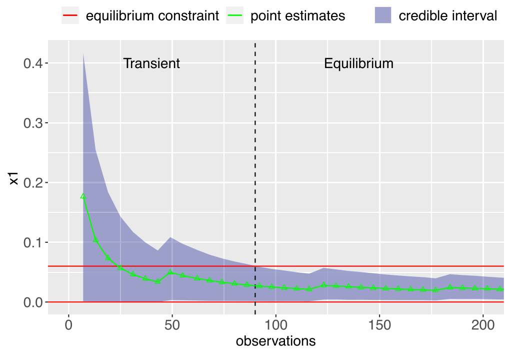
Definition 4 (Disequilibrium). Given the posterior $f(\theta_i|y)$ for each θ_i and the equilibrium constraints $\mathcal{H}^*$, we say that the disequilibrium condition holds iff:

$$\forall \prod_{x \in \theta} [l_x, u_x] \in \mathcal{H}^*, \exists x \in \theta :\ [q_x, q'_x] \cap [l_x, u_x] = \emptyset. \tag{17}$$

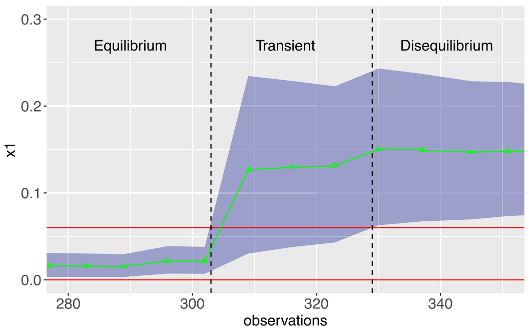
In the case that none holds, we say that the system has *transient behavior* in which changes in the equilibrium condition are detected but the degree of uncertainty is substantial. Thus, further evidence must be collected in order to reduce the size of the credible intervals.

The formal notions of the equilibrium, disequilibrium, and transient behavior can be explained by Figure 4, which contains three different snapshots of the credible intervals computed by monitoring the region θ_1 of the specification in Figure 3. Figure 4(a) shows an equilibrium condition. In this case, all of the uncertainty is below the required level since we can see that all of the credible values for θ parameters fall inside the bounds of the equilibrium constraint. Figure 4(b) shows a disequilibrium condition. In this latter case, the credible intervals for x_1 and x_2 exclude all valid instantiations $\mathcal{M}[val]$ according to the equilibrium constraint. Figure 4(c) shows instead a condition in which the uncertainty is higher than the required level since the credible intervals admit both valid and non-valid instantiations according to the equilibrium constraints. In this latter scenario, we recognize that the equilibrium is subject to changes due to variations in the semantic space.

Example 6 (Runtime Verification). Figure 5(a) shows an example of runtime verification by considering a single variable x_1 in the region θ_1 of the pMDP in Figure 3. We assume that the equilibrium constraints yield the bounds $[0.0, 0.06]$. From 0 to 90 observations, the behavior of the system



(a) Equilibrium scenario.



(b) Disequilibrium scenario.

Fig. 5. Online verification examples.

is transient since the uncertainty is substantial; therefore, more evidence is required. From the observation 90, the credible interval falls into the constraint by satisfying the equilibrium condition in Equation (16).

It is worth noting that uncertain parameters θ are prone to dynamic changes due to changes in the semantic space (affecting the shared phenomena). On the one hand, Bayesian inference in Equation (14) gets slower to detect changes as the number of observation increases. This might even lead to failures in detecting them when they are short-lived [28]. On the other hand, the equilibrium conditions in Equations (16) and (17) require a reduction of the uncertainty that can be achieved only by collecting more observations. Such a trade-off implies that traditional *static* observation aging mechanisms [28] do not fit our context. To overcome this issue, our observation aging mechanism follows the *Lightweight Adaptive Filtering* approach introduced in [29]. Our monitoring process produces interval estimates that show a certain degree of dispersion around the actual values. Then, for each incoming measurement, we decide whether it can be reasonably explained under this assumption or not. The quantification of this dispersion is given by the magnitude of the credible interval $[q_x, q'_x]$ for all $x \in \theta$ that we use to define the boundaries able to detect changes. At the core of the adaptive mechanism, there is the Bayesian inference process, which enables the online dynamic correction of the boundaries computed through the HDR (see Equation (9)) of the marginalized distribution $f_x(\cdot)$ for all $x \in \theta$. In addition to this machinery, we carry out point estimates by considering only the observations $\hat{y}$ in a small *timestep*.[2] We compute a new point estimate e according to Equation (8), as $e = \mathrm{E}[f_x(\theta|\hat{y})]$. A new estimate outside the boundaries is likely assumed to be a bad smell of undergoing changes. In this latter case, we filter out the history up to the latest timestep in order to provide a quick response to changes. If the new estimate does not contradict the hypothesis, we keep the full history in order to increase the accuracy of the inference process.

We denote a new point estimate e inside the boundaries of a region A using the indicator function $\mathbb{1}_A(e) = 1$ if $e \in A$, 0 otherwise. Thus, given k timesteps, the observations $\hat{y}$ in the k-th timestep, the full history of observations y, and the two credible intervals $\mathrm{HDR}_{x,y} = \mathrm{HDR}[f_x(\theta|y)]$ and $\mathrm{HDR}_{x,\hat{y}} = \mathrm{HDR}[f_x(\theta|\hat{y})]$, we compute the boundaries $[q_x, q'_x]$ for all $x \in \theta$ at timestep k, as follows:

$$\begin{aligned} q_x &= \inf \mathrm{HDR}_{x,y} \cdot \mathbb{1}_{\mathrm{HDR}_{x,y}}(e) + \inf \mathrm{HDR}_{x,\hat{y}} \cdot (1 - \mathbb{1}_{\mathrm{HDR}_{x,y}}(e)) \\ q'_x &= \sup \mathrm{HDR}_{x,y} \cdot \mathbb{1}_{\mathrm{HDR}_{x,y}}(e) + \sup \mathrm{HDR}_{x,\hat{y}} \cdot (1 - \mathbb{1}_{\mathrm{HDR}_{x,y}}(e)) \end{aligned} \tag{18}$$

[2]Following the guideline in [29], we adopt a timestep of 75 observations.

According to Equation (18), the indicator function is equal to 1 if and only if the estimate in the last timestep k does not contradict the hypothesis made with 95% credibility according to the evidence y. Larger (inaccurate) credible intervals reduce the ability to detect small changes, whereas very small (and accurate) intervals increase the sensitivity. Whenever a change is detected (i.e., the outcome of the indicator function turns from 1 to 0), the filter removes historical data older than k by setting $y = \hat{y}$.

Example 7 (Adaptive Filtering). Let us consider the scenario in Figure 5(b). The point estimates for x_1 fall inside the boundaries of the credible interval until 303 observations. As the forthcoming point estimates cannot be reasonably explained under the current credible interval, we consider them as a bad smell of changes. The adaptive filtering mechanism is triggered. We keep only the history of the last timestep and the dynamic boundaries $[q_x, q'_x]$ change according to Equation (18). The verification process detects a transient behavior that eventually leads to a disequilibrium condition (from observation 329).

4.4 Runtime Equilibrium Enforcement

According to our methodology in Figure 2, this step occurs after evaluating the equilibrium at runtime. If the equilibrium condition does not hold, the enforcement mechanism is triggered. In this case, the outcome of the verification stage is used as input to calculate the optimal policy aiming at maximizing the probability of enforcing the equilibrium.

To describe the enforcement mechanism, we first introduce the notion of *equilibrium reward structure*, motivated by the practical need to identify the controllable actions that maximize the probability of avoiding the model regions exceeding the boundaries of the viability zone.

Definition 5 (Equilibrium Reward Structure). Given a pMDP specification, the set of uncertain model regions θ_i and the posterior $f(\theta_i|y)$ for all i, the equilibrium constraints $\mathcal{H}^*$ and c, C, constant values such that $C \gg c$, the *equilibrium reward structure* r_e is defined as follows:

$$r_e(s, a, s') = \begin{cases} c & [q_x, q'_x] \cap [l_x, u_x] = \emptyset, \text{ with } x \text{ attached to } (s, a, s') \text{ and } [l_x, u_x] \in \mathcal{H}^* \\ C & \text{otherwise} \end{cases}. \quad (19)$$

The rationale of the reward structure in Equation (19) is as follows. We assign a low reward c to model transitions associated with variables that, according to the posterior knowledge, fall outside the corresponding boundaries defined by the equilibrium constraints. Instead, we assign a high reward $C \gg c$ to transitions associated with variables that, according to the posterior knowledge, meet the corresponding constraint.

Given this reward structure r_e, the enforcement mechanism calculates the optimal exploration policy π^* maximizing the expected cumulative reward in the long run by using the *Value Iteration* algorithm [19]. The optimal policy is then given as input to the decision-maker component that makes on-the-fly choices according to π^* to steer the execution and, if possible, avoid those regions violating the equilibrium (i.e., regions associated with low rewards). The decision maker selects the action $a = \pi^*(s)$ for all $s \in S$, such that s is controllable.

It is worth noting that the equilibrium reward structure is stationary, that is, it does not change as long as the outcome of the verification step remains the same. As shown in Figure 2, there exists a loop between the two latter runtime stages. A change in the equilibrium condition triggers the (re)computation of the optimal policy that, in turn, feeds the decision maker.

Example 8 (Equilibrium Enforcement). Let us consider the pMDP specification in Figure 3. By carrying out the equilibrium verification step, we may discover that the uncertain region θ_3 does not meet the equilibrium condition, for instance, by breaking the system-level requirement R_3

stating that a crash between the robot and an obstacle will occur in less than 1% of the runs. In this case, the enforcement mechanism computes the equilibrium policy by assigning a low reward to transitions in θ_3 and a high reward to transitions in other regions. Thus, whenever the robot recognizes a *critical distance* (i.e., controllable state S_{10}), the decision-maker component can steer the execution by forcing the action *brake*, thus hitting θ_4 rather than θ_3.

5 EVALUATION

In this section, we report on the empirical evaluation of RUNE^2. We introduce our research questions (Section 5.1), a second case study we used in the evaluation together with the rescue robot system (Section 5.2), the design of the evaluation (Section 5.3), and then we present the major results (Section 5.4).

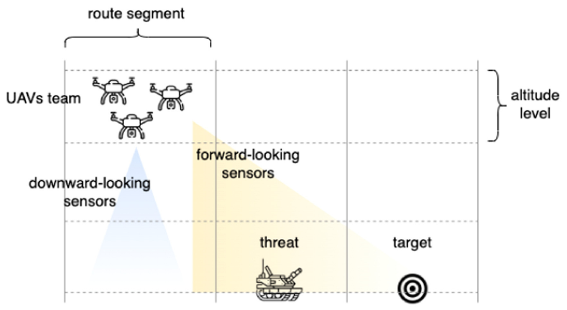
5.1 Research Questions

The purpose of our empirical evaluation is to study the extent to which RUNE^2 can detect violations of the equilibrium argument, the effectiveness and latency of the detection and the enforcement process, as well as the scalability of the approach. In particular, we aim to answer the following research questions.

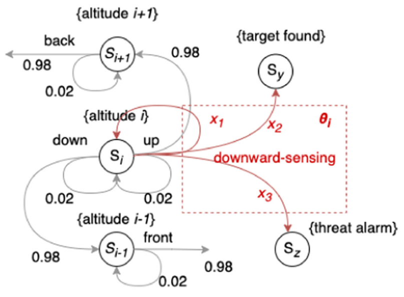
RQ1: What is the ability of our approach in detecting equilibrium violations due to changes in the semantic space?
RQ2: What is the effectiveness of the equilibrium violation detection when adopting the credible intervals compared with point estimates?
RQ3: What is the latency of the equilibrium violation detection when applying the adaptive observation aging mechanism compared with no aging?
RQ4: What is the effectiveness of the resilience enforcement policy after detecting equilibrium violations?
RQ5: What is the scalability of the resilience enforcement policy computation?

5.2 The Unmanned Aerial Vehicles System

In addition to the first case study introduced in Section 2.1, we evaluate RUNE^2 using another existing CPS benchmark, an autonomous team of unmanned aerial vehicles (UAVs) carrying out a surveying mission in a hostile and unknown environment [17, 18]. While flying along the planned route at a constant speed, the UAVs have to detect targets on the ground using downward-looking sensors. Along the route, there might be threats that can damage the vehicles. For this reason, the team uses forward-looking sensors to spot their location. The distance from the UVAs to existing threats affects the likelihood of detecting them, whereas the distance from the UVAs to the ground affects the likelihood of detecting the targets but also the likelihood of being damaged by a threat. Figure 6(a) shows a high-level view of the UAVs; Figure 6(b) shows a small extract of the pMDP model partially specifying their behavior. At each route segment, the team can decide to increase/decrease the altitude or sense the presence of targets/threats. As shown in the pMDP model, the downward-sensing (controllable) action may lead to a successful mission (e.g., probability x_2 from the altitude state S_i) or may lead to UAV detection by existing threats (e.g., probability x_3 from S_i) with a substantial degree of uncertainty (i.e., uncertain region θ_i). According to the PCTL system-level requirements listed in Figure 6, the overall probability of eventually satisfying the mission objective to find the targets will be higher than a given lower bound. At the same time, the probability of eventually being detected by existing threats shall be lower than a given upper bound. The ability of reasoning about this trade-off is complicated by the uncertainty about the environment that, depending on the configuration of the vehicles, may affect the probability of



(a) High-level view of the UAVs mission.



(b) Small extract of the UAVs pMDP model.

requirement	scenario	natural language statement	PCTL property
R_1	target detection	The team of UAVs shall eventually detect the target in more than 90% of the runs.	$P_{>0.90}[\mathsf{F}\ \textit{target found}]$
R_2	threat avoidance	Existing threats shall eventually detect the presence of the UAVs in less than 20% of the runs.	$P_{<0.2}[\mathsf{F}\ \textit{threat alarm}]$

Fig. 6. Overview of the UAV mission, an extract of the pMDP specifications, and the system-level requirements.

Table 3. Semantic Space of the UAV System

Variable	Space	Type	Domain
Formation	Configuration	Categorical	{*tight*, *loose*}
Flying speed	Configuration	Continuous	[5, 50] mph
Electronic countermeasure	Configuration	Boolean	Yes/No
Weather	Environment	Categorical	{*sun*, *clouds*, *rain*, *fog*}
Time of day	Environment	Discrete	[0:00 am, 11:59 pm]
Threat range	Environment	Continuous	[0.9, 3.7] km
#threats	Environment	Discrete	[1, 10]

detecting the targets as well as the probability of being eventually damaged by threats. The semantic space (both configuration and environment dimensions) of the UAVs is reported in Table 3. The environment and configuration variables affect the ability to satisfy the requirements. For instance, a tight formation and the adoption of electronic countermeasures reduce the probability of being damaged by a threat. Other examples are the weather condition and the time of the day. The presence of fog combined with starlight during the night hours may reduce the likelihood of being detected by threats but at the same time may reduce the likelihood of finding the target.

The total number of route segments and altitude levels affects the size of the pMDP specification in terms of structural elements. In our experiments, we set 20 route segments and 20 altitude levels, leading to a model with 482 structural elements.

5.3 Design of the Evaluation

We addressed the research questions RQ1 to RQ5 by conducting a number of controlled experiments simulating our 2 case studies (the search-and-rescue robot and the team of UAVs) and an additional 24 systems synthetically generated from pseudo-random pMDP models having increasing structural complexity. Table 4 lists the systems used in our evaluation and summarizes their structural complexity in terms of states, actions, and the total number of structural elements.

Table 4. Case Studies and Their Structural Complexity

System	#States	#Actions	#Structural Elements	#Uncertain Regions
Case study 1: search-and-rescue robot	12	3	37	4
Case study 2: team of UAVs	42	6	482	8
Synthetic 1 – synthetic 24	50–1600	8–64	400–102400	8–1200

Table 5. Map of RQs to Systems, Factors, and Measurements

RQ	Subject	Systems	Factors	Measurements
1	Violation detection ability	Case study 1 Case study 2	Changes in semantic space Magnitude of the change Duration of the change Equilibrium constraints Adaptive aging: Yes/No	Point estimates Interval estimates
2	Violation detection effectiveness	Case study 1 Case study 2	Changes in semantic space Magnitude of the change Duration of the change Equilibrium constraints Adaptive aging: Yes/No	#Violations detected
3	Violation detection latency	Case study 1 Case study 2	Changes in semantic space Magnitude of the change Equilibrium constraints Adaptive aging: Yes/No	#Observations
4	Equilibrium enforcement effectiveness	Case study 1 Case study 2	Regions violating the equilibrium Enforcement mechanism: Yes/No	#Hits regions
5	Equilibrium enforcement scalability	Synthetic 1 Synthetic 2 ... Synthetic 24	Structural complexity: states, actions Regions violating the equilibrium	Execution time

Table 5 maps each research question to the systems as well as factors and measurements in our controlled experiments.

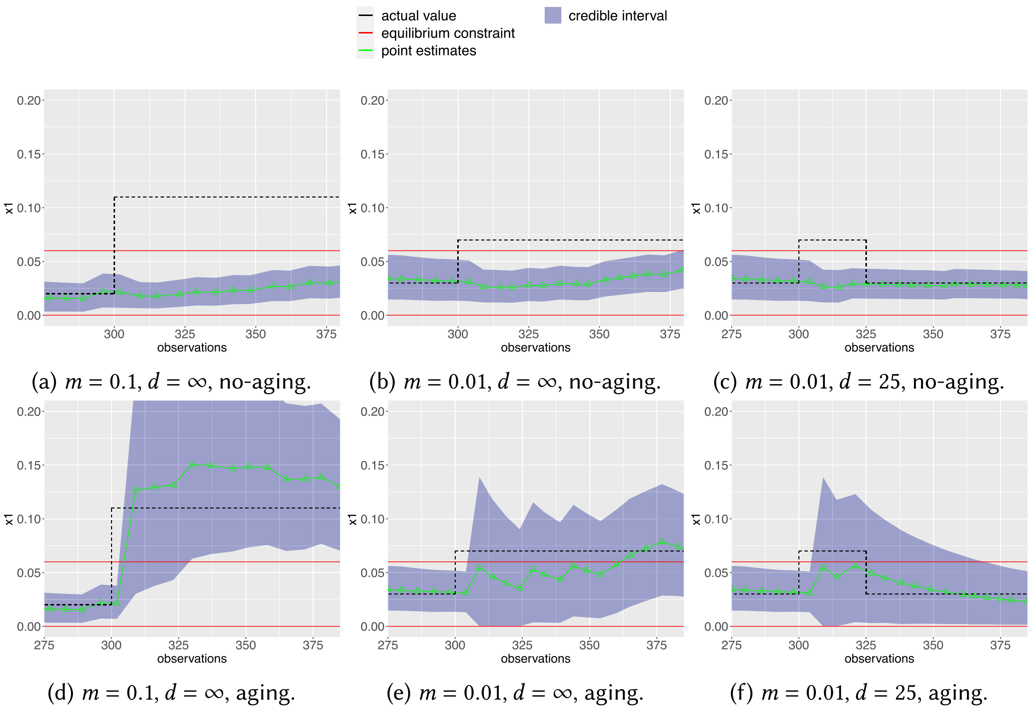
In our experiments, we controlled for the factors of interest depending on the research question: the semantic space (environment and configuration dimensions); the execution context that determines the changes occurring in the semantic space (magnitude and duration of the perturbations occurring in the uncertain regions); the equilibrium constraints (lower- and upper-bounds for the uncertain parameters); the technique used to filter historical data (lightweight adaptive filtering vs. no filtering); the equilibrium enforcement mechanism (enforcement vs. no enforcement); and the structural complexity of the pMDP specifications in terms of the number of structural elements (order of magnitude from 10 to 10^5).

We repeatedly executed[3] the runtime equilibrium verification and enforcement methods on our case studies and carried out both point and interval estimates for all uncertain regions. Then, we measured the ability to detect equilibrium violations, the number of violations detected, the number of observations required to detect them, the number of times regions violating the equilibrium are executed, and the time required to compute the equilibrium enforcement policy.

All of the experiments have been conducted by using a commodity hardware machine equipped with: 2.3 GHz Dual-Core Intel Core i5 CPU, and 8 GB 2133 MHz LPDDR3 RAM.

In the following, we discuss the most relevant results and refer the reader to our dataset for the replicability of our results.[4]

[3]Each execution includes multiple runs of one or more scenarios according to the pMDP specifications (from the initial state to an absorbing state) up to a given budget in terms of number of observations.
[4]Replication package publicly available at https://doi.org/10.5281/zenodo.4737491.



(a) $m = 0.1$, $d = \infty$, no-aging. (b) $m = 0.01$, $d = \infty$, no-aging. (c) $m = 0.01$, $d = 25$, no-aging.

(d) $m = 0.1$, $d = \infty$, aging. (e) $m = 0.01$, $d = \infty$, aging. (f) $m = 0.01$, $d = 25$, aging.

Fig. 7. Equilibrium verification over time in the rescue robot system under different scenarios.

5.4 Results

5.4.1 RQ1 (Violation Detection Ability). To address the first question, we conducted a preliminary study in which we observed the results of our runtime equilibrium verification on our two case studies (the search-and-rescue robot and the team of UAVs) under alternative execution contexts. We controlled the execution context by perturbing the actual value of the uncertain parameters to cause equilibrium violations. The injected violations have different *magnitude* (distance from the actual value to the nearest bound of the equilibrium constraint) and *duration* (length of the perturbation in terms of the number of observations). We created the following three scenarios of interest for both case studies:

(1) Substantial magnitude ($m = 0.1$) and unbounded duration ($d = \infty$)
(2) Small magnitude ($m = 0.01$) and unbounded duration ($d = \infty$)
(3) Small magnitude ($m = 0.01$) and bounded duration ($d = 25$)

Case study 1 (search-and-rescue robot). Figure 7 shows an extract of the results for each context by focusing on a single variable x_1 in the uncertain region θ_1.[5] Here, we discuss major results of the equilibrium verification without (and then with) the lightweight adaptive filtering mechanism. Scenario (1) reproduces a sudden change of x_1 from 0.02 to 0.11 occurring after 300 observations. Figure 7(a) shows both point and interval estimates over time when the aging mechanism is not adopted. Here, we can see the negative effect of historical data making the inference process very slow. Even though the magnitude of the change is substantial, the violation of the equilibrium

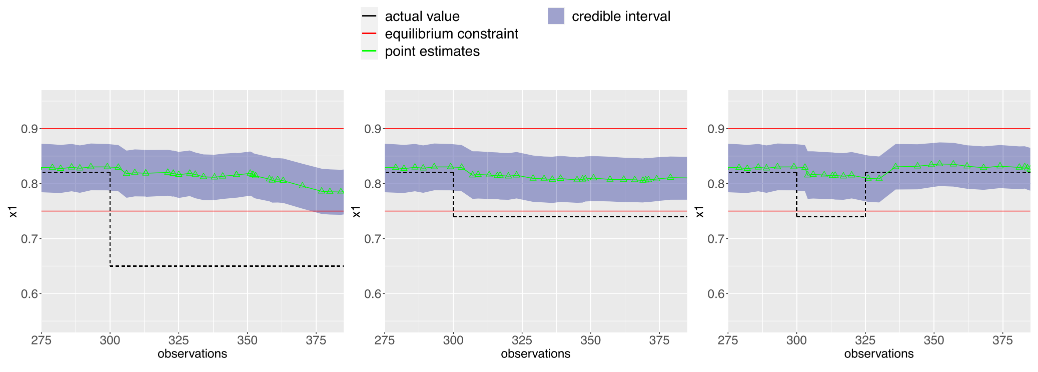
[5]We let the reader refer to our dataset to access the data of all the uncertain regions.

condition is not detected within the visible observation window (from 300 to 380). A similar situation occurs in scenario (2) illustrated in Figure 7(b). While the inference process is eventually expected to converge in the long term, scenario (3) in Figure 7(c) shows a short-lived perturbation that cannot be detected. Figure 7(d) shows scenario (1) but adopts the adaptive aging mechanism. Here, we can see a large gap between point estimates from 303 to 310 observations. By monitoring the credible interval, we are able to detect transient behavior from 310 to 329 observations. At this stage, the point estimates already fall outside the equilibrium constraint, even though the uncertainty is still higher than the required threshold. From observation 329 on, the credible interval is small enough to detect the disequilibrium. Note that point estimates are not able to distinguish between transient and steady states. This negative effect is even worse in scenarios (2) and (3), illustrated in Figures 7(e) and 7(f), respectively. Figure 7(b) shows that a number of point estimates fall inside the equilibrium constraint, whereas the credible interval detects a transient behavior right after the injection (from 310 on). Since the duration is not bounded, the point estimates eventually converge to the actual value of x_1. Thus, we can detect the equilibrium violations from observation 363. Figure 7(c) shows that, under a small and short lived perturbation, we are able to spot transient behavior through interval estimates, whereas the point estimates do not exceed the equilibrium constraint. In this latter case, the uncertainty of point estimates is significant and leads to wrong verification outcomes.

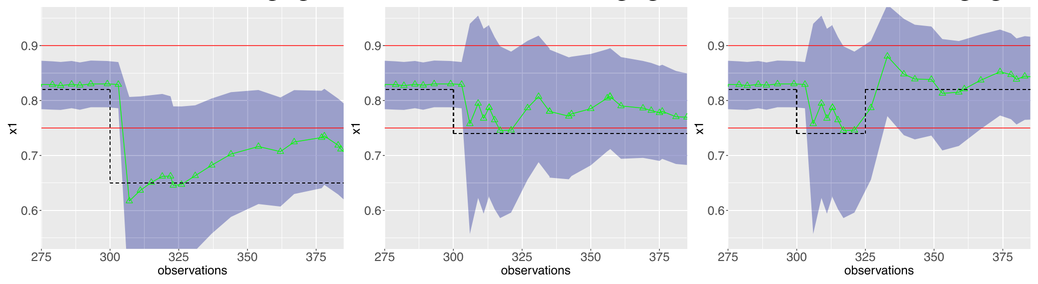
Case study 2 (team of UAVs). Figure 9 shows an extract of the results for each scenario, without (and then with) the adaptive filtering mechanism, by focusing again on the variable x_1. Overall, the results are comparable to the previous case study. Without the aging mechanism, we can observe a very slow inference process that leads in many cases to wrong verification outcomes. Scenario (1) in Figure 8(a) introduces a long-lived perturbation with substantial magnitude. Here, we can see that using credible intervals we detect a transient behavior from observation 375. Scenario (2) in Figure 8(b) is even worse since the magnitude of the change is smaller. In this case, both interval and point estimates remain inside the constraint. Equilibrium violations are also not detected in scenario (3) in Figure 8(c). As shown in Figure 8(d), the adaptive aging mechanism makes the inference process faster when the perturbation occurs. Here, we notice that a transient behavior is detected from observation 308. In scenario (2) in Figure 8(e), we can see that the point estimates detect the violation from observations 315 to 323. Nevertheless, the uncertainty is substantial and after observation 323, the violation is no longer detected although the actual value of x_1 remains outside the constraint. In scenario (3) in Figure 8(f), we can see that even under a small and short-lived perturbation, we are able to detect the presence of transient behavior right after the injection by means of interval estimates (from observation 306).

RQ1 Summary: According to our results with the two case studies, our approach is able to detect violations of the equilibrium constraint in all of the selected scenarios. The adaptive aging mechanism makes the inference process able to promptly react to changes occurring in the semantic space. The credible intervals quantify the uncertainty and avoid wrong verification outcomes obtained through point estimates.

5.4.2 RQ2 (Violation Detection Effectiveness). To deepen the investigation carried out in the context of RQ1, we carried out additional controlled experiments injecting different perturbations into the uncertain regions by varying the magnitude and the duration. We controlled the *magnitude* of the violations, from small (0.01) to substantial (0.12), and the *duration*, from short (25) to long (100). For each perturbation, we executed our two systems 100 times; for each run, we measured the *detection effectiveness* of the equilibrium verification, as follows.



(a) $m = 0.1, d = \infty$, no-aging. (b) $m = 0.01, d = \infty$, no-aging. (c) $m = 0.01, d = 25$, no-aging.



(d) $m = 0.1, d = \infty$, aging. (e) $m = 0.01, d = \infty$, aging. (f) $m = 0.01, d = 25$, aging.

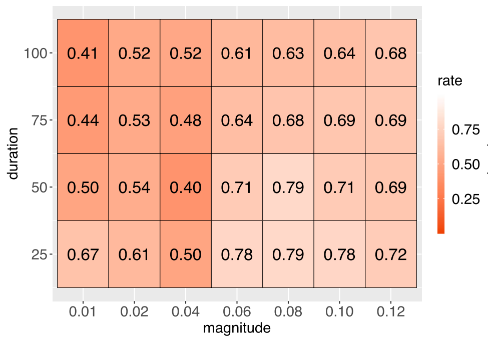
Fig. 8. Equilibrium verification over time in the UAV system under different scenarios.

Metric 1 (Detection Effectiveness). We measure the detection effectiveness as the *rate of the estimates* falling outside the equilibrium constraints during a perturbation (i.e., the higher the rate, the higher the effectiveness).

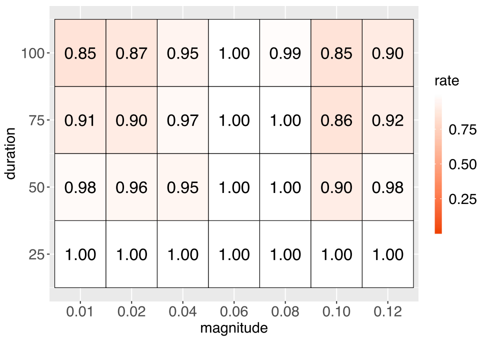
Figure 9 shows the major results using a number of heat maps illustrating the average effectiveness measured over 100 runs of the two case studies using point and interval estimates.

Case study 1 (search-and-rescue robot). Figures 9(a) and 9(b) illustrate the major results obtained with point and interval estimates, respectively. Interval estimates yield higher effectiveness compared with point estimates, that is, ~35% higher rate with interval estimates on average. The shape of the perturbation (i.e., magnitude and duration) has a strong impact on effectiveness when using point estimates. Specifically, small magnitude and duration are likely to decrease the rate. The adoption of interval estimates alleviates this negative effect. We can observe that the usage of interval estimates substantially improves the effectiveness, especially when the injected perturbation has a small magnitude. On average, we measured an improvement from 25% to 43% when reducing the magnitude of the perturbation from 0.12 to 0.01 (i.e., the smaller the magnitude, the higher the effectiveness of interval estimates).

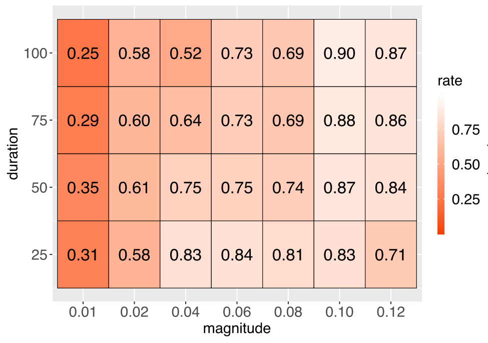
Case study 2 (team of UAVs). According to the measurements in Figures 9(c) and 9(d), the average detection effectiveness is higher with interval estimates (~95%) compared with point estimates (~68%). Consistent with the previous results, the magnitude of the perturbation has a strong impact on the effectiveness when using point estimates. Furthermore, interval estimates improve the effectiveness, especially under small perturbations. By using interval estimates, we can observe an average improvement from 18% to 56% when reducing the magnitude of the perturbation from 0.12 to 0.01.



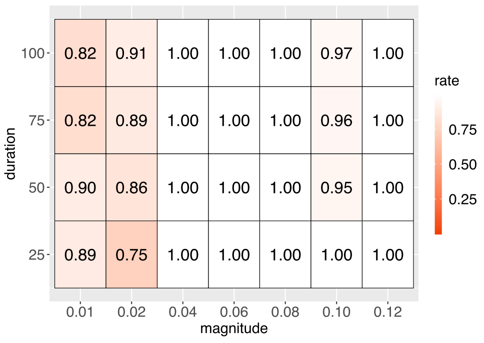
(a) Search rescue robot with point estimates.



(b) Search rescue robot with interval estimates.



(c) UAVs with point estimates.



(d) UAVs with interval estimates.

Fig. 9. Detection effectiveness of interval versus point estimates.

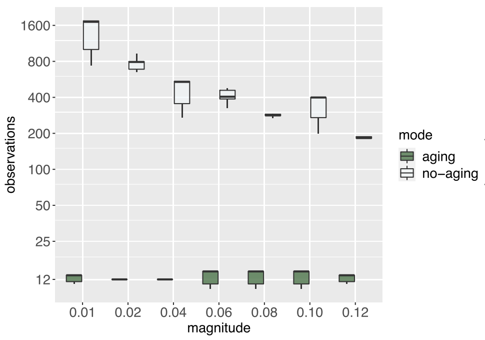
RQ2 Summary: According to our experience with the two case studies, the average effectiveness is consistently higher when using interval estimates, especially when the injected perturbation has a small magnitude. We observed that the smaller the magnitude, the higher the effectiveness of the interval estimates compared with point estimates.

5.4.3 RQ3 (Violation Detection Latency). To address this question, we conducted additional controlled experiments with our two case studies by injecting perturbations having unbounded duration. We varied the magnitude factor from small (0.01) to substantial (0.12), as described in the context of RQ2. Each scenario has been repeated 100 times for each case study. In each run, we executed our runtime equilibrium verification method with and then without the adaptive observation aging mechanism (i.e., aging and no aging). For each run, we measured the *latency* of the detection, as follows.

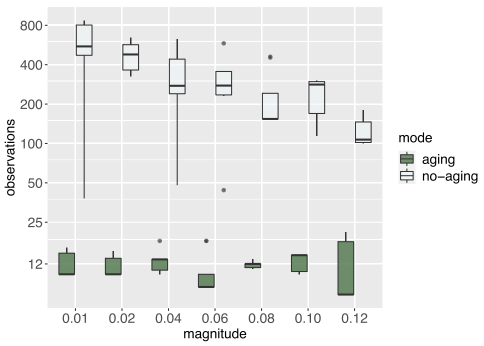
Metric 2 (Detection Latency). We measure the detection latency as the *number of observations* required to detect equilibrium violations after the injection of a perturbation (i.e., the lower the number of observations, the lower the latency).

Figure 10 shows the results of our experiments through a box plot illustrating the latency measured with and then without the adaptive aging mechanism.

Case study 1 (search and rescue robot). Figure 10(a) shows the results for the search-and-rescue robot case study. We can observe that the latency measured without the aging mechanism is



(a) Rescuerobot system.



(b) UAVs system.

Fig. 10. Latency of the equilibrium verification in terms of number of observations (log scale).

always higher. The magnitude of the perturbation has a strong impact on the latency in the no-aging case. In this latter case, the median latency ranges from ~1600 (magnitude 0.01) to ~200 (magnitude 0.12). According to the relative distance between median points in Figure 10, adaptive aging decreases the latency from ~94% to ~99%.

Case study 2 (team of UAVs). The results for the team of the UAVs case study are reported in Figure 10(b). The latency without the aging mechanism is always higher and the magnitude of the perturbation has a strong negative impact in the no-aging case. We can observe a median latency ranging from ~600 to ~100 when the magnitude changes from 0.01 to 0.12. In this case, the median latency decreases from ~88% to ~98%.

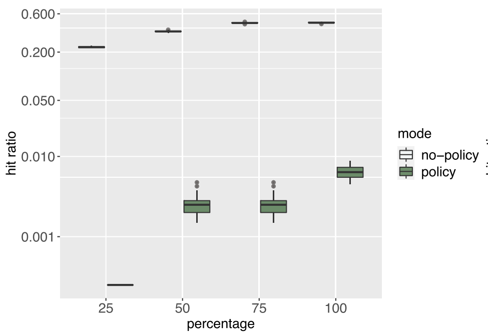
RQ3 Summary: The results of our experiments show that the latency is always lower when using the adaptive observation aging mechanism. The magnitude of the perturbations has a strong negative impact on the latency in the no-aging case. We observed the following trend: the smaller the magnitude, the better the aging option compared with no aging.

5.4.4 RQ4 (Equilibrium Enforcement Effectiveness). We addressed this question through experiments in which we controlled the constraints, causing equilibrium violations in selected model regions. As shown in Table 4, the specifications of the search-and-rescue robot contain 4 uncertain regions, whereas the specifications of the UAVs contain 8 uncertain regions in total. For each case study, we caused violations in 25%, 50%, 75%, and 100% of the uncertain regions. We then executed each case study 100 times for each percentage with and then without the equilibrium enforcement mechanism. In each run, we measured the *enforcement effectiveness*, as follows.

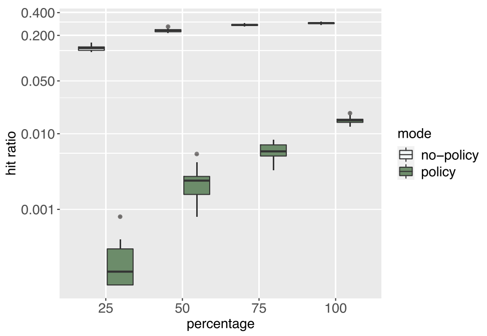
Metric 3 (Enforcement Effectiveness). We measure the enforcement effectiveness as the *hit ratio* of the uncertain regions violating the equilibrium, that is, the number of times the regions violating the constraints are executed divided by the total number of executed regions (i.e., the less the hit ratio, the higher the enforcement effectiveness).

Figure 11 shows the results of our experiments through a box plot illustrating the hit ratio with and then without the equilibrium enforcement mechanism.

Case study 1 (search-and-rescue robot). Figure 11(a) shows the hit ratio we measured with the search-and-rescue robot case study executed 100 times with and then without the equilibrium enforcement policy. Overall, we can observe that the hit ratio with the enforcement policy is always



(a) Rescuerobot system.



(b) UAVs system.

Fig. 11. Effectiveness of the equilibrium enforcement mechanism in terms of hit ratio (log scale).

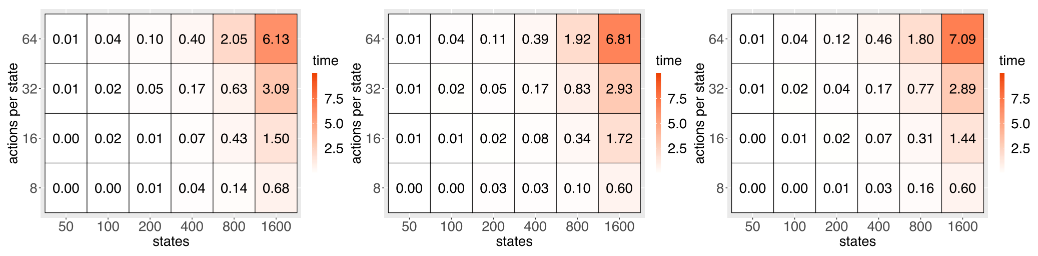
lower, meaning that the enforcement mechanism is effective in avoiding those regions violating the equilibrium. Intuitively, the number of regions violating the equilibrium is likely to affect the hit ratio. Specifically, the higher the percentage, the higher the hit ratio. Without the enforcement mechanism, the median hit ratio ranges from ∼0.23 to ∼0.46 when varying the percentage from 25% to 100%. The enforcement mechanism reduces the hit ratio up to three orders of magnitude. In this latter case, the median ratio ranges from ∼0.0002 to ∼0.0063.

Case study 2 (team of UAVs). We replicated the same experiments with the second case study. The hit ratio is reported in Figure 11(b). We can observe that the hit ratio is again lower when using the enforcement policy. Without the enforcement mechanism, the median hit ratio ranges from ∼0.13 (percentage 25%) to ∼0.29 (percentage 100%). Using the enforcement mechanism, the median ratio ranges from ∼0.0002 to ∼0.015.

RQ4 Summary: The results of our experiments show that the enforcement mechanism is effective in reducing the likelihood of observing equilibrium violations. The hit ratio generally increases with the percentage of regions violating the constraints. According to our results, the enforcement policy reduces the hit ratio up to three orders of magnitude.

5.4.5 RQ5 (Equilibrium Enforcement Scalability). We conducted additional experiments to study the scalability of the runtime stages by increasing the structural complexity of the pMDP specifications. As discussed in Section 4, model exploration at runtime is required only for the computation of the optimal equilibrium policy that makes use of value iteration. Thus, in the following, we show the extent to which the latter procedure is scalable.

As reported in Table 5, to answer RQ5, we used 24 synthetically generated pMDPs controlling the structural complexity in terms of the number of states (from 50 to 1, 600) and actions per state (from 8 to 64). The order of magnitude of the structural complexity varies from 10^2 to 10^5, as reported in Table 4. We also controlled the percentage of the regions violating the equilibrium (from 25% to 75%). For each selected synthetic system and percentage value, we computed the equilibrium enforcement policy 100 times and measured the cost in terms of wall-clock time. Figure 12 shows the results of our experiments through a number of heat maps illustrating the average wall-clock time (seconds). The results show that the cost is negligible even when increasing the structural complexity. The measured wall-clock time is always less than 1 second up to 800 states and 32 actions per state (25, 600 structural elements). In the worst case,1, 600 states and 64 actions per state (102, 400 structural elements), the time required is 6.67 seconds on average. The results obtained



(a) Disequilibrium in 25% regions. (b) Disequilibrium in 50% regions. (c) Disequilibrium in 75% regions.

Fig. 12. Cost of computing the resilience enforcement policy in terms of average wall-clock time (seconds).

using different percentage values (25% in Figure 12(a), 50% in Figure 12(b), 75% in Figure 12(c)) do not yield specific trends. This means that the percentage of regions violating the equilibrium is not likely to affect the cost of calculating the equilibrium enforcement policy.

RQ5 Summary: Overall, we observed a negligible cost even when increasing the structural complexity. According to our results, the wall-clock time is always less than 1 second up to $25,600$ structural elements, whereas it increases up to 6.67 seconds with $102,400$ structural elements. The percentage of regions violating the equilibrium is not likely to affect the cost of calculating the equilibrium enforcement policy.

6 RELATED WORK

The resilience property in CPSs and self-adaptive software has been studied in recent years to increase the level of assurance of these systems, which are increasingly expected to continuously exhibit acceptable behavior even in the case of unexpected events. Recent advances from the community of self-adaptive systems aim to model systems taking into account the sources of uncertainty. Taxonomies describing common sources, types, occurrence, impact of uncertainty, and mitigating strategies include the work by Ramirez et al. [30] and Esfahani and Malek [16]. Other recent taxonomies more focused on resilience assessment under uncertainty are presented in [7, 8]. According to these lworks, there exists a lack of resilience verification methods embedding systematic approaches quantifying and mitigating existing sources of uncertainty.

In the following, we describe relevant related work by focusing on existing research aimed at endowing self-adaptive software with continuous assurance mechanisms tailored to verification and/or enforcement of dependability requirements in the presence of changes and uncertainty. Table 6 summarizes the main approaches specifically targeting uncertainty and resilience of self-adaptive systems by comparing them with RUNE^2 according to the following characteristics:

- Formalisms adopted for system and requirements modeling, plus analysis techniques
- Software life-cycle phase applicability: design-time, development (or preproduction) when the system executes in a testing environment, or operation while the system executes in its final operational environment
- Target scope or purpose (diagnosis only, quantifying, policy enforcement, etc.)
- approach evaluation study (efficacy, effectiveness, scalability, performance, etc.).

Existing frameworks based on Petri Nets formalisms [31, 38] aim to specify and verify timed robustness properties with different temporal semantics for mandatory and optional uncertain timed events. These approaches support design-time verification of adaptation procedures and are supported by software tools such as ZAFETY [31]. Calinescu et al. [27] introduced the

Table 6. Qualitative Comparison of Related Approaches

Approach	Formalisms and technique	Phase	Purpose	Evaluation Study
ZAFETY [31]	• Zone-based Time Basic Petri nets • Exhaustive exploration of the Time Reachability Graph	Design-time	Verification of timed safety, liveness, and robustness properties	Efficacy and performance
FACT [27]	• Discrete-time PMC with PCTL • Frequentist statistics	Preproduction	Quantitative verification with synthesis of confidence intervals for reliability, performance, and other QoS	Efficacy, effectiveness, and performance
KAMI [32]	• DTMC and CTMC with PCTL • Bayesian estimation	• Preproduction • Operation	Continuous assurance of reliability and performance properties	Performance
COVE [33]	• DTMC with PCTL • Bayesian estimation with aging mechanism	• Preproduction • Operation	Runtime verification of nonfunctional properties	Effectiveness and scalability
LAF [29]	• DTMC with PCTL • Bayesian estimation with aging mechanism	• Preproduction • Operation	Runtime verification of nonfunctional properties	Estimation accuracy and time overhead
Cámara and de Lemos [34]	• DTMC with PCTL • Stimulus generation, model generation, probabilistic model checking	• Preproduction • Operation	pProvide levels of confidence regarding service delivery	Estimation accuracy
PLA [35]	• MDP with probabilistic reward computation-tree logic (PRCTL) • Probabilistic model checking	• Preproduction • Operation	Proactive latency-aware adaptation under uncertainty	Adaptation latency
ENTRUST [36]	• DTMCs and CTMCs, MDPs and probabilistic automata augmented with PCTL • Synthesis of dynamic assurance arguments, probabilistic model checking	• Preproduction • Operation	Runtime verification of assurance cases	Correctness, effectiveness, efficiency, and generality
METRIC [37]	• MDP with PCTL • On-the-fly model-based testing, Bayesian inference, and point estimates	• Preproduction	Runtime verification of nonfunctional requirements under uncertainty	Accuracy and performance
RUNE [12]	• pMDP with PCTL • Bayesian inference	• Preproduction • Operation	Runtime verification of the *equilibrium* property	Efficacy, effectiveness, and latency
RUNE^2 (this work)	• pMDP with rewards, PCTL • Bayesian inference, credible intervals, and optimal policy	• Preproduction • Operation	Runtime verification and enforcement of the equilibrium property	Efficacy, effectiveness, latency, and scalability

so-called Formal verificAtion with Confidence inTervals (FACT) approach to assure quality properties (reliability, performance, and others) of systems that exhibit stochastic behavior. This mathematical framework aims to establish confidence intervals for the quality properties of a software system modeled as a Parametric Markov Chain (PMC) with uncertain parameters, that is, a Markov chain for which unknown transition probabilities are specified as variables and their observations are considered by adopting frequentist statistics rather than Bayesian. The aforementioned methods are tailored to design-time or preproduction analysis using computationally intensive verification techniques, whereas dynamically assured resilience requires the usage of models at runtime as well as efficient runtime verification techniques that can be seamlessly deployed along with the target system in production [7].

Bayesian reasoning to perform statistical inference and runtime calibration of the model's transition probabilities (which may be unknown or subject to change) from the observation of a running system has recently gained high interest because it provides a natural and principled way of combining prior information with observations. The approach KAMI defined by Epifani et al. [32] applies Bayesian inference to calibrate the transition probabilities of a DTMC kept alive along with the running system in production. Improvements have been proposed by Calinescu et al. [33] with the approach COntinual VErification (COVE) and by Filieri et al. [29] with the approach Lightweight Adaptive Filtering (LAF) to alleviate the negative effect of historical data on the estimation by using aging mechanisms (e.g., Kalman filters [39]) to discard old information.

The approach proposed by Cámara and de Lemos [34] addresses the verification of nonfunctional (resilience) properties for service-based systems. The approach aims to build a DTMC model representing the system's response to changes occurring in the environment. It relies on stimulation and probabilistic model checking to provide levels of confidence regarding trustworthy services delivered when the system undergoes adaptation as a consequence of changes.

Proactive latency-aware adaptation (PLA) [35] is an approach that addresses the limitations of reactive adaptation by using a "look-ahead horizon" to proactively adapt, taking into account not only the current conditions but also their possible evolution. Adaptation decisions in the model are left underspecified through nondeterminism; a probabilistic model checker then resolves the nondeterministic choices by taking into account latency and possible conflicts between them.

ENgineering of TRUstworthy Self-adaptive sofTware (ENTRUST) is another related approach by Calinescu et al. [36] that uses dynamic safety cases to verify that a target self-adaptive software continues to safely achieve its goals during the adaptation process.

Camilli et al. [37] have introduced a model–based hypothesis testing approach called METRIC to quantify and mitigate software system uncertainty during testing by combining on–the–fly model–based testing and Bayesian inference.

The applicability of these approaches to CPSs requires significant extension effort due to the physical aspects of these systems and their equilibrium goals. The RUNE^2 approach differs from these previous approaches since it leverages a quantified uncertainty (given in terms of credible intervals) for the runtime verification of equilibrium constraints of a CPS and provides a runtime enforcement mechanism to ensure that the running system remains within the boundaries of its viability zone.

7 DISCUSSION

In this section, we discuss the major strengths (Section 7.1) and limitations (Section 7.2) of RUNE^2 as well as how threats to validity have been mitigated (Section 7.3).

7.1 Strengths

Our evaluation shows the extent to which RUNE^2 can be effectively used to verify and enforce the satisfaction of system-level dependability requirements under changes in the semantic space. In our experience, we find that it has the following strengths.

Efficient and scalable runtime procedures. It is worth noting that even though parametric model checking requires exhaustive exploration of the state space to compute the equilibrium constraint, we keep this activity separate from the runtime stages of RUNE^2. The pre-computation of the constraints occurs offline, where we can usually execute demanding activities without interfering with the system's operation. We keep instead all the online stages as lightweight as possible. In particular, the Bayesian inference process is computationally inexpensive. Furthermore, the runtime equilibrium enforcement exhibits good scalability: the computation takes a few seconds even with 10^5 structural elements.

Uncertainty quantification using credible intervals. Our approach does not focus on average behavior through point estimates. Especially under changes in the semantic space, we show that point estimates are likely to fail in capturing equilibrium violations. By computing the highest density regions (i.e., the credible intervals) we essentially quantify the existing uncertainty so that verification is always aware of the highest possible accuracy of the estimation process.

Asymptotic stability. Asymptotic stability of our inference process is defined in terms of the relative distance between actual and estimated values for any given constant input, regardless of the initial estimates. According to [39], the method is asymptotically stable if the limit of such a difference converges to zero (within a convenient accuracy) as time tends to infinity. Under this condition, the interval estimates will eventually include the actual values because of the asymptotic behavior of the posterior in the limit of infinite observations. That is, Bayesian inference is consistent and the posterior converges to a distribution independent of the initial prior [40]. In this case, the indicator function in Equation (18) is equal to 1; therefore, the filter reduces to Bayesian inference summarized using credible intervals.

Fast detection of changes. A well-known issue in Bayesian inference is the possible negative effect of historical data. The inference process becomes slower to detect changes as the number of observations increases. We mitigated this issue through a lightweight adaptive filtering approach that discards old observations when it detects a high degree of dispersion around the estimated values. On the one hand, we show that this aging mechanism reduces the latency of detecting equilibrium violations. On the other hand, there is the risk of causing oscillatory behavior under nonstationary input. Nonetheless, the extent to which our method detects changes depends on the accuracy of the inference process (the magnitude of the credible intervals), that is, when the indicator function in Equation (18) is equal to 0, the estimates in the last observation period contradict the hypothesis made with 95% credibility according to the evidence. Very small intervals increase the sensitivity. However, sensitivity does not mean instability [39]. Furthermore, when incoming measurements cannot be reasonably explained under the current credible intervals, we decrease the level of confidence (i.e., we adopt larger credible intervals). This reduces the risk of obtaining oscillatory outcomes of the verification procedure under ongoing changes in the input.

Effective enforcement of the resilience property. RUNE^2 exploits the outcome of the runtime equilibrium verification to avoid future violations, thus enforcing the resilience property. Our experiments show that the enforcement mechanism is effective in reducing the likelihood of hitting violating regions. Specifically, the application of the enforcement policy reduces the likelihood up to three orders of magnitude.

Fully automated runtime verification and enforcement. The runtime stages of RUNE^2 are fully automated by our software toolchain. The stages (iii) and (iv) work at runtime along with the managed system (either for testing purposes or after the deployment in production).

7.2 Limitations

In the following, we briefly summarize the main limitations emerging both from the evaluation and from our experience.

Definition of the specifications and requirements. The offline stages of RUNE^2 require manual effort by the engineers who specify the shared phenomena of interest through the pMDP modeling formalism and the system-level requirements through the PCTL language. This requires modeling skills as well as substantial knowledge of the formal notations adopted by our approach. Existing approaches in model-driven engineering may be adopted to mitigate this concern. For instance, model-to-model transformation allows engineers to specify higher-level source models (or meta-models) as well as automated mappings to create one or more target lower-level models [25, 32]. As an example, the METRIC approach [25] transforms a UML activity diagram into a Markov Decision Process. It is also worth noting that large specifications usually model parametric problems, meaning that a number of small portions of the specifications are automatically generated from templates. As an example, the UAVs in Figure 6(b) is a parametric problem [17] instantiated by generating and connecting smaller pMDPs depending on the number of route segments and altitude levels. Another possible way to mitigate this limitation is the usage of existing user-friendly modeling tools. To simplify the formulation of probabilistic properties, engineers usually make use of property specification patterns (expressed in structured natural language) that can be then customized by using off-the-shelf tools, such as PSPWizard [41].

Nontrivial definition of accurate priors. The absence of prior knowledge (i.e., uninformative priors) may slow down the detection of equilibrium violations, especially in a cold start. In this case, RUNE^2 recognizes the presence of transient behavior but could fail at detecting violations as long as the credible intervals and constraints are not disjoint. The definition of accurate priors may mitigate this limitation. This activity can be supported by existing tools for interactive (machine-assisted) elicitation of prior density functions [42].

Lack of explanations. RUNE^2 cannot be used to provide operators with human interpretable explanations for the (dis)equilibrium condition. This may hinder the ability to trace unsuccessful adaptations back to understandable root causes. Explainability in self-adaptation [43] is emerging as a crucial aspect to achieve the ultimate goal of developing systematic engineering approaches to overcome the lack of transparency, thus helping engineers better understand the failures, localize them, and improve the system's dependability.

7.3 Threats to Validity

In considering the work in [44], we took into account the traditional categories of validity threats: *construct*, *conclusion*, *internal*, and *external*. Thus, potential threats to the validity of our approach and our evaluation have been addressed as follows.

Construct validity. Threats in this category may arise due to assumptions made when modeling the system from our case study. To mitigate these threats, we used models and system-level dependability requirements based on established case studies from the research literature. We conducted our evaluation using well-known representative systems in the community of software engineering for self-adaptive systems: the search-and-rescue robot [12, 16] and the team of UAVs [17, 18].

Conclusion validity. We mitigated conclusion validity threats by reducing the possibility of obtaining results by chance [45]. This threat exists as the behavior of the simulated systems is governed by probabilistic functions. Following the guideline in [46], each experiment executed to answer our research questions has been repeated 100 times.

Internal validity. Threats may be caused by bias in establishing cause–effect relationships in our experiments. To limit these threats, we assessed RUNE^2 through extensive simulation campaigns and considering different verification scenarios by directly controlling a number of factors of interest listed in Table 5 for each research question. Such a fine-grained access to factors increases internal validity compared with observations without manipulation. Direct manipulation of factors has been crucial to create the same experimental conditions within repeated runs, and, therefore, systematically assess cause–effect relations emerging from our results.

External validity. Threats may exist if the characteristics of the system in our case study are not indicative of the characteristics of other systems. We mitigated these threats by considering case studies from the literature with nontrivial semantic spaces including, a wide range of multidimensional changes. In addition, RUNE^2 supports the verification of systems whose behavior is modeled using a pMDP with requirements specified in PCTL. Parametric Markov models have been widely used in recent years to model uncertain conditions in different application domains (e.g., [36, 47–49]). The PCTL language represents the de facto standard to specify properties of discrete-time Markov models [36]. As described in [24], Bayesian inference is the statistical framework of choice when uncertain assumptions need to be updated as more evidence becomes available. Our experiments were conducted on a simulation basis. We designed a set of controlled experiments using an open-source simulator.[6] The simulator instruments both controllable and observable events of a target Java program behaving according to a given pMDP specification. Generalization of our results to real CPSs "on the field" (e.g., actual robotic systems) requires additional experiments that we postponed as future work. Another external validity threat may arise if the considered software systems require the use of larger models than those used in our evaluation. To mitigate this threat, we conducted a scalability study by considering synthetically generated pMDP specifications with a number of states up to 1,600 and a total number of structural elements up to 102,400.

8 CONCLUSION AND FUTURE DIRECTIONS

In this article, we presented RUNE^2, a method tailored to runtime equilibrium verification and enforcement for resilient CPSs. RUNE^2 relies on a pMDP specification that captures uncertainties whose source could be partial knowledge or possible changes in the environment. To verify the resilience property, RUNE^2 pre-computes the equilibrium constraints that collectively define the boundaries of the viability zone for the system. Then, it applies a runtime verification step that exploits Bayesian inference and credible intervals to quantitatively reason on the ability of the system to remain inside such boundaries. Finally, it enacts a resilience enforcement mechanism that enforces the system inside its viability zone by means of the optimal policy that maximizes the notion of equilibrium reward in the long run.

We evaluated RUNE^2 on two well-known case studies in the research community of software engineering for self-adaptive systems (the search-and-rescue robotic system, and the team of UAVs) as well as 24 systems synthetically generated from pseudo-random pMDP specifications with increasing structural complexity. We demonstrated the ability in detecting equilibrium violations, the effectiveness compared with standard inference methods based on point estimates, the capability in enforcing the system in its viability zone, and the scalability.

We are currently investigating methods able to provide strong, ideally provable, assurances for resilient CPSs through extensive testing activities. We are developing an *equilibrium falsification* technique that aims to minimize the equilibrium by changing the parameters in the semantic space.

[6] https://github.com/SELab-unimi/mbt-module.

The testing process is driven by the specifications and uses *metaheuristic optimization* to find those circumstances under which the behavior falsifies system-level PCTL dependability requirements. We are also planning additional experimental campaigns using real robotic systems to further generalize our findings.

REFERENCES

[1] Matteo Camilli, Angelo Gargantini, Patrizia Scandurra, and Carlo Bellettini. 2017. Towards inverse uncertainty quantification in software development (short paper). In *Proceedings of Software Engineering and Formal Methods - 15th International Conference (SEFM'17), Trento, Italy, September 4–8, 2017 (Lecture Notes in Computer Science)*, Alessandro Cimatti and Marjan Sirjani (Eds.), Vol. 10469. Springer, Berlin, 375–381. DOI: http://dx.doi.org/10.1007/978-3-319-66197-1_24

[2] J. O. Kephart and D. M. Chess. 2003. The vision of autonomic computing. *Computer* 36, 1 (2003), 41–50. DOI: http://dx.doi.org/10.1109/MC.2003.1160055

[3] Danny Weyns, Bradley Schmerl, Vincenzo Grassi, Sam Malek, Raffaela Mirandola, Christian Prehofer, Jochen Wuttke, Jesper Andersson, Holger Giese, and Karl M. Göschka. 2013. *On Patterns for Decentralized Control in Self-Adaptive Systems*. Springer, Berlin, 76–107. DOI: http://dx.doi.org/10.1007/978-3-642-35813-5_4

[4] V. Gunes, Steffen Peter, T. Givargis, and F. Vahid. 2014. A survey on concepts, applications, and challenges in cyber-physical systems. *KSII Trans. Internet Inf. Syst.* 8 (2014), 4242–4268.

[5] Edward Lee. 2015. The past, present and future of cyber-physical systems: A focus on models. *Sensors (Basel, Switzerland)* 15 (03 2015), 4837–4869. DOI: http://dx.doi.org/10.3390/s150304837

[6] Lorenzo Pagliari, Raffaela Mirandola, and Catia Trubiani. 2020. Engineering cyber-physical systems through performance-based modelling and analysis: A case study experience report. *Journal of Software: Evolution and Process* 32, 1 (2020). DOI: http://dx.doi.org/10.1002/smr.2179

[7] A. Bennaceur, C. Ghezzi, K. Tei, T. Kehrer, D. Weyns, R. Calinescu, S. Dustdar, Z. Hu, S. Honiden, F. Ishikawa, Z. Jin, J. Kramer, M. Litoiu, M. Loreti, G. Moreno, H. Muller, L. Nenzi, B. Nuseibeh, L. Pasquale, W. Reisig, H. Schmidt, C. Tsigkanos, and H. Zhao. 2019. Modelling and analysing resilient cyber-physical systems. In *Proceedings of the IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'19)*. 70–76. DOI: http://dx.doi.org/10.1109/SEAMS.2019.00018

[8] Jesper Andersson, Vincenzo Grassi, Raffaela Mirandola, and Diego Perez-Palacin. 2020. A conceptual framework for resilience: Fundamental definitions, strategies and metrics. *Computing* (12 2020), 1–30. DOI: http://dx.doi.org/10.1007/s00607-020-00874-x

[9] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl E. Landwehr. 2004. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing* 1, 1 (2004), 11–33.

[10] Diego Perez-Palacin and Raffaela Mirandola. 2014. Uncertainties in the modeling of self-adaptive systems: A taxonomy and an example of availability evaluation. In *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering (ICPE'14)*. ACM, New York, NY, 3–14. DOI: http://dx.doi.org/10.1145/2568088.2568095

[11] Jean-Pierre Aubin, Alexandre Bayen, and Patrick Saint-Pierre. 2011. *Viability Theory: New Directions*. Springer Berlin, Heidelberg, 830. DOI: http://dx.doi.org/10.1007/978-3-642-16684-6

[12] Matteo Camilli, Raffaela Mirandola, and Patrizia Scandurra. 2021. Runtime equilibrium verification for resilient cyber-physical systems. In *2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS'21)*. 71–80. DOI: http://dx.doi.org/10.1109/ACSOS52086.2021.00025

[13] Michael Jackson. 1995. The world and the machine. In *Proceedings of ICSE 1995*. ACM, 283–292. DOI: http://dx.doi.org/10.1145/225014.225041

[14] C. Ghezzi. 2017. Of software and change. *Journal of Software: Evolution and Process* 29, 9 (2017), e1888. DOI: http://dx.doi.org/10.1002/smr.1888

[15] Rogério de Lemos, David Garlan, Carlo Ghezzi, and Holger Giese. 2013. Software engineering for self-adaptive systems: Assurances (Dagstuhl seminar 13511). *Dagstuhl Reports* 3, 12 (2013), 67–96.

[16] Naeem Esfahani and Sam Malek. 2013. Uncertainty in self-adaptive software systems. In *Software Engineering for Self-Adaptive Systems II*. Springer, Berlin, 214–238. DOI: http://dx.doi.org/10.1007/978-3-642-35813-5_9

[17] Gabriel Moreno, Cody Kinneer, Ashutosh Pandey, and David Garlan. 2019. DARTSim: An exemplar for evaluation and comparison of self-adaptation approaches for smart cyber-physical systems. In *2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'19)*. 181–187. DOI: http://dx.doi.org/10.1109/SEAMS.2019.00031

[18] Ashutosh Pandey, Ivan Ruchkin, Bradley Schmerl, and David Garlan. 2020. Hybrid planning using learning and model checking for autonomous systems. In *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS'20)*. 55–64. DOI: http://dx.doi.org/10.1109/ACSOS49614.2020.00026

[19] Martin L. Puterman. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming* (1st ed.). John Wiley & Sons, Inc., New York, NY.

[20] Ernst Moritz Hahn, Tingting Han, and Lijun Zhang. 2011. Synthesis for PCTL in parametric Markov decision processes. In *Proceedings of NFM'11*. Springer, Berlin, 146–161.

[21] Costas Courcoubetis and Mihalis Yannakakis. 1995. The complexity of probabilistic verification. *Journal of the ACM* 42, 4 (July 1995), 857–907. DOI: http://dx.doi.org/10.1145/210332.210339

[22] Richard Bellman. 1952. On the theory of dynamic programming. *Proceedings of the National Academy of Sciences* 38, 8 (1952), 716–719. DOI: http://dx.doi.org/10.1073/pnas.38.8.716

[23] J. O. Berger. 1985. *Statistical Decision Theory and Bayesian Analysis*. Springer.

[24] Christian P. Robert. 2007. *The Bayesian Choice: From Decision-Theoretic Foundations to Computational Implementation* (2nd ed.). Springer.

[25] Matteo Camilli, Angelo Gargantini, and Patrizia Scandurra. 2020. Model-based hypothesis testing of uncertain software systems. *Software Testing, Verification and Reliability* 30, 2 (2020), e1730. DOI: http://dx.doi.org/10.1002/stvr.1730 e1730 stvr.1730.

[26] Ernst Moritz Hahn, Holger Hermanns, Björn Wachter, and Lijun Zhang. 2010. PARAM: A model checker for parametric Markov models. In *Computer Aided Verification*, Tayssir Touili, Byron Cook, and Paul Jackson (Eds.). Springer, Berlin, 660–664.

[27] Radu Calinescu, Carlo Ghezzi, Kenneth Johnson, Mauro Pezzè, Yasmin Rafiq, and Giordano Tamburrelli. 2016. Formal verification with confidence intervals to establish quality of service properties of software systems. *IEEE Transactions on Reliability* 65, 1 (2016), 107–125.

[28] Radu Calinescu, Kenneth Johnson, and Yasmin Rafiq. 2011. Using observation ageing to improve markovian model learning in QoS engineering. In *Proceedings of the 2nd ACM/SPEC ICPE*. ACM, New York,NY, 505–510. DOI: http://dx.doi.org/10.1145/1958746.1958823

[29] Antonio Filieri, Lars Grunske, and Alberto Leva. 2015. Lightweight adaptive filtering for efficient learning and updating of probabilistic models. In *Proceedings of ICSE'15*. IEEE, 200–211.

[30] Andres J. Ramirez, Adam C. Jensen, and Betty H. C. Cheng. 2012. A taxonomy of uncertainty for dynamically adaptive systems. In *Proceedings of SEAMS'12*. IEEE, 99–108.

[31] Matteo Camilli, Angelo Gargantini, and Patrizia Scandurra. 2018. Zone-based formal specification and timing analysis of real-time self-adaptive systems. *Science of Computer Programming* 159 (2018), 28–57. DOI: http://dx.doi.org/10.1016/j.scico.2018.03.002

[32] Ilenia Epifani, Carlo Ghezzi, Raffaela Mirandola, and Giordano Tamburrelli. 2009. Model evolution by run-time parameter adaptation. In *Proceedings of the 31st International Conference on Software Engineering (ICSE'09)*. IEEE Computer Society, Washington, DC, 111–121. DOI: http://dx.doi.org/10.1109/ICSE.2009.5070513

[33] Radu Calinescu, Yasmin Rafiq, Kenneth Johnson, and Mehmet Emin Bakir. 2014. Adaptive model learning for continual verification of non-functional properties. In *Proceedings of the 5th ACM/SPEC ICPE*. ACM, 87–98. DOI: http://dx.doi.org/10.1145/2568088.2568094

[34] Javier Cámara and Rogério de Lemos. 2012. Evaluation of resilience in self-adaptive systems using probabilistic model-checking. In *Proceedings of SEAMS 2012*. IEEE Computer Society, 53–62. DOI: http://dx.doi.org/10.1109/SEAMS.2012.6224391

[35] Gabriel A. Moreno, Javier Cámara, David Garlan, and Bradley Schmerl. 2015. Proactive self-adaptation under uncertainty: A probabilistic model checking approach. In *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE'15)*. ACM, New York, NY, 1–12. DOI: http://dx.doi.org/10.1145/2786805.2786853

[36] Radu Calinescu, Danny Weyns, Simos Gerasimou, Muhammad Usman Iftikhar, Ibrahim Habli, and Tim Kelly. 2018. Engineering trustworthy self-adaptive software with dynamic assurance cases. *IEEE Transactions on Software Engineering* 44, 11 (2018), 1039–1069.

[37] Matteo Camilli, Angelo Gargantini, and Patrizia Scandurra. 2020. Model-based hypothesis testing of uncertain software systems. *Software Testing, Verification and Reliability* 30, 2 (2020). DOI: http://dx.doi.org/10.1002/stvr.1730

[38] Lorenzo Capra and Matteo Camilli. 2018. Towards evolving petri nets: A symmetric nets-based framework. *IFAC-PapersOnLine* 51, 7 (2018), 480–485. DOI: http://dx.doi.org/10.1016/j.ifacol.2018.06.343 14th IFAC Workshop on Discrete Event Systems WODES 2018.

[39] Karl J. Astrom and Bjorn Wittenmark. 1990. *Computer-controlled Systems: Theory and Design (2nd Ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ.

[40] David A. Freedman. 1965. On the asymptotic behavior of Bayes estimates in the discrete case II. *The Annals of Mathematical Statistics* 36, 2 (1965), 454–456.

[41] Marco Autili, Lars Grunske, Markus Lumpe, Patrizio Pelliccione, and Antony Tang. 2015. Aligning qualitative, real-time, and probabilistic property specification patterns using a structured English grammar. *IEEE Transactions on Software Engineering* 41, 7 (2015), 620–638. DOI: http://dx.doi.org/10.1109/TSE.2015.2398877

[42] David E. Morris, Jeremy E. Oakley, and John A. Crowe. 2014. A web-based tool for eliciting probability distributions from experts. *Environmental Modelling & Software* 52 (2014), 1–4. DOI: http://dx.doi.org/10.1016/j.envsoft.2013.10.010

[43] Matteo Camilli, Raffaela Mirandola, and Patrizia Scandurra. 2023. XSA: EXplainable self-adaptation. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE'22)*. ACM, New York, NY, Article 189, 5 pages. DOI: http://dx.doi.org/10.1145/3551349.3559552

[44] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in Software Engineering*. Springer.

[45] Shuai Wang, Shaukat Ali, and Arnaud Gotlieb. 2013. Minimizing test suites in software product lines using weight-based genetic algorithms. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation (GECCO'13)*. ACM, New York, NY, 1493–1500. DOI: http://dx.doi.org/10.1145/2463372.2463545

[46] Andrea Arcuri and Lionel Briand. 2011. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *Proceedings of the 33rd International Conference on Software Engineering (ICSE'11)*. ACM, New York, NY, 1–10. DOI: http://dx.doi.org/10.1145/1985793.1985795

[47] M. Camilli, C. Bellettini, A. Gargantini, and P. Scandurra. 2018. Online model-based testing under uncertainty. In *2018 ISSRE*. 36–46. DOI: http://dx.doi.org/10.1109/ISSRE.2018.00015

[48] Matteo Camilli, Angelo Gargantini, Patrizia Scandurra, and Catia Trubiani. 2021. Uncertainty-aware exploration in model-based testing. In *14th IEEE Conference on Software Testing, Verification and Validation (ICST'21)*. 71–81. DOI: http://dx.doi.org/10.1109/ICST49551.2021.00019

[49] Matteo Camilli and Barbara Russo. 2020. Model-based testing under parametric variability of uncertain beliefs. In *Software Engineering and Formal Methods*, Frank de Boer and Antonio Cerone (Eds.). Springer International Publishing, Cham, 175–192. DOI: http://dx.doi.org/10.1007/978-3-030-58768-0_10

Received 29 March 2022; revised 26 September 2022; accepted 5 February 2023