SOFTWARE TESTING, VERIFICATION AND RELIABILITY
Softw. Test. Verif. Reliab. 2010; **00**:1–**??**
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/stvr

Model-based Hypothesis Testing of Uncertain Software Systems

Matteo Camilli[1*] Angelo Gargantini[2] and Patrizia Scandurra[2]

[1] *Faculty of Computer Science, Free University of Bozen-Bolzano, Bolzano, Italy*
[2] *Dept. of Management, Information, and Production Engineering, Università degli Studi di Bergamo, Bergamo, Italy*

SUMMARY

Nowadays there exists an increasing demand for reliable software systems able to fulfill their requirements in different operational environments and to cope with uncertainty that can be introduced both at design-time and at runtime due to the lack of control over third-party system components and complex interactions among software, hardware infrastructures and physical phenomena. This article addresses the problem of the discrepancy between measured data at runtime and the design-time formal specification by using an *Inverse Uncertainty Quantification* approach. Namely, we introduce a methodology called METRIC and its supporting toolchain to quantify and mitigate software system uncertainty during testing by combining (on-the-fly) *Model-based Testing* and *Bayesian inference*. Our approach connects probabilistic input/output conformance theory with statistical hypothesis testing in order to assess if the behavior of the system under test corresponds to its probabilistic formal specification provided in terms of a *Markov Decision Process*. An uncertainty-aware model-based test case generation strategy is used as a means to collect evidence from software components affected by sources of uncertainty. Test results serve as input to a Bayesian inference process that updates beliefs on model parameters encoding uncertain quality attributes of the system under test. This article describes our approach from both theoretical and practical perspectives. An extensive empirical evaluation activity has been conducted in order to assess the cost-effectiveness of our approach. We show that, under same effort constraints, our uncertainty-aware testing strategy increases the accuracy of the uncertainty quantification process up to 50 times with respect to traditional model-based testing methods.
Copyright © 2010 John Wiley & Sons, Ltd.

Received ...

KEY WORDS: Uncertainty Quantification, Formal Methods, Probabilistic Systems, Bayesian Inference, on-the-fly Model-based Testing.

1. INTRODUCTION

Nowadays there is a growing demand for reliable software systems able to fulfill their requirements in highly uncertain and changing operational environments. The negative impact of unreliable or unpredictable software cannot be tolerated as society increasingly depends on it to carry out tasks in many different application domains. However, predictability is very hard to achieve since modern software-intensive systems are often situated in complex ecosystems that can be hard to understand and specify completely at design-time. Thus, there exists an increasing need for systematic and effective approaches to deal with incomplete knowledge and uncertainty. According to Esfahani et al. [1], uncertainty can be introduced into the system by the system itself (e.g., uncertainty in the software structure or in the implementation of the algorithms) or its execution environment (e.g., uncertain inputs or operational profiles). In the past, software engineers used to abstract sources of uncertainty away because of insufficient understanding and unavailability of adequate methods

*Correspondence to: Matteo Camilli, P.zza Domenicani 3, 39100 Bolzano, Italy. E-mail: matteo.camilli@unibz.it

Copyright © 2010 John Wiley & Sons, Ltd.

and techniques. Today, endowing conventional software engineering methodologies with techniques and practices able to model, quantify, and manage uncertainty explicitly is becoming increasingly crucial [1, 2]. In particular, techniques that explicitly consider uncertainty in software testing is an emerging research direction.

This article focuses on the problem of uncertainty quantification and introduces a methodology able to deal with it by means of a *Model-based Hypothesis Testing* approach. Namely, we use statistical hypothesis testing to reduce the discrepancy between delivered products and initial design-time uncertainty assumptions. As described by Broy at al. [3], Model-based Testing (MBT) is a software testing technique where run-time behavior of a software System Under Test (SUT) is checked against predictions made by a model description of the system's behavior. In our approach, the design-time model description contains uncertain aspects explicitly specified by means of probability. Uncertainty must be mitigated accounting for evidence during the actual system's execution. Thus, we make use of (on-the-fly) MBT as a means to extract evidence from the SUT execution. This data is used to apply statistical hypothesis testing and infer a probability distribution describing the best knowledge on uncertain aspects. Specific methods and techniques used to achieve our goal of uncertainty quantification compose a methodology, so called METRIC[†]. The methodology follows an *Inverse Uncertainty Quantification* (IUQ) approach [4,5], meaning that it mainly reasons at runtime (on the discrepancy between evidence and the uncertain design-time assumptions) and then it propagates back the posterior knowledge to calibrate the initially uncertain design-time model. To this purpose, METRIC works by combining *Bayesian inference* [6] and (on-the-fly) model-based test case generation based on *infinite horizon* optimization algorithms [7], This approach allows the system state space to be explored in a controlled way by maximizing the probability to stress the uncertain components of the SUT. The Bayesian inference process updates the Posterior knowledge that encodes values of uncertain parameters. In our vision, new estimations out of the MBT activity, represent the basis of new verification phases and the prior knowledge for future evolutions of the software system. In METRIC, the modeling formalism of choice is Markov Decision Process (MDP) [7] and requirements are expressed in Probabilistic Computation Tree Logic (PCTL) [8]. This work assumes that uncertainty can be encoded to transition probabilities, modeling events or actions whose probability of occurrence becomes quantified and corrected by applying the METRIC methodology.

The whole methodology is supported by a software toolchain whose core component is a model-based hypothesis testing module which integrates test case generation, execution and evaluation by means of an *uncertainty-aware exploration policy*. An extensive evaluation activity of our METRIC toolchain implementation is reported in this article. Our major objective is to show the effectiveness of METRIC in statistical hypothesis testing (rather than functional testing) of uncertain software systems by measuring both the accuracy and the effort of the inference process. We show a comparative evaluation between our approach and traditional pseudorandom model-based test case generation algorithms, thus showing the convenience of METRIC.

This approach has been introduced by Camilli et al. [5, 9, 10] through a preliminary sketch of a testing method under uncertainty supported by a prototypal software implementation. Here we provide an extended presentation of the approach as part of a comprehensive methodology. Specifically, the contributions of this article can be summarized as follow: (i) characterization of the uncertainty quantification problem in software development; (ii) introduction of the METRIC methodology to model, quantify, and mitigate uncertainty in software systems; (iii) improved definition of the METRIC theoretical foundation; (iv) richer description of the software toolchain supporting METRIC; (v) description of additional empirical evaluation activities. The empirical evaluation shows that, on the one hand, under same effort constraints, our testing method increases the accuracy of the uncertainty mitigation up to 50 times with respect to traditional pseudorandom strategies. On the other one hand, under same termination conditions based on uncertainty mitigation, our testing method requires up to 80% less effort.

[†]METRIC stands for Modeling & vErification, Testing, Inference & Calibration

Copyright © 2010 John Wiley & Sons, Ltd.
Prepared using stvrauth.cls

DOI: 10.1002/stvr

The remainder of this article is structured as follows. First, in Section 2 we briefly recall the necessary background concepts and we present a characterization of the uncertainty quantification problem in software systems, thus providing a reference context where our approach can be classified. Section 3 introduces a preview of the METRIC methodology to IUQ. Section 4 presents a running example (i.e. a Tele Assistance System), used throughout the article to illustrate in details the main phases of the METRIC methodology. Section 5 describes the theoretical foundation of METRIC. Section 6 describes engineering aspects of the METRIC toolchain.Section 7 reports a broad set of experiments to assess the effectiveness of the IUQ process. It also provides discussion on both lessons learned by using METRIC in practice, and threats to validity. Section 8 describes related work. Finally, Section 9 concludes with a summary of the challenges ahead.

2. BACKGROUND AND PROBLEM STATEMENT

This section briefly recalls the necessary background concepts and a comprehensive characterization of the problem that has been addressed. The theoretical background includes: Markov Decision Processes, the temporal logic PCTL extended with rewards, Bayesian inference, and model-based testing of probabilistic systems.

2.1. Markov Decision Processes and Rewards

MDPs [7,11] represent a widely used formalism for modeling systems that exhibit both probabilistic and nondeterministic behavior. Formally, a MDP is a tuple $\mathcal{M} = (S, s_0, A, \delta, L)$ where:

- S is a finite set of states ($s_0 \in S$ initial state);
- A is a finite alphabet;
- $\delta : S \times A \to Dist(S)$[‡] is a (partial) probabilistic transition function;
- $L : S \to 2^{AP}$ is a labeling function mapping each state to a set of atomic propositions taken from a set AP.

Transitions between states occur in two steps: (i) a nondeterministic choice among the available actions $A(s) = \{a \in A : \exists \delta(s, a)\}$; and (ii) a random choice of the successor state s', according to the probability distribution δ, such that $\delta(s, a)(s')$ represents the probability that a transition to s' occurs. Note that δ satisfies $\sum_{s'} \delta(s, a)(s') = 1$, foreach s, a and successor state s'. The execution of an MDP model $\mathcal{M}$ can be identified by a sequence of transitions between states $E_{\mathcal{M}} = \{\langle s, s' \rangle\}_{s,s' \in S}$, such that $\exists a \in A(s) : \delta(s, a)(s') > 0$.

Nondeterminism is used in general to capture different aspects of system behavior (e.g. concurrency, underspecification, etc.). Our approach uses it to capture the possible ways that a controller (or decision maker) has to influence the behavior of the system by means of *controllable action*s. This terminology comes from the domain of testing, where certain operations are under the control of the tester, and certain operations are only *observable*. Intuitively, a sequence of controllable actions can be viewed as a *test* in the sense that it identifies a sequence of inputs to guide the execution of the system under test and verify that the produced outputs are predictable by the MDP model.

MDPs can be augmented with *reward*s to quantify a benefit (or loss) due to the sojourn in a specific state or to the occurrence of a certain state transition. A reward is a non-negative value assigned to states and/or transitions that can represent information such as average execution time, power consumption or usability. A reward structure associated with a MDP $\mathcal{M}$ is defined as a pair $r = (r_s, r_a)$ composed of a *state* reward function $r_s : S \to \mathbb{R}_{\geq 0}$ and an *action* reward function $r_a : S \times A \to \mathbb{R}_{\geq 0}$ that assigns rewards to states and transitions, respectively.

[‡] *Dist(S)* is the set of discrete probability distributions over a countable set S.

Copyright © 2010 John Wiley & Sons, Ltd.

Prepared using stvrauth.cls
DOI: 10.1002/stvr

Given a state s, there exist many paths connecting s to one of the final states (i.e., states having a single self-loop transition with probability 1). Given a reward structure, each of these paths cumulates as reward the sum of the rewards in the path. A common problem of MDPs is to find a policy function π that specifies the action $\pi(s)$ chosen by a decision maker when state s holds. The best (or optimal) policy π^* maximizes some function of the cumulated rewards, typically the expected discounted sum over a potentially infinite path. Namely, given a reward structure r, π^* can be computed solving a *dynamic decision problem* [7] as follows:

$$\pi^*(s) = \arg\max_{a \in A(s)} \sum_{s'} \delta(s,a)(s') \cdot (r_a(s,a,s') + \gamma V^*(s')) \tag{1}$$

where: $V^*(s')$ represents the expected cumulated reward when starting from s' and acting optimally along a infinite path; $\gamma \in [0,1]$ represents a discount factor that alleviates the contribution of future rewards in favor of present rewards. The best policy π^* returns for each state s the action that allows the cumulated reward to be maximized.

2.2. Probabilistic Computation Tree Logic

PCTL [8] is an extension of the temporal logic CTL [12] which allows for probabilistic quantification of properties interpreted over an MDP. In particular, a PCTL formula is satisfied in a state s if it is satisfied under all existing policies. The syntax supports the definition of state formulas ϕ and path formulas ψ, which are evaluated over states and paths, respectively. Formally, a state formula is defined as follows.

$$\phi ::= \mathit{true} \mid a \mid \phi \wedge \phi \mid \neg\phi \mid \mathcal{P}\bowtie p[\psi] \tag{2}$$

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. A path formula is defined as follows.

$$\psi ::= \mathtt{X}\phi \mid \phi \,\mathtt{U}\, \phi \mid \phi \,\mathtt{U}^{\leq k}\, \phi \tag{3}$$

where $\mathtt{X}$ represents the *next* operator, $\mathtt{U}$ is the *until* operator, and $\mathtt{U}^{\leq k}$ with $k \in \mathbb{N}_{\geq 0}$ is the *bounded until* operator. The temporal operators $\mathtt{G}$ (i.e., *globally*) and $\mathtt{F}$ (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 $\bowtie p$. Here we focus on PCTL extended reward-based properties which introduces the *reward operator* $\mathcal{R}\bowtie r[\xi]$, where:

$$\xi := I^{=k} \mid C^{\leq k} \mid \mathtt{F}\phi \tag{4}$$

Intuitively, $\mathcal{R}\bowtie p[I^{=k}]$ holds in s if the expected reward, after exactly k steps along the paths originating in s, meets the bound $\bowtie\rho$, with $\rho \in \mathbb{R}_{\geq}0$. $\mathcal{R}\bowtie\rho[C^{\leq k}]$ holds in s if, from state s, the expected reward cumulated after k steps meets the bound $\bowtie\rho$. $\mathcal{R}\bowtie\rho[\mathtt{F}\phi]$ holds in s if, from state s, the expected reward cumulated before reaching a state satisfying ϕ meets the bound $\bowtie\rho$.

In the following sections we will use $\mathcal{R}_r$ to express reward-based properties that refer to the reward structure r. Where the context is clear, we will drop the subscript r.

We let the reader refer to the comprehensive description and theoretical treatment by Baier et al. [12] of PCTL extended with reward-based properties.

2.3. Bayesian Inference

The Bayesian approach [6] represents a very popular framework for inference and prediction because of its easiness to be applied in practice. Bayesian and probabilistic techniques really come into their own in domains where uncertainty must be taken into account.

The main goal of Bayesian inference is to learn about one or more uncertain/unknown parameters θ that describe a stochastic phenomenon of interest. To incrementally update our *prior* knowledge (*hypothesis*) about θ, we observe the phenomenon of interest to collect a sample of data $y =$

Copyright © 2010 John Wiley & Sons, Ltd.
Prepared using stvrauth.cls

DOI: 10.1002/stvr

$(y_1, y_2, ..., y_n)$ and calculate the conditional density $f(y|\theta)$ of the data given θ (usually referred to as the likelihood function). The Bayesian approach also takes into account the hypothesis about θ. This information is often available from external sources such as expert information based on past experience or previous studies. This information is represented by the *prior* distribution $f(\theta)$. By combining the prior and the likelihood using the Bayes' theorem we obtain the *posterior* distribution $f(\theta|y)$, describing the best knowledge of the true value of θ, given the observed data y.

$$f(\theta|y) \propto f(\theta) \cdot f(y|\theta) \tag{5}$$

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

$$E[\theta|y] = \int \theta \cdot f(\theta|y) d\theta \tag{6}$$

if the true value of θ is known in advance it is possible to measure the Relative Error (RE) which represents the discrepancy between an exact value and some approximation of it. The RE can be easily computed by dividing the absolute error by the magnitude of the exact value.
Interval estimation is typically addressed by calculating the shortest possible region of probability 0.95, that is the *Highest Posterior Density* (HPD) region [13] defined as the following set of θ values.

$$\mathcal{C} = \{\theta : f(\theta|y) \geq 0.95\}. \tag{7}$$

The width of the HPD region yields the highest possible accuracy in the estimation of the true value of θ and is usually adopted as a measure of the confidence gained after the inference activity.

2.4. *On-the-fly Model-based Testing of Probabilistic Systems*

Model-based testing is a software testing technique where run-time behavior of a SUT is checked against predictions made by a model description of the system's behavior. An on-the-fly (or online) MBT approach combines test derivation from a model and test execution/evaluation into a single algorithm that generates test cases at run time stochastically sampling the state space rather than exhaustively attempting to enumerate it. Testing probabilistic systems, such as those modeled as MDPs, is usually composed of two major activities: *functional testing* to assess the functional correctness of the SUT; and *statistical hypothesis testing* [14] that focuses on determining whether probabilities conform to the mathematical model. The functional evaluation procedure adopted by METRIC is comparable to the one introduced by Veanes et al. [15]. Informally, all outputs produced by the SUT must be predictable by the model. This condition is checked by executing a probabilistic input/output *conformance game* [15] between two players: the *controller* (or decision maker) that choses the input from those available in the model; and the *observer* that verifies if the current execution trace is feasible in the model. The conformance game can be formalized by leveraging the notion of *alternating simulation* and *refinement*. We let the reader refer to the work by de Alfaro et at. [16] for a comprehensive theoretical discussion of this functional evaluation approach. Beside the conformance game, statistical hypothesis testing (i.e., the focus of this work) assesses whether the frequencies observed during the test process correspond to the probabilities specified in the model. This represents a fundamental activity that usually makes use of statistical methods and frequency analysis on the gathered sample to give a verdict based on a chosen level of confidence or to modify the model in accordance with the observations. METRIC makes use of Bayesian inference while gathering evidence from test executions, thus to compute the Posterior density function associated with specific uncertain parameters θ of the MDP model.

2.5. *Problem statement*

Uncertainty is a natural and inevitable part of pattern classification in real-world domains. Uncertainty can be defined as any departure from the unachievable ideal of complete deterministic knowledge of the system. Preliminary research activities towards uncertainty classification in

Copyright © 2010 John Wiley & Sons, Ltd.
Prepared using stvrauth.cls

DOI: 10.1002/stvr

software systems aim at establishing a common vocabulary and taxonomy of uncertainty from the perspective of a software system (see work by Ramirez et al. [17], Esfahani et al. [1], Perez et at. [18] to name a few). Sources of uncertainty can occur either at requirements, design, or execution phases [1, 19]. At each of these phases, uncertainty can be introduced into the system by the system itself (i.e., *system uncertainty*) or its execution environment (i.e., *environmental uncertainty*). Examples of sources of uncertainty include: input parameter uncertainty (due to uncertain input values given to the mathematical model), structural uncertainty (due to approximations in the mathematical model), algorithmic uncertainty (coming from numerical approximations per implementation of the computer model), and experimental uncertainty (due to the inherent variability of experimental measurements). Uncertainty quantification intends to work toward reducing these sources of uncertainty (both in their epistemic and aleatory natures [1]). Traditionally, there exist two complementary approaches to uncertainty quantification: *Forward Uncertainty Propagation* (FUP) [20] and *Inverse Uncertainty Quantification* (IUQ) [4]. The first one focuses on studying at design-time the quantification of the uncertainty in system output(s) propagated from uncertain inputs. The latter one is essentially the *inverse* problem. Given some experimental measurements of a system and some simulation outputs from its mathematical model, IUQ evaluates the discrepancy between the measured data at runtime and the mathematical model (i.e., *bias correction*) and estimates the values of uncertain parameters in the model (i.e., *parameter calibration*). More precisely, IUQ aims at estimating the discrepancy between measured data y^e at runtime and the response $y^{\mathcal{M}}(\theta)$ of the system model $\mathcal{M}$ that depends on different uncertain parameters θ. Starting from the initial approximation $y^{\mathcal{M}}(\theta) \simeq y^e$, and performing a sequence of observations is it possible to apply statistical machinery to infer a probability distribution for θ^* describing the best knowledge of the true parameter values, such that $y^{\mathcal{M}}(\theta^*) = y^e$.

The IUQ problem is recently drawing increasing attention, since uncertainty quantification of a model and its inference from the true system response(s) are crucial for engineering reliable systems. As an example, consider the category of service-based systems [21] characterized by a workflow of interactions with distributed components (e.g., web-services or microservices) owned by multiple third-party providers with different Quality of Service (QoS) attributes θ, such as reliability, performance, and cost. In this context, both functional and non-functional aspects of the final system depend on the ability of the external services to comply with the design-time assumptions during the early design-time specification phases. Here predictability is very hard to achieve. In fact, modern service-based systems are often situated in complex ecosystems that can be hard to fully understand and specify at design-time. Thus, our IUQ approach can be used to complement traditional verification activities (e.g., model checking) in order to update beliefs on θ parameters so that formal verification can then assess with increased confidence whether the specification, including updated knowledge, satisfies design-time requirements.

3. METRIC METHODOLOGY OVERVIEW

As anticipated in Sect. 1, the key idea of METRIC is to apply hypothesis testing to estimate the discrepancy between runtime evidence and the probabilistic representation that depends on different uncertain parameters. Although the METRIC methodology applies in principle to any probabilistic representation of non-functional QoS attributes, here we focus on systems modeled as MDPs that are a widely accepted formalism to model software system reliability [11]. MDPs are often used for design-time reliability assessment of systems composed of interacting parts, such as component-based software systems or service oriented architectures [22]. In addition to that, they provide the ability to specify both non-deterministic (i.e., external inputs) and probabilistic (i.e., uncertain responses) behavior. As detailed in the next sections, this clear separation represents a key aspect that will allow us to formalize in a natural way testing related concerns, such as input/output conformance and (on-the-fly) test case generation. Hereafter, we give a preview of the approach we devise and we describe how selected methods and techniques are packaged together inside the METRIC methodology to deal with the IUQ problem. A high-level overview of the proposed methodology is shown in Figure 1. It relies on the iteration of three different phases:

Copyright © 2010 John Wiley & Sons, Ltd.
Prepared using stvrauth.cls

DOI: 10.1002/stvr

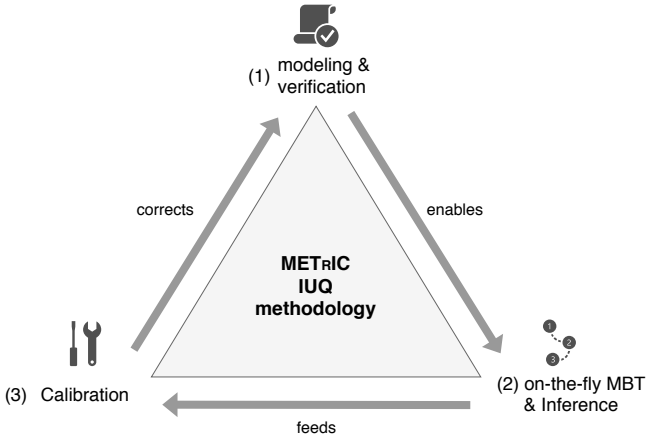


Figure 1. The METRIC methodology

(1) design-time *modeling & verification* phase;

(2) *on-the-fly MBT & inference* phase; and

(3) *calibration* phase.

Essentially, during the (1) modeling & verification phase, the MDP model of the system is created and formally verified against requirements. However, models developed at design time are often subject to different sources of uncertainty. For example, the failure probability or the response time of an operation (represented by a transition in the model) may be hard to predict and assumptions (i.e., prior knowledge) may be inaccurate. To account for uncertainty in model parameters, we use Prior probability distributions (or simply Priors) that express beliefs about uncertain quantities before some evidence is taken into account [23]. The (2) on-the-fly MBT & inference phase takes as input the MDP model of the system and the set of Priors. Priors represent model parameters whose value becomes known after the model-based hypothesis testing process. As described by Gerhold et al. [14], hypothesis testing (differently from functional testing) represents a fundamental activity to assess whether the frequencies observed during testing processes correspond to the probabilities specified in the model. We assume that the system implementation (i.e., the SUT) is available. This could be done, for instance, by deriving the implementation from the model using a model-driven software engineering process or the implementation may be already deployed long before its model. METRIC makes no assumption on how the implementation has been developed and deployed. The objective of MBT here, is to guide the execution of the system and collect a sample via multiple test runs. Intuitively, the larger the sample of the uncertain parameters, the higher the effectiveness of the hypothesis testing process. Thus, the MBT leverages uncertainty awareness, in the sense of θ parameters location, to perform a controlled model-based exploration and increase the sample size. Namely, test case generation is grounded on the computation of *infinite horizon* optimal policies of the MDP model that maximize the probability to execute those paths affected by θ. Evidence is gathered by monitoring the system in its operational environment during MBT. In fact, testing feeds a Bayesian inference process that incrementally updates the Priors depending on the gathered sample.

Let us consider an example where a transition in the model leads to a failure state and its probability value represents an uncertain failure rate of a software component. The (on-the-fly) test case generation provides the ability to select those inputs that will guide testing towards the (multiple) execution of the faulty component. The Bayesian inference process updates the prior knowledge based on the actual value of the component's failure rate observed during the MBT activity. Bayesian inference [13] represents an effective technique used to update belief as more evidence (i.e., experimental data) becomes available.

Copyright © 2010 John Wiley & Sons, Ltd.

Prepared using stvrauth.cls
DOI: 10.1002/stvr

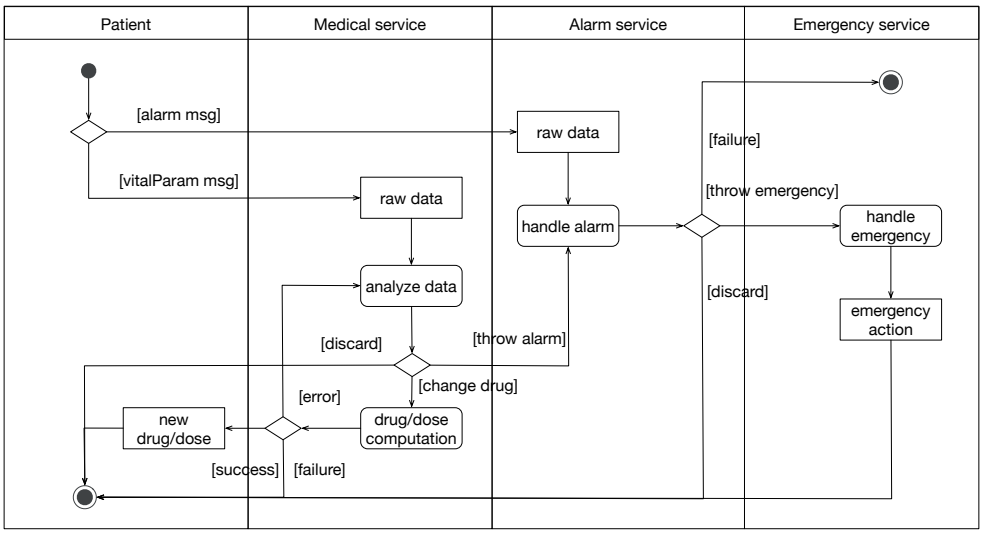


Figure 2. Activity diagram showing the TAS workflow.

Bayesian inference computes the Posterior probability distributions (or simply Posteriors) that represent the probability distributions of the uncertain parameters conditional on the evidence obtained from testing. The Posteriors compose the input of the (3) calibration phase, which is in charge of updating the θ parameters of the MDP model by summarizing the Posterior probability distributions as described by Robert et al. [23].

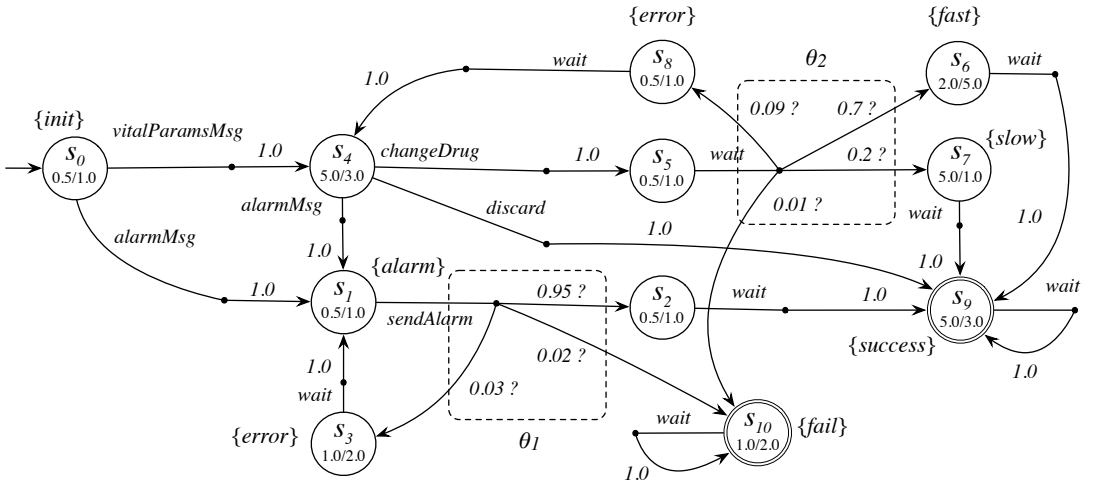
In short, the METRIC methodology to IUQ iterates over three different phases which use methods and techniques able to reduce the overall system uncertainty and deliver increased confidence on the software product. Namely, (on-the-fly) test case generation is used to stress the uncertain software components and observe the actual behavior of the system implementation. A statistical hypothesis testing process based on Bayesian inference is then used to update beliefs and calibrate the uncertain parameters in the MDP model. New values of model parameters, estimated out from a METRIC iteration, represent the basis of new verification phases and the prior knowledge of future evolutions of the software system.

4. A RUNNING EXAMPLE: THE TELE ASSISTANCE SYSTEM

The running example we use in the article is based on a case study introduced by Weyns et al. [24] which deals with a distributed system for medical assistance. This application, called Tele Assistance System (TAS), consists in a service-based system providing health support to chronic condition patients at their homes. It is composed of a number of sensors embedded in one or more wearable devices to track patients' vital parameters and a number of remote services provided by healthcare, pharmacy and emergency units. Figure 2 shows a high-level overview of the workflow which coordinates services typically managed by external organizations, other than the owner of the service composition. This implies that both functional and non-functional properties of the composed service rely on behaviors of third-party partners that influence the obtained results. The workflow starts as soon as a Patient enables the wearable device which periodically sends a message to target medical service. The medical service replies to the following messages: *vitalParam* message and *alarm* message. The first message contains the patient's vital parameters that are forwarded to the medical service by executing the activity operation *analyze data*. The medical service is in charge of analyzing the data and it replies by either discarding the data (i.e., no action is required), changing the drug/dose, or sending an alarm. The latter message may trigger the intervention of a first-aid squad composed of doctors, nurses, and paramedics, whose task is to visit the patient at home in case of emergency. To alert the squad, the TAS workflow invokes the

Copyright © 2010 John Wiley & Sons, Ltd.
Prepared using stvrauth.cls

DOI: 10.1002/stvr



Mapping between workflow activities and model states:

analyze data	S_4
drug/dose computation	S_5, S_6, S_7, S_8
handle alarm	S_1, S_3
handle emergency	S_2

Figure 3. Markov Decision Process of the TAS behavior.

handle emergency activity of the emergency service. The TAS is an example of a wide category of service-based systems (e.g., e-commerce, e-health, online banking, taxi-hailing, etc.) characterized by a workflow of interactions with distributed components (e.g., web-services or microservices) owned by multiple third-party providers with different Quality of Service (QoS) attributes, such as reliability, performance, and cost as described by Camilli et al. [21]. As anticipated in Sect. 2, the overall functional and non-functional quality attributes of the final system also depend on the capability of the external services to comply with the assumptions made at design-time during the specification of the system.

Figure 3 shows a formalization of the TAS behavior by means of a MDP. Labels associated with states (e.g., $\{alarm\}$, $\{error\}$) represent the labeling function L. The model is augmented with two reward structures $r_{\text{time}}/r_{\text{energy}}$ associated with states (e.g., S_0 maps to $0.5/1.0$). These values represent response time and power consumption of components, respectively.

As specified by the model, the TAS takes periodical measurements of the vital parameters of a patient and employs a third-party medical service for their analysis. This is represented by the occurrence of the action *vitalParamsMsg* from the initial state S_0 and the sojourn in state S_4. The analysis made by the medical service may trigger either the computation of new medication/dose to the patient (i.e., *changeDrug* action), or the invocation of the alarm service eventually leading to dispatch a first aid team to the patient home (i.e., *alarmMsg* action). The legend provided in Figure 3 lists a mapping between workflow activities and model states. Some activities are modeled by a single state in the MDP (e.g., the activity *handle emergency* maps to state S_2). Some other activities are decomposed into different states to represent further refined information about the functional status of services (e.g., the activity *handle alarm* maps to states S_1, S_3, representing a working state and a recoverable error state, respectively).

The TAS specification contains a number of uncertain transition probabilities, identifying the set of uncertain parameters θ, that are explicitly represented in Figure 3 by numeric values followed by the question mark "?" symbol. For example, the alarm service can be invoked directly by the patient, by using a special button placed on a wearable device. This behavior of the workflow is represented by the *alarmMsg* action directly occurring from state S_0. Once an alarm has been sent (i.e., *sendAlarm* action), the alarm service may successfully complete the execution (most likely with probability 0.95), or exhibit a faulty behavior such as data loss on the communication channel

Copyright © 2010 John Wiley & Sons, Ltd.
Prepared using stvrauth.cls

DOI: 10.1002/stvr

or an unexpected exception (with probability 0.03 as initial assumption), or exhibit a failing behavior such as unreachability of the alarm service (calculated as one minus the sum of the previous two probabilities, i.e., 0.02 in this case). As another example, the pharmacy service, represented by the sojourn in state S_5, accepts as input a single action named *wait*. Intuitively, this means that waiting in state S_5 can ends up in different target states with different probabilities. The operation can succeed slowly (probability 0.2 as initial assumption), succeed very fast (i.e., most likely with probability 0.7), fail with a recoverable error (with low probability, such as 0.09), or even fail in an unrecoverable manner (i.e., a very unlikely event, thus, with probability 0.01).

It is worth noting that, during the initial stages of the development process, these transition probabilities represent initial assumptions/beliefs on the QoS-related properties of the TAS. These assumptions are intrinsically subject to different types of uncertainty, usually found in service-based systems, such as *network data loss*, *service failure*, *service response time*, *inadequate design*, etc. (see the work by Ramirez at al. [19] for a comprehensive classification of uncertainty in this context).

Finally, let us assume that the TAS must satisfy the non-functional requirements reported in Table I. All these non-functional properties can be easily verified by means of off-the-shelf model checking software tools supporting PCTL as probabilistic temporal logics, such as PRISM [25]. However, the uncertainty, discussed early on, can jeopardize the ability to achieve these properties. A crucial point is that TAS must ensure *reliability under uncertainty*, meaning that reliability is a first class concern but it cannot be proven until the uncertainty has been mitigated by accounting evidence during testing/execution. This could make the usage of classic model checking (or more in general formal verification) ineffective or even leading to erroneous conclusions, if used in isolation only at design-time.

5. THEORETICAL ASPECTS OF METRIC

In this section we provide a formalization of the core techniques adopted inside METRIC. After giving some preliminary definitions about expressing the SUT behavior as a MDP, we provide a detailed description of the input/output conformance game, the model-based test case generation strategies, and the mathematical machinery involved during the model inference/calibration process.

5.1. Preliminary definitions

As introduced by Camilli et al. [9], the execution of a generic program P can be viewed in the same way as the one of its formal specification. In particular, the MDP model extracted from the SUT execution is called *model program* and denoted by $\mathcal{M}_{SUT}$. This model is defined exploiting the notion of *binding* declared by the tester using the DSL introduced in Sect. 3. In the following, we write $\vec{v}_{in}$ to identify a sequence of input parameters (i.e., the arguments of a subroutine), and we write $\vec{v}_{out}$ for the output parameters including the return value.

Definition 1 (Binding)
Given a MDP $\mathcal{M} = (S, S_0, A, \delta, L)$ and a set of subroutines of H of a program P, a *binding* is a tuple of partial functions $(\mathcal{H}, \mathcal{I}, \mathcal{P}re, \mathcal{P}ost)$ with domain $S \times A$, such that:

- $\mathcal{H}(a)$, with $a \in A(s)$, identifies a subroutine $h \in H$;
- $\mathcal{I}(a)$, with $a \in A(s)$, identifies a valid vector of input parameters $\vec{v}_{in}$ for the subroutine $\mathcal{H}(a)$;
- $\mathcal{P}re(s, a)$, with $a \in A(s)$, maps to a pre-condition that must hold for $\vec{v}_{in}$ given to $\mathcal{H}(a)$ from the source state s;
- $\mathcal{P}ost(s, a)$, with $a \in A(s)$, maps to a post-condition that must hold for $\vec{v}_{out}$ after the execution of $\mathcal{H}(a)$ in the target state s.

The notion of binding allows allows to view the behavior of the SUT associated to the MDP specification. In particular, both controllable and observable components are connected to the on-the-fly MBT module (see Figure 5) through the automatically generated test harness. Given a binding, the model program $\mathcal{M}_{SUT}$ is defined taking into account the specification $\mathcal{M}$ as follows.

Copyright © 2010 John Wiley & Sons, Ltd.
Prepared using stvrauth.cls

DOI: 10.1002/stvr

Definition 2 (Model Program)
Given a MDP specification $\mathcal{M} = (S, s_0, A, \delta, L)$ and a binding $(\mathcal{H}, \mathcal{I}, \mathcal{P}re, \mathcal{P}ost)$, the model program $\mathcal{M}_{SUT} = (S', s'_0, A', \delta', L')$ is a MDP model, such that:

- $S' \subseteq S$, $A' \supseteq A$;
- $s_0 = s'_0$, $L'(s) = L(s)$ if $s \in S$, $\emptyset$ otherwise;
- $\delta'(s, a)(s') > 0$ iff.
  - (i) the pre-condition $\mathcal{P}re(s, a)$ holds for $\vec{v}_{in} = \mathcal{I}(a)$
  - (ii) the post-condition $\mathcal{P}ost(s', a)$ holds for $\vec{v}_{out}$, resulting from the execution of $\mathcal{H}(a)(\vec{v}_{in})$

The intuition behind the notion of model program is as follows. The first condition ensures that, on one hand all observable configurations (states) of the implementation are possible in the model, and on the other hand that all possible actions in the model are possible in the implementation. The second condition imposes that the model and the implementation have the same initial state and labeling function. The latter condition describes possible transitions defined by δ' from a source state s and an action a to a target state s', provided that the model elements s, a, and s' have been bound to the implementation.

5.2. Conformance Checking

Given the preliminary definitions above, we formally define the *conformance relation* between the MDP models $\mathcal{M}$ (i.e., the specification) and $\mathcal{M}_{SUT}$ (i.e., the model program) using the notions of *alternating simulation* and *refinement* as described by Veanes et al. [15].

Definition 3 (Alternating simulation)
An alternating simulation between $\mathcal{M}$ and $\mathcal{M}_{SUT}$ is a binary relation $\sigma \subseteq S \times S'$, such that for all $(s, s') \in \sigma$,

(i) $A(s) \supseteq A'(s')$

(ii) $\forall a \in A(s), t \in S : \delta(s, a)(t) > 0, \exists a' \in A'(s'), t' \in S' : \delta'(s', a')(t') > 0$ s.t. $(t, t') \in \sigma$

Intuitively, the condition (i) ensures that the available actions in the model are possible in the implementation. The condition (ii) guarantees that if (i) holds for a given pair of source states then it also holds in the resulting target states from the application of any controllable action enabled in the model. Now, we can formally define when the output produced by the implementation are predictable by the model, using the notion of refinement as follows.

Definition 4 (Refinement)
A MDP $\mathcal{M}$ refines a MDP $\mathcal{M}_{SUT}$ iff. there exists an alternating simulation σ from $\mathcal{M}$ and $\mathcal{M}_{SUT}$ s.t. $(s_0, s'_0) \in \sigma$.

As anticipated in Sect. 2, the notion of refinement can be explained in terms of *conformance game* [15, 26] between two players: a *controller* and an *observer*. The game, visually represented in Figure 4, starts from the initial state s_0 of the model $\mathcal{M}$ and the initial configuration s'_0 of the implementation $\mathcal{M}_{SUT}$, and it consists of a sequence of steps. For each step, the controller makes it own move, i.e., it choses an available action in $A(s)$ from the current state s of the specification $\mathcal{M}$ and it executes the subroutine $\mathcal{H}(a)$ with a valid input vector of parameters $\mathcal{I}(a)$. There is a conformance failure if it is not possible to determine the available actions, or the subroutine, or a valid input. After the controller, the observer makes its own move, i.e., it evaluates the pre-condition $\mathcal{P}re(s, a)$ on the input vector, then if the precondition holds it determines the target state s', such that the post-condition $\mathcal{P}ost(s', a)$ evaluated on the output vector holds. Whenever a pre-condition does not hold or does not exist a target state s' such that the post-condition holds, there is a conformance failure. The game continues until the controller decides to end the game (i.e., a termination condition has been reached) or a conformance failure is found.

Copyright © 2010 John Wiley & Sons, Ltd.

Prepared using stvrauth.cls
DOI: 10.1002/stvr

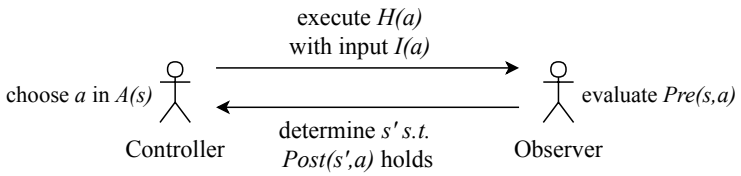


Figure 4. Conformance game.

5.3. *Uncertainty-aware Test Case Generation*

On-the-fly MBT is a technique able to derive tests from a program model and then execute them by means of a single algorithm. The idea is to stochastically sample a large state space at runtime rather than pre-computing a huge amount test cases derived from all possible responses from the SUT. Our algorithm dynamically generates test cases by executing the conformance game and providing to the user control over test scenarios by selecting actions during the test run based on specific test case generation strategies and termination conditions. The uncertainty-aware strategy is governed by a probabilistic function that has the following form:

$$\mathcal{P}(s,a) = \begin{cases} 0 & \omega(s,a) = 0 \\ \omega(s,a) / \sum_{a' \in A(s)} \omega(s,a') & \text{otherwise} \end{cases} \tag{8}$$

The function ω is a per-state weight function that maps a state s and an action a to a value in $\mathbb{N}_{\geq 0}$. The weight function allows the generation of test cases to be configured. Our approach takes explicitly into account the uncertainty modeled by means of the Prior distributions associated with θ parameters in order to stress the uncertain components of the SUT during testing. To achieve this goal, we first construct a set of uncertainty-aware reward structures defined as follows.

Definition 5 (Uncertainty-aware Reward Structure)
Given a MDP $\mathcal{M}$ and a set of uncertain parameters $\theta_i \subseteq \theta$, the *uncertainty*-aware reward structure is a reward structure $u = (u_s, u_a)$, s.t.,

- $u_s(s) = 0, \forall s \in S$
- $u_a(s,a,s') = \begin{cases} k \in \mathbb{N}_{>0} & \delta(s,a)(s') \in \theta_i \\ 0 & \text{otherwise} \end{cases}$

The rationale of the structure uncertainty-aware structure u is to assign a high reward value (k) to model transitions associated with uncertain parameters, and a low reward value (0) to other transitions. Given such a reward structure, we construct a suitable decision maker able to choose actions maximizing the (infinite horizon) cumulative reward over the uncertainty-aware structure u. Namely, it leverages the optimal policy π^*_u, that maximizes the expected sum of u rewards. The best policy $\pi^*_u(s)$ returns for each state s the action that allows the model-based exploration to run optimally towards the uncertain parameters.

The subsets θ_i are motivated by the practical need of constructing regions of θ parameters attached to transitions sharing the same source state and action. Namely, the subsets θ_i represent partitions of θ defined as follows:

$$\theta_i = \{\alpha \in \theta \textit{ s.t. } \exists a \in A(s_i), s_j \in S : \delta(s_i,a)(s_j) = \alpha\}, \textit{s.t.} \bigcup_i \theta_i = \theta \tag{9}$$

Intuitively, we partition θ in order to identify different uncertain model regions and then we compute the set of optimal policies that maximize the probability to reach each different uncertain region of the model.

Considering our running example, the set θ is partitioned in two subsets θ_1 and θ_2. As shown in Figure 3, θ_1 contains the parameters attached to transitions starting from s_1 when choosing the

Copyright © 2010 John Wiley & Sons, Ltd.
Prepared using stvrauth.cls

DOI: 10.1002/stvr

action *sendAlarm*; θ_2 contains the parameters attached to transitions starting from s_5 when choosing the action *wait*.

The set of optimal policies $\{\pi^*_{u_i}\}$, is used in turn to construct the uncertainty-aware weight function ω as follows.

$$\omega(s,a) = \begin{cases} k \in \mathbb{N}_{>0} & \exists i : \sigma^*_{u_i}(s) = a \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

The weight function ω makes the controller able to stochastically sample the available actions maximizing the probability to reach the uncertain model regions.

The optimal policies computed on the TAS example, considering the uncertainty-aware reward structures given by θ_1 and θ_2, are reported in Table II. In this example, this computation is trivial, however in general it can be arbitrarily complex due to complex combination of transition probabilities, alternative paths and loops in a MDP model.

5.4. Inference & Calibration

During testing, we set up a Bayesian reasoner, where the prior knowledge $f(\theta)$ is incrementally updated taking into account the evidence y. In the following we introduce a brief overview on the statistical machinery used to perform this calibration and we refer the reader to the comprehensive description introduced by Insua et al. [13] for more details.

A natural conjugate Prior for the uncertain transition probabilities of a MDP model is defined by letting $p^a_i = (p^a_{i,j}, ..., p^a_{i,k})$ have a *Dirichlet* distribution [27], where $p^a_{i,j}$ is the probability to observe a transition from s_i to s_j when the action a is chosen.

$$p^a_i \sim Dir(\alpha_i), \text{ where } \alpha_i = (\alpha_{i,j}, ..., \alpha_{i,k}) \tag{11}$$

During the conformance game, the observer component collects $n^a_{i,j}$ that represents how many times the transition from s_i to s_j has been observed, when the action a is selected. Given a sequence of observations (i.e., a *sample*), the Posterior distribution is also a Dirichlet distribution and can be computed very efficiently as follows.

$$p^a_i | y \sim Dir(\alpha'_i), \text{ where } \alpha'_i = (\alpha_{i,j} + n^a_{i,j}, ..., \alpha_{i,k} + n^a_{i,k}) \tag{12}$$

When little prior information, a natural possibility is to use a uninformative Dirichlet Prior with $\alpha^a_{i,j} = 1/2, \forall i, j, a$. Otherwise, when past experience is available, is it possible to use a Dirichlet Prior with $\alpha_{i,j} = n^a_{i,j}$.

For instance, considering the TAS we may describe the hypothesis on θ_1 with a Dirichlet Prior with $\alpha_1 = (\alpha_{1,2} = 900, \alpha_{1,3} = 90, \alpha_{1,10} = 10)$, if in our past experience, we observed 900 transitions from s_1 to s_2, 90 transitions from s_1 to s_3 and 10 transitions from s_1 to s_{10}, in a sample of $1K$ observations.

At termination, our on-the-fly MBT algorithm performs the calibration of the θ parameters by performing point and interval estimation of the Posterior distributions. For instance, let us consider once again the TAS example. Assume that starting from the Prior example given above, and by running the on-the-fly MBT, we eventually come up with a Posterior distribution with updated $\alpha'_1 = (\alpha'_{1,2} = 88000, \alpha'_{1,3} = 11000, \alpha'_{1,10} = 1000)$. This Posterior leads to update the parameters attached to transitions $\langle s_1, s_2 \rangle$ and $\langle s_1, s_3 \rangle$ (when *sendAlarm* is chosen) to 0.88 and 0.12, respectively. The calibration process in this case can be carried out with high confidence because of a very small HPD region (less than 0.05 for each parameter). The calibrated model, along with the new estimations, can lead to invalidate the design-time requirements of the TAS (Table I). For instance, by using the PRISM model checker we can easily verify that R_2 and R_3 do not hold after the calibration activity mentioned above.

5.5. Termination conditions

The tester has the ability of configure the on-the-fly MBT algorithm of METRIC with different termination conditions which determine when the controller can decide to end the conformance

Copyright © 2010 John Wiley & Sons, Ltd.

Prepared using stvrauth.cls
DOI: 10.1002/stvr

game. Two first conditions are classical criteria based on the desired level of coverage of the available state-action pairs and the desired length of test runs in terms of number of executed methods. Both these termination conditions are typically used in combination with random and history-based sampling policies aiming at testing the SUT by generating a pseudorandom permutation of actions until either a conformance failure is found, or the desired level of confidence has been reached. In addition to that, METRIC introduces a novel termination condition based on the precision achieved during the inference process. This condition is grounded on a model selection criterion based on the convergence of the *Bayes factor* [13]. The Bayes factor is used to choose between two probabilistic models with different parameters θ and θ', on the basis of observed data y as follows.

$$\mathcal{F} = \frac{f(y|\theta)}{f(y|\theta')} \tag{13}$$

The two terms of the ratio $\mathcal{F}$ represent the likelihood that data y are produced under different assumptions θ and θ'. A positive value of $\mathcal{F}$ means that the data under consideration supports more the assumption θ than θ'. The usual interpretation of this value considers $\mathcal{F} \in [10^0, 10^{1/2}]$ as not substantial. So, our termination condition reduces to check whether the Bayes factor K falls in this interval. More precisely, given a sample size N, the on-the-fly MBT algorithm computes the Bayes factor for each Posterior, every N observations. Thus, the convergence of the computed $\mathcal{F}$ values is used as a termination condition of the algorithm. The rationale behind this choice is to exploit the Bayes factor to recognize when the observed data does not make the posterior knowledge change more than a significant threshold.

6. METRIC TOOLCHAIN

The METRIC methodology has been implemented by a toolchain whose design follows the high-level architecture shown in Figure 5. This schema shows the METRIC macro-components and interactions. There are two main subsystems: the *front end* (taking into account design-time concerns), and the *back end* (in charge of applying testing-related activities). The major components of our toolchain has been developed using the JAVA programming language, ASPECTJ [28], and the grammarware framework XTEXT/XTEND [29]. However, the approach, as described in Sect. 5, is general and does not refer to any specific feature of our programming language and/or libraries of choice. Therefore, it can be substantially applied to other languages with limited technological modifications. The major components of our toolchain have been released as open source software[§] to encourage the adoption of the proposed approach and to allow the replication of experiments.

In the remaining of the section we provide further details on both the front end and the back end.

6.1. The Front end: Modeling & verification

The core components of the front end are the *Modeler* (used to create/edit the MDP model and the uncertain parameters), the *Generator* (in charge of generating the test harness), and the *PRISM model checker* (used to perform design-time verification). Numbered labels refer to the major components detailed in the following.

6.1.1. Modeler. This component (1) allows the tester to create and/or edit the SUT behavior (including the uncertain parameters) in terms of MDP model. In addition, the tester specifies the Prior distributions, and the *binding*, describing how the SUT components map to the provided model. All these concepts are defined using a textual Domain Specific Language (DSL). As an example, Listing 1 reports an extract of the TAS specification using the DSL. The language allows the MDP to be defined intuitively by using the keywords: actions (line 2), states (line 4), and arcs

[§]Publicly available at https://github.com/SELab-unimi/mdp-generator and https://github.com/SELab-unimi/mdp-simulator-monitored, together with all artifacts produced for the TAS running example.

Copyright © 2010 John Wiley & Sons, Ltd.
Prepared using stvrauth.cls

DOI: 10.1002/stvr

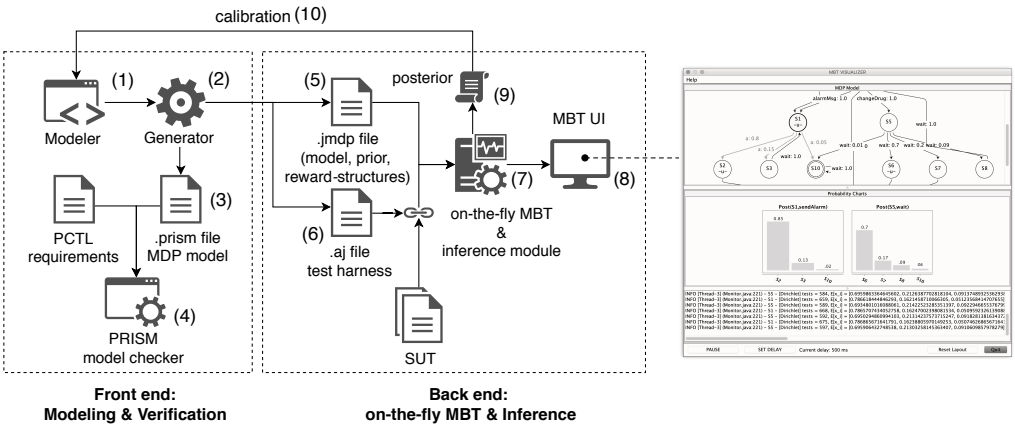


Figure 5. METRIC high-level toolchain.

(line 16). States can be augmented with Prior density functions using the keyword Dir (which stands for *Dirichlet*) describing the hypothesis on the uncertain parameters.

Listing 1 contains also the definition of the *binding* between the specification of the system's behavior and the implementation. The binding is defined by using the keywords observe (line 24) and control (line 32). Essentially, we follow the notation introduced in Sect. 5 to distinguish between *controllable* behavior from the tester (i.e., the environment, such as user requests) and *observable* behavior from the running software system. In particular, the observe section contains a mapping between transitions of the model and methods (or more generally subroutines) of the SUT along with pre- and post-conditions (i.e., arbitrary conditions on input parameters and return values). The control keyword is used to define a mapping between states of the model and states of *quiescence* [30] of the SUT, i.e., states where the SUT expects inputs from the external environment using the available SUT APIs. Inputs, selected during test case generation, depend on the available actions from model states and they are declared in the actions section by mapping actions to arbitrary JAVA objects. For instance, line 3 declares that the Strings "v" and "a" are associated to the model actions *vitalParamsMsg* and *alarmMsg*, respectively.

The Modeler is currently implemented as an ECLIPSE IDE plug-in and it supports a comprehensive set of standard editing features: syntax highlighting, error checking, auto-completion, and parse tree exploration.

6.1.2. Generator. This component (2) is in charge of translating the textual MDP model to a `.prism` file (3), i.e., the input format accepted by the PRISM model checker (4). Then, the Generator creates a number of software artifacts used as input by the testing module. These artifacts include: (5) a concise textual representation of the MDP model (a `.jmdp` file) equipped with the priors and the reward structures needed by the MBT module to carry out Bayesian inference; and (6) the ASPECTJ instrumentation script (i.e., the test harness as `.aj` file) that allows the SUT to be linked to the MBT structure depending on the binding.

6.1.3. Model checker. The PRISM model checker is part of the front end and it enables formal verification of design-time requirements of the system under development. As anticipated in Sect. 4, requirements are formalized using PCTL (see Table I) and verified against the MDP model of the system behavior. Before accounting evidence during testing, design-time model checking serves as a means to verify the desired requirements against the system's model that includes a number of assumptions/belief. These uncertain parameters become quantified only after executing the hypothesis testing process.

Copyright © 2010 John Wiley & Sons, Ltd.
Prepared using stvrauth.cls

DOI: 10.1002/stvr

```
model "TAS"
actions
  vitalParamsMsg {"v"} alarmMsg {"a"} finish {"f"} sendAlarm {"s"} ...
states
  S0 {} initial
  S1 {alarm} Dir(sendAlarm, <S2, 190.0> <S10, 4.0> <S3, 6.0>)
  S2 {}
  S3 {error}
  S4 {}
  S5 {} Dir(wait, <S6, 140.0> <S7, 40.0> <S8, 18.0> <S10, 2.0>)
  S6 {}
  S7 {slow}
  S8 {fast}
  S9 {success}
  S10 {fail}
arcs
  a0 : (S0, vitalParamsMsg) -> S4, 1.0
  a1 : (S0, alarmMsg) -> S1, 1.0
  a2 : (S0, finish) -> S9, 1.0
  a3 : (S1, sendAlarm) -> S2, 0.95
  a4 : (S1, sendAlarm) -> S10, 0.02
  a5 : (S1, sendAlarm) -> S3, 0.03
  ...
observe
  a0 -> "public IntegerState MDPSimulator.doAction(..)", args {state:"IntegerState" action:"char"},
        precondition "state.label().equals(\"S0\") && action=='v'",
        postcondition "result.label().equals(\"S4\")"
  a1 -> "public IntegerState MDPSimulator.doAction(..)", args {state:"IntegerState" action:"char"},
        precondition "state.label().equals(\"S0\") && action=='a'",
        postcondition "result.label().equals(\"S1\")"
  ...
control
  S0 -> "private char MDPDriver.waitForAction(..)",
        args {actionList: "Actions<CharAction>" input:"InputStream"}
  S2 -> "private char MDPDriver.waitForAction(..)",
        args {actionList:"Actions<CharAction>" input:"InputStream"}
  S5 -> "private char MDPDriver.waitForAction(..)",
        args {actionList:"Actions<CharAction>" input:"InputStream"}
sampleSize 2000
explorationPolicy uncertainty
```

Listing 1. Extract of the TAS specification using our DSL.

6.2. *The Back end: On-the-fly MBT & Inference*

The core macro-component of the back end is (7) the *on-the-fly MBT & inference* module. In addition to that, the METRIC framework also provides a (8) User Interface (UI) that the tester may use to visualize information about the testing/inference activity conducted by the core module.

6.2.1. MBT Module. The (6) customized test harness allows controllable/observable components of the SUT to be handled at runtime by the MBT module. More precisely, this instrumentation provides a particular high-level view of the SUT behavior matching the abstraction level of the MDP specification. Moreover, it provides a serialized view of the execution of observable methods in order to gather useful data for the inference activity. Controllable methods are handled by supplying external inputs via the available APIs; technically, this is realized by injecting suitable input arguments (specified through the DSL) upon the execution of the controllable methods during testing. The MBT module dynamically generates test cases from the MDP specification depending on the selected test case generation *strategy*. During testing, the MBT module collects data and performs a Bayesian inference process to compute the Posterior density functions associated with the uncertain parameters θ. If a conformance failure is found, the overall testing process ends and a detailed report about the failure is reported to the tester. Whenever testing reaches termination, a summarization (9) of the Posterior distributions is produced. This last information is used in turn

Copyright © 2010 John Wiley & Sons, Ltd.

Prepared using stvrauth.cls
DOI: 10.1002/stvr

to perform point/interval estimation in order to (10) calibrate the initial MDP model and update the knowledge on the uncertain parameters.

The overall testing and learning process terminates when the discrepancy between the model and the implementation became acceptable and all the initial requirements are satisfied.

6.2.2. MBT UI Module. The MBT UI allows information about both functional testing and Bayesian inference to be easily visualized. As shown in Figure 5, three different canvas in the main UI window show: the MBT model, the Posterior density functions, and a log produced by the MBT module. The MDP model (upper) canvas contains an animated visualization of the model. During testing, the UI highlights the current model state and the current action selected by the test case generation strategy. The Probability charts (middle) canvas displays the Posterior distributions so that the tester can see, how the inference process updates the knowledge on θ parameters while testing goes on. The log (lower) canvas shows textual information generated by the MBT module. Specifically, it shows for each uncertain parameter: the number of executed tests, the summarization of the Posterior density functions in terms of mean value and HPD region, and the Bayes factor, used as model selection criteria to determine the termination condition.

7. EMPIRICAL EVALUATION

In this section we introduce the objective of our empirical evaluation by means of a number of research questions (Sect. 7.1); we describe the overall design of the evaluation (Sect. 7.2); we present the collected results (Sect. 7.3); we further discuss them (Sect. 7.4); and then we describe how threats to validity have been mitigated (Sect. 7.5).

7.1. Research questions

The major objective of this section is to show the evaluation activity to assess the capability of METRIC to statistical hypothesis testing of uncertain software systems. In particular, we aim at answering the following research questions:

RQ1: How much is the accuracy of the uncertainty quantification process applied by our uncertainty-aware testing method?

RQ2: How much is the effort required by our uncertainty-aware testing method?

RQ3: How does the uncertainty-aware testing method compare to traditional pseudo-random model-based exploration strategies?

7.2. Design of the evaluation

The METRIC approach to IUQ has been evaluated through a large testing campaign using using the TAS example to construct a number of testing scenarios. Experiments have been carried out on a machine with following setting: Intel Xeon E5-2630 at 2.30GHz CPU, 32GB of RAM, Ubuntu 16.04.3 LTS (GNU/Linux 4.4.0-112-generic x86_64), and Oracle Java Runtime Environment 1.8. For each research question we selected a number of testing scenarios that have been designed taking control over a number of independent variables. This direct manipulation of these factors allowed specific experimental conditions to be controlled with decreased threats to internal validity, as further elaborated below. Table III summarizes these scenarios by listing independent variables (i.e., factors) and dependent variables (i.e., measured quantities) used in experiments. Beliefs on θ parameters have been controlled by setting upfront the Prior RE (i.e., amount of error) and the Prior HPD region width (i.e., degree of confidence), respectively. The sample size represents another factor affecting the existing relation between accuracy and effort of the inference process applied during testing. The accuracy of testing here represents the ability of mitigating the system uncertainties through the inference process. Thus, we measured it by means of the Posterior RE (i.e., amount of error in the inferred θ values) and the Posterior HPD region width (i.e., degree of

Copyright © 2010 John Wiley & Sons, Ltd.
Prepared using stvrauth.cls

DOI: 10.1002/stvr

confidence in the inferred values). The effort spent during testing has been measured in standard ways by using either the number of executed test cases or the number of executed methods until termination. Finally, both the exploration policy and the termination condition have been controlled to conduct a quantitative comparison of cost-effectiveness between our uncertainty-aware and existing (traditional) testing methods. To reduce the risk of obtaining results by chance, all the experiments have been repeated multiple times and using very large sample size values (200 to 4000). The comparative evaluation has been conducted by following the guidelines introduced by Arcuri et al. [31] to detected statistical differences through a pairwise comparison using the Mann-Whitney U test to calculate p-value with significance level $\alpha = 0.05$.

7.3. Results

Here we discuss the most significant results collected during experiments to address the research questions reported above. We refer the reader to our implementation for the replicability of the presented data. The software repository contains the METRIC MBT module implementation packages together the inputs used to obtain results described in this section.

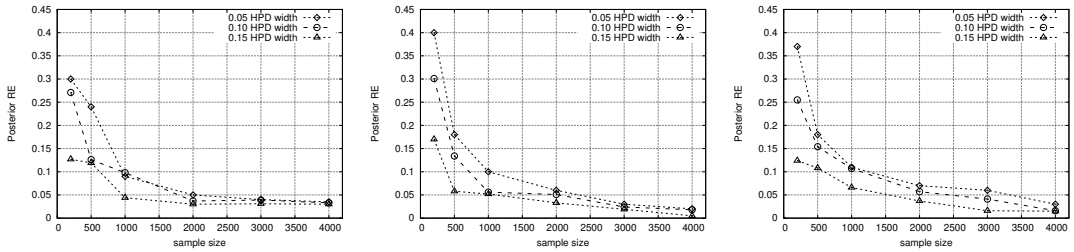
Results for RQ1. We addressed this research question by studying the existing relation between accuracy (out of testing), initial beliefs (before testing), and sample size. Figure 6 shows the Posterior RE out of multiple testing activities (with the uncertainty-aware policy) having diverse hypothesis (Prior RE and HPD region width). Namely, we selected different error and confidence scales as follows. Selected RE scales are: *small error* (0.4), *substantial error* (0.8), and *large error* (1.6). Selected HPD region widths scales are: *vague information* (0.15), *definite information* (0.10), *precise information* (0.05). Within each combination of these scales, we varied the sample size from *small sample* (200) to *large sample* (4000).

Figure 6 shows that there exists a direct relation between sample size and accuracy. Namely, the larger the sample, the higher the accuracy (i.e., the lower the Posterior RE). An increase in the data sample implies a more significant Bayes factor, thus allowing for better model selection supported by the data under consideration. For each selected scenario, testing leads to calibrate θ parameters with a very high accuracy (i.e., the order of magnitude of the Posterior RE is 10^{-2}) even in the worst-case scenario (i.e., *precise information* expressing *large error*). The shape of accuracy lines is very similar even changing the RE scale. We found that the Prior HPD width scale has a major impact on achieved accuracy especially when using *small sample*s. In case the tester does not have strong hypothesis on uncertain θ parameters, uninformative Priors represent always a better choice.

Results for RQ2. This research question has been addressed by measuring the number of tests required by the whole testing activity until termination using the same experimental setting described above. Figure 7 shows data extracted from experiments. Intuitively, there exists a direct relation between the sample size and the effort. Initial beliefs have a major impact on the effort as well. Namely, the higher the confidence in the hypothesis, the higher the effort in case of wrong Prior information. For instance, considering the scenario in Figure 7b (i.e., *substantial error*) and sample size $2k$, we observe about 65% decrease of the effort when passing from 0.10 to 0.05 HPD width. This behavior is more evident when increasing the RE. In fact, a very informative (but wrong) Prior has a negative impact on performance of the inference process. Considering the scenario in Figure 7c (i.e., *large error*) and sample size $2k$, we observe about 85% decrease of the effort when passing from 0.10 to 0.05 HPD width. To summarize, testing with wrong beliefs increases the required effort. The convergence of the Bayes factor is even slower when using Priors expressing precise (but wrong) information. In this case testing relies more on Priors rather than observed evidence. Even though worst cases (in the sense of completely wrong hypothesis) cause a slowdown of the whole testing process, the termination condition based on the Bayes factor has been showing high effectiveness in keeping inference up and running until the achievement of an adequate degree of accuracy.

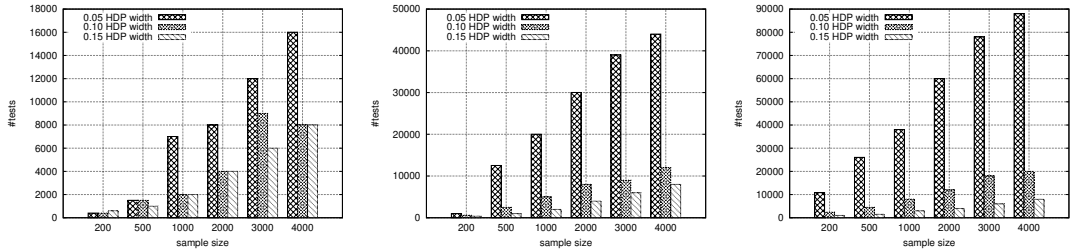
Copyright © 2010 John Wiley & Sons, Ltd.
Prepared using stvrauth.cls

DOI: 10.1002/stvr



(a) 0.4 Prior RE (b) 0.8 Prior RE (c) 1.6 Prior RE

Figure 6. Achieved accuracy by varying the sample size.



(a) 0.4 Prior RE (b) 0.8 Prior RE (c) 1.6 Prior RE

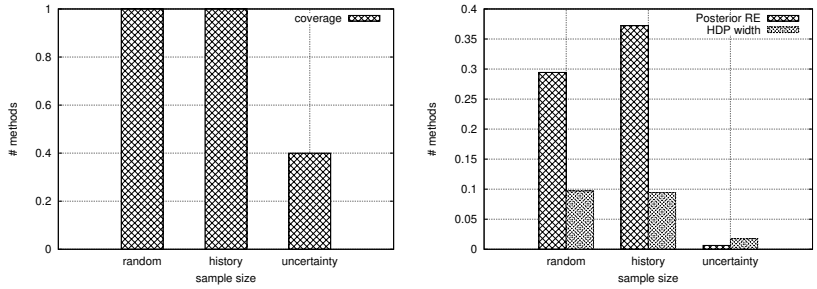
Figure 7. Required effort by varying the sample size.

Results for RQ3: As shown in Table III, to deal with this research question we have run multiple testing activities by taking control over the exploration policy (i.e., the strategy used to generate test cases), the termination condition, and the sample size. During experiments we measured the achieved degree of coverage (in terms of state-actions in the model), the accuracy out of testing (in terms of Posterior RE and HPD width), and the effort (in terms of executed methods in the SUT). Initial beliefs on θ parameters have been fixed using the following values: 0.4 Prior RE and 0.1 HPD width. The sample size has been fixed to $2k$. The three selected strategies are: the uncertainty-aware test case generation, a random walk test case generation approach, and a history-based approach. We let the reader refer to the detailed description provided by Camilli et al. [9] for further information about the two latter traditional model-based testing methods. Data collected during operation is shown in Figure 8 (i.e., coverage and accuracy) and Figure 9 (i.e., effort).

Figure 8a shows the percentage of coverage depending on the selected testing strategy. Traditional pseudorandom methods generally achieve higher coverage values. However, as shown in Figure 8b the achieved accuracy (in terms of Posterior RE and HPD width) is an order of magnitude higher when using the uncertainty-aware approach. Namely, the uncertainty-aware strategy increased achieved accuracy by a factor up to ~ 50. Figure 9a and Figure 9b show the effort (in terms of number of executed methods) depending on the testing strategy and the sample size. The two figures show data extracted from experiments by using alternative scenarios: a simplifies TAS version with decreased structural complexity (to increase the likelihood of hitting θ, i.e., 0.33 using a random walk) and the regular TAS version having higher structural complexity (to decrease the likelihood of hitting θ, i.e., 0.11 using a random walk). Random and the history-based strategies are comparable in terms of required effort. The uncertainty-aware strategy instead is more efficient in both the two scenarios we considered. In particular, with the simplified TAS we measured on average 42% less effort. With increased structural complexity, the convenience of the uncertainty-aware strategy is even more evident. Here, we measured on average 80% less effort. To sum up, the uncertainty-aware strategy is more efficient with respect to traditional model-based testing methods and this is more evident when increasing the complexity of the underlying model (i.e., the lower the likelihood of hitting uncertain regions, the more the convenience of the uncertainty-aware strategy).

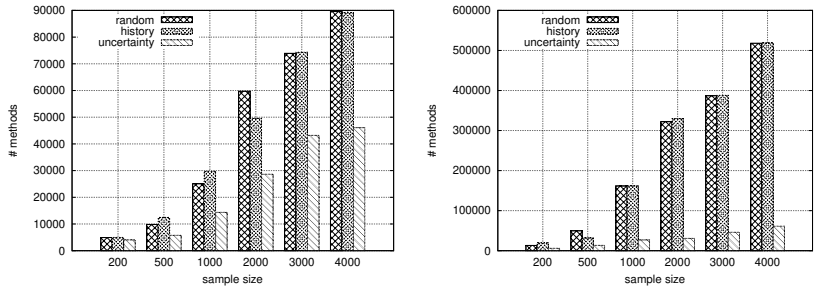
Copyright © 2010 John Wiley & Sons, Ltd.

Prepared using stvrauth.cls
DOI: 10.1002/stvr



(a) Coverage comparison (b) Accuracy comparison

Figure 8. Coverage and accuracy comparison among test case generation strategies.



(a) θ_1 region scenario (b) θ_2 region scenario

Figure 9. Effort comparison among test case generation strategies.

7.4. Discussion

During our practical experience in using METRIC, we found the uncertainty-aware testing is likely to achieve more cost-effectiveness in delivering confidence vs. traditional pseudorandom model-based testing methods. We found a number of configuration settings having a major impact on the overall hypothesis testing process. There exists a trade-off between accuracy and effort while setting up the sample size. High accuracy requires large samples, therefore more effort during testing. This shortcoming is even more severe when using high confidence in wrong beliefs. Whenever the confidence in the hypothesis is low the tester should adopt Prior expressing vague information (i.e., a large HPD width) in order to rely more on the observed data rather than the initial guess. Another important factor is the termination condition. Our experience suggests that traditional termination conditions based on coverage properties are not effective when dealing with uncertainty quantification. The condition we developed, based on the Bayes factor, showed increased effectiveness in terms of accuracy. In particular, the Bayes factor gives the ability of recognizing in a fully automated way the required effort (i.e., when the runtime evidence does not cause significant changes in the Posterior knowledge).

Our empirical evaluation suggests useful insights to test case generation strategies depending on the characteristics of the system under test. Traditional pseudorandom testing can be usually tuned by means of fixed weights that can be used to selectively increase or decrease the probability associated with available actions. Nevertheless, arbitrary complex combination of actions, transition probabilities, and alternative paths makes this approach not feasible in practice. In our experience, the tester can benefit from pseudorandom strategies whenever no clues on how to drive testing exist. However, uniform sampling of the input space (whereby all the available inputs are equally likely to be observed) may lead to an unbalanced model-based exploration decreasing the effectiveness of hypothesis testing of low probability uncertain regions.

Copyright © 2010 John Wiley & Sons, Ltd.

Prepared using stvrauth.cls
DOI: 10.1002/stvr

7.5. Threats to validity

Below, we discuss a number of threats to the validity of experiments. We describe these issues and how we tried to limit them.

External Validity. Generalization of results represent a typical external validity threat in empirical evaluations. Our results have been obtained by conducting a large testing campaign on different versions/configurations of our running TAS example (see Sect. 4) that represents a traditional benchmarking example in the area of service-based (healthcare) systems. Different TAS versions combined with different Priors were synthesized to assess the effectiveness of our approach varying the structural complexity of the underlying MDP model. Additional experiments with case studies in different application domains (e.g., sensor networks, robotics, security protocols) are required to further generalize the results. We tried to reduce threats to external validity by detailing the characteristics (i.e., model, uncertain regions, Priors, values given to independent variables) of every experiment we conducted in order to evaluate the applicability in this specific context.

Internal Validity. To achieve high degree of internal validity in our empirical study, we designed each experiment to have direct manipulation of independent variables. In particular, our experimental environment allows for direct access to both true values associated with θ-parameters and design-time beliefs expressed by Priors. Directly manipulating the overall degree of uncertainty has been fundamental to assess cause-effect relations between external factors and uncertainty quantification capability. In fact, we had the possibility to conduct precise measurements of both initial (before testing) and final (after testing) RE on estimation of θ parameters. This fine grained access to independent variables allows for greater internal validity than conclusions based on an association observed without manipulation. Direct manipulation of independent variables allowed us to create same experiment contexts when varying the model-based test case generation strategy. This means we used the same criteria to empirically evaluate uncertainty quantification capability during testing for each considered MBT strategy (i.e., both uncertainty-aware strategy and uncertainty-unaware strategies).

Conclusion Validity. Since model-based test case generation algorithms are governed by probabilistic functions there exists the possibility that results have been produced by chance as described by Wang et al. [32]. We reduced this threat to validity by repeating experiments multiple times and using for each experiment a very large sample size (between 200 and 4000). Following the practical guidelines introduced in [31], during the comparative evaluation we have run the MBT algorithms 1000 times. To detect statistical differences, we conducted a pairwise comparison using the Mann-Whitney U test to calculate p-value with significance level $\alpha = 0.05$.

Construct Validity. As introduced in de Oliveira Barros et al. [33], we handled major construct threats by assessing the validity of cost-effectiveness measures used during our experiments. The cost of executing MBT algorithms has been measured by considering the number of tests required to achieve termination. Such a selected metric is traditionally accepted as valid in assessing randomized testing algorithms as stated by Arcuri et al. [31]. Similarly, the effectiveness has been measured by adopting standard methods. The magnitude of the improvement has been computed by using the RE (independent from the measurement unit) very often adopted in statistical inference to assess difference between an exact value and an approximation. As a measure of confidence, we adopted the HPD width which yields the highest possible accuracy in estimating the θ parameters. As described Insua et al. [13], this measure is usually adopted in Bayesian statistics as a measure of the confidence gained after obtaining the Posterior knowledge.

8. RELATED WORK

Uncertainty mitigation has received a lot of attention in different fields of software engineering. The work presented in this article has been mainly influenced by different lines of research. In this section, we compare our work with selected literature that we considered the most relevant to our context, grouping them according to the research lines that inspired our approach.

Copyright © 2010 John Wiley & Sons, Ltd.
Prepared using stvrauth.cls

DOI: 10.1002/stvr

Design-time Uncertainty Assessment. A popular technique to deal with uncertainty at design-time is parametric model checking [34]. It follows a *Forward Uncertainty Propagation* [20] approach which focuses on the influence on the model outputs from the parametric variability in the sources of uncertainty. Our technique follows instead a IUQ (inverse) approach to estimate the discrepancy between experimental data (from a real system) and the mathematical model.

Many pieces of work, inside the community of self-adaptive systems, aim at making adaptation decisions taking into account the sources of uncertainty. Notable examples include recent activities by Camilli et al. [35], Ramirez et al. [19], and Esfahani et al. [1]. Most of them employ Markov Models to express uncertain QoS properties and make decisions under probability theory. These approaches are promising, but they are known to be computationally expensive for use at runtime, where often decisions have to be made very fast [1].

An interesting effort has been shown in recent research activities by Calinescu et al. [36]. Authors focus primarily on design-time verification aspects to assure quality-of-service (QoS) properties (reliability, performance, and others) of systems that exhibit stochastic behavior. The presented theoretical framework and associated toolchain aim at establishing confidence intervals for the QoS properties of a software system modeled as a Markov chain with uncertain transition probabilities, i.e. transition probabilities are unknown but observations of these transitions are available.

Design-time UML-based models of uncertainty have been proposed by Ma et al. [37]. The approach is called fuzzy UML data modeling and it relies on *fuzzy set* and *possibility distribution* theories. The Fuzzy Description Logic [38] represents an extension of this model and it aims at checking correctness of fuzzy properties. These pieces of work focus on the analyses at the design time, whereas METRIC focuses on testing.

Bayesian Reasoning. The usage of Bayesian reasoning has been mainly influenced by different existing approaches [39, 40]. Bayesian estimators [13, 23] have recently gained high interest for online calibration thanks to the ease with which the basic ideas are put into place. In fact, because of the Markov property [41], the learning problem using Bayesian inference usually reduces to the independent learning of different independent categorical distributions [13]. Moreover, convergence is usually fast and the Bayesian approach allows expert knowledge to be embedded in the inference framework. The approaches described by Filieri et al. [39] apply Bayesian reasoning to calibrate transition probabilities of Discrete Time Markov Chains kept alive along with the running system in production. Improvements of these approaches have been proposed by Calinescu et al. [42] and Filieri et al. [43] in order to alleviate the negative effect of historical data on the estimation by using aging mechanisms (e.g., Kalman filters [44]) to discard old information.

Model-based Testing. A comprehensive survey on MBT approaches has been presented by Dias et al. [45]. Here, the strategy for test case generation is highlighted as one of the main challenges of this domain. In the following we discuss some recent activities dealing with this issue and proposing different alternatives as model-based exploration methods. Work presented by kastner et al. [46] introduces the idea of variability-aware testing in product-lines. Roughly speaking, the idea is to analyze the product generator instead of the generated products in order to minimize the effort and maximize the accuracy. Experiments are carried out by generating test cases that are simulated using a model checker. The testing strategy introduced by Fraser et a. [47] is guided by behavioral coverage. Namely, machine learning algorithms are employed to augment standard syntactic testing. This is used in turn as a test case selection criterion. Arcaini et al. [48] proposed to divide a system model into subsystems leveraging the notion of independent variables. Here, tests generated for single subsystems are then combined in a bottom-up fashion to obtain tests for the whole system. All these recent activities aim at developing MBT techniques with optimized test case generation. Nevertheless, none of these approaches considers system uncertainties for such a scope.

Testing Under Uncertainty. When jointly considering uncertainty quantification and testing, there are few and recently defined approaches that deal with MBT driven by uncertainty assessment. We here give a brief overview of the most significant ones.

The basic idea of on-the-fly MBT is not new. It has been introduced in the context of labeled transition systems using both *ioco theory*, as described by Bijl et al. [26] and *alternating simulation*, as introduced by Veanes et al. [15]. MBT under uncertainty, in particular, has been considered

Copyright © 2010 John Wiley & Sons, Ltd.
Prepared using stvrauth.cls

DOI: 10.1002/stvr

to improve dependability of Cyber-Physical Systems (CPSs) by defining frameworks supporting MBT of CPSs under uncertainty in a cost-effective manner. *UncerTum* [49], for example, is a test modeling framework providing a UML profile and model libraries to capture uncertainty explicitly in UML state machine models. The same authors propose the testing framework *UncerTest* [50]. UncerTest provides a set of uncertainty-wise test case generation and test case minimization strategies that rely on UML test ready models explicitly specifying subjective uncertainty. Their methodology is based on a so called *Uncertainty Theory* [51] and multi-objective search [52]. UncerTest uses an off-line test case generation approach. To avoid explosion in this process they introduce cost-effective minimization algorithms (i.e., uncertainty-wise test minimization) aiming at reducing the number of test cases but maintain coverage of models. To increase chances of occurrences of uncertainties, UncerTest enable occurrences of uncertainty sources specified by means of design-time models. Since an uncertainty may have multiple sources, authors propose a set of strategies to decide which sources to introduce, where to introduce them and how to introduce them during test generation. Here, major objective of testing is discovering unknown occurrences of uncertainties (i.e., uncertainties occurred with unknown sources) by observing the existing causal relationship between uncertainties and their (known or unknown) sources. Differently from UncerTest, METRIC adopts on-the-fly approaches to generate test cases and it is mainly concerned with hypothesis testing rather than discovering unknown sources. Another approach that focuses on fault detection has been recently introduced by Ma et al. [53]. Authors deal with testing of self-healing CPSs in the presence of sources uncertainty. In their vision uncertainties are treated as controllable quantities during testing. Such a process aims at learning optimal policies for invoking operations and introducing uncertainties, respectively, to effectively detect faults.

The *Active Learning* strategy to black-box test case generation has been proposed by Walkinshaw et al. [54] to select test cases for execution to decrease uncertainty about the correctness of a software system. This work aims at overcoming the problem of intractability in MBT and generating test cases which the inferred model is "least certain" about. Our approach deals with intractability by using an online approach that stochastically samples alternative choices, rather than exhaustively enumerate them.

The iterative MBT approach introduced by Ji et al. [55] aims at generating evolving test models to discover unknown behaviors in uncertain network conditions. Similarly to UncerTest, authors propose a modeling approach based on UML state diagrams. Model-based test case generation is then used to discover uncertain behavior as follows. Test cases lead the SUT to states where possible uncertain behavior can be observed inducing particular environment settings (i.e., user operations and network conditions).

Wang et al. [56] introduced an approach for testing timed systems under uncertainty. The approach aims at the automatic generation of stochastic oracles that verify the capability of a software system to fulfill timing constraints in the presence of time uncertainty is proposed. Such stochastic oracles entail the statistical analysis of repeated test case executions based on test output probabilities predicted by means of statistical model checking. In their context, uncertainty is caused by the inherent concurrent and indeterminate nature of timed systems.

To the best of our knowledge, the METRIC approach differs from the state-of-the-art in literature since it enhances current on-the-fly MBT techniques with an uncertainty-aware test case generation strategy having the ability of learning from runtime data to calibrate the initial assumptions.

9. CONCLUSION AND FUTURE DIRECTIONS

In this article, we introduced the METRIC methodology, its theoretical foundation and the current software toolchain that aims at aiding the software deployment process by mitigating sources of uncertainty through testing.

Contribution. METRIC makes use of a novel on-the-fly MBT technique that combines test case generation guided by uncertainty-aware strategies and Bayesian inference. The key ideas are: (i) explicitly include the notion of uncertainty inside the system specification; (ii) exploit this definition to drive model-based test case generation in order to observe the software product in its

Copyright © 2010 John Wiley & Sons, Ltd.
Prepared using stvrauth.cls

DOI: 10.1002/stvr

own uncertain components; (iii) quantify the design-time uncertainty by using hypothesis testing approaches.

Results. To study the effectiveness of METRIC, we performed a validation activity through a large testing campaign within a number of scenarios constructed using our TAS running example. The experience collected during our experimentation suggests that the uncertainty-aware test case generation strategy, paired with the termination condition based on the Bayes factor, outperforms traditional pseudorandom testing methods in terms of cost-effectiveness. In particular, our comparative empirical evaluation shows that, under same effort constraints, our uncertainty-aware testing strategy increases the accuracy of the uncertainty mitigation by a factor of ~ 50 with respect to traditional model-based testing methods. Furthermore, under same termination conditions, the uncertainty-aware testing strategy requires up to 80% less effort in terms of number of executed methods along the whole testing process. The METRIC toolchain has been released as open source software to allow and encourage the replication of experiments.

Challenges Ahead. To the best of our knowledge, there are no approaches providing fine grained test generation strategies based on the nature of the uncertainty and/or structural properties of the model. This represents a challenge that calls for novel uncertainty-aware testing methods able to increase delivered confidence. In general, the ability of collecting enough evidence from multiple uncertain regions is affected by structural properties of the SUT model that might exhibit trap-like regions [57]. These regions may cause unbalanced model-based exploration leading to uneven distribution of confidence over θ parameters.

APPENDIX A LIST OF ACRONYMS

CPS	Cyber-Physical System.
CTL	Computation Tree Logic.
DSL	Domain Specific Language.
FUP	Forward Uncertainty Propagation.
HPD	Highest Posterior Density.
IOCO	Input/Output Conformance.
IUQ	Inverse Uncertainty Quantification.
MBT	Model-based Testing.
MDP	Markov Decision Process.
METrIC	Modeling & vErification, Testing, Inference & Calibration.
PCTL	Probabilistic Computation Tree Logic.
QoS	Quality of Service.
RE	Relative Error.
SUT	System Under Test.
TAS	Tele Assistance System.
UI	User Interface.
UML	Unified Modeling Language.

REFERENCES

1. Esfahani N, Malek S. Uncertainty in self-adaptive software systems. *Software Engineering for Self-Adaptive Systems II: Int. Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers*. Springer Berlin Heidelberg: Berlin, Heidelberg, 2013; 214–238, doi:10.1007/978-3-642-35813-5_9.
2. Garlan D. Software engineering in an uncertain world. *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, FoSER '10, ACM: New York, NY, USA, 2010; 125–128, doi:10.1145/1882362.1882389. URL http://doi.acm.org/10.1145/1882362.1882389.
3. Broy M, Jonsson B, Katoen JP, Leucker M, Pretschner A. *Model-Based Testing of Reactive Systems: Advanced Lectures (Lecture Notes in Computer Science)*. Springer-Verlag New York, Inc.: Secaucus, NJ, USA, 2005.
4. Arendt PD, Apley DW, Chen W. Quantification of model uncertainty: Calibration, model discrepancy, and identifiability. *J. Mech. Des.* 2012; **134**(10):100 908, doi:10.1115/1.4007390.
5. Camilli M, Gargantini A, Scandurra P, Bellettini C. Towards inverse uncertainty quantification in software development (short paper). *Software Engineering and Formal Methods - 15th International Conference, SEFM*

Copyright © 2010 John Wiley & Sons, Ltd.
Prepared using stvrauth.cls

2017, Trento, Italy, September 4-8, 2017, Proceedings, *Lecture Notes in Computer Science*, vol. 10469, Cimatti A, Sirjani M (eds.), Springer, 2017; 375–381, doi:10.1007/978-3-319-66197-1_24. URL https://doi.org/10.1007/978-3-319-66197-1_24.
6. Berger J. *Statistical Decision Theory and Bayesian Analysis*. Springer Series in Statistics, Springer, 1985.
7. Puterman ML. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. 1st edn., John Wiley & Sons, Inc.: New York, NY, USA, 1994.
8. Aziz A, Singhal V, Balarin F, Brayton RK, Sangiovanni-Vincentelli AL. It usually works: The temporal logic of stochastic systems. *Computer Aided Verification*, Wolper P (ed.), Springer Berlin Heidelberg: Berlin, Heidelberg, 1995; 155–165.
9. Camilli M, Bellettini C, Gargantini A, Scandurra P. Online model-based testing under uncertainty. *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*, 2018. To appear.
10. Camilli M, Gargantini A, Madaudo R, Scandurra P. Hyppotest: Hypothesis testing toolkit for uncertain service-based web applications. *Integrated Formal Methods*, Ahrendt W, Tapia Tarifa SL (eds.), Springer International Publishing: Cham, 2019; 495–503.
11. Forejt V, Kwiatkowska M, Norman G, Parker D. Automated verification techniques for probabilistic systems. *Formal Methods for Eternal Networked Software Systems: 11th International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM 2011, Bertinoro, Italy, June 13-18, 2011. Advanced Lectures*. Springer Berlin Heidelberg: Berlin, Heidelberg, 2011; 53–113, doi:10.1007/978-3-642-21455-4_3. URL https://doi.org/10.1007/978-3-642-21455-4_3.
12. Baier C, Katoen JP. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.
13. Insua D, Ruggeri F, Wiper M. *Bayesian Analysis of Stochastic Process Models*. Wiley Series in Probability and Statistics, Wiley, 2012.
14. Gerhold M, Stoelinga M. Model-based testing of probabilistic systems. *Formal Aspects of Computing* Jan 2018; **30**(1):77–106, doi:10.1007/s00165-017-0440-4. URL https://doi.org/10.1007/s00165-017-0440-4.
15. Veanes M, Campbell C, Schulte W, Tillmann N. Online testing with model programs. *Proceedings of the 10th European Software Engineering Conf. / 13th ACM Int. Symp. on Foundations of Software Engineering*, 2005; 273–282, doi:10.1145/1081706.1081751.
16. de Alfaro L. Game models for open systems. *Verification: Theory and Practice: Essays Dedicated to Zohar Manna on the Occasion of His 64th Birthday*, Dershowitz N (ed.). Springer Berlin Heidelberg: Berlin, Heidelberg, 2003; 269–289, doi:10.1007/978-3-540-39910-0_12. URL https://doi.org/10.1007/978-3-540-39910-0_12.
17. Ramirez AJ, Jensen AC, Cheng BHC. A taxonomy of uncertainty for dynamically adaptive systems. *Proc. of the 7th Int. Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2012; 99–108.
18. Perez-Palacin D, Mirandola R. Uncertainties in the modeling of self-adaptive systems: A taxonomy and an example of availability evaluation. *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering*, ICPE '14, ACM: New York, NY, USA, 2014; 3–14, doi:10.1145/2568088.2568095. URL http://doi.acm.org/10.1145/2568088.2568095.
19. Ramirez AJ, Jensen AC, Cheng BHC. A taxonomy of uncertainty for dynamically adaptive systems. *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '12, IEEE Press: Piscataway, NJ, USA, 2012; 99–108. URL http://dl.acm.org/citation.cfm?id=2666795.2666812.
20. Lee SH, Chen W. A comparative study of uncertainty propagation methods for black-box-type problems. *Structural and Multidisciplinary Optimization* 2008; **37**(3):239, doi:10.1007/s00158-008-0234-7.
21. Camilli M, Bellettini C, Capra L, Monga M. A formal framework for specifying and verifying microservices based process flows. *Software Engineering and Formal Methods*, Cerone A, Roveri M (eds.), Springer International Publishing: Cham, 2018; 187–202.
22. Immonen A, Niemelä E. Survey of reliability and availability prediction methods from the viewpoint of software architecture. *Software & Systems Modeling* 2007; **7**(1):49, doi:10.1007/s10270-006-0040-x. URL http://dx.doi.org/10.1007/s10270-006-0040-x.
23. Robert CP. *The Bayesian Choice: From Decision-Theoretic Foundations to Computational Implementation*. 2nd edn., Springer, 2007.
24. Weyns D, Calinescu R. Tele assistance: A self-adaptive service-based system examplar. *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '15, IEEE Press: Piscataway, NJ, USA, 2015; 88–92. URL http://dl.acm.org/citation.cfm?id=2821357.2821373.
25. Kwiatkowska M, Norman G, Parker D. PRISM 4.0: Verification of probabilistic real-time systems. *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, *LNCS*, vol. 6806, Gopalakrishnan G, Qadeer S (eds.), Springer, 2011; 585–591.
26. van der Bijl M, Rensink A, Tretmans J. Compositional testing with ioco. *Formal Approaches to Software Testing*, Petrenko A, Ulrich A (eds.), Springer Berlin Heidelberg: Berlin, Heidelberg, 2004; 86–100.
27. Diaconis P, Ylvisaker D. Conjugate priors for exponential families. *Ann. Statist.* 03 1979; **7**(2):269–281, doi:10.1214/aos/1176344611. URL https://doi.org/10.1214/aos/1176344611.
28. Kiczales G, Hilsdale E, Hugunin J, Kersten M, Palm J, Griswold WG. An overview of aspectj. *ECOOP 2001 — Object-Oriented Programming*, Knudsen JL (ed.), Springer Berlin Heidelberg: Berlin, Heidelberg, 2001; 327–354.
29. Eysholdt M, Behrens H. Xtext: Implement your language faster than the quick and dirty way. *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*, OOPSLA '10, ACM: New York, NY, USA, 2010; 307–309, doi:10.1145/1869542.1869625. URL http://doi.acm.org/10.1145/1869542.1869625.
30. Tretmans J, Belinfante A. Automatic testing with formal methods. *7th European Int. Conf. on Software Testing, Analysis & Review*, 1999; 8–12.

Copyright © 2010 John Wiley & Sons, Ltd.
Prepared using stvrauth.cls

31. Arcuri A, Briand L. A practical guide for using statistical tests to assess randomized algorithms in software engineering. *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, ACM: New York, NY, USA, 2011; 1–10, doi:10.1145/1985793.1985795. URL http://doi.acm.org/10.1145/1985793.1985795.
32. Wang S, Ali S, Gotlieb A. Minimizing test suites in software product lines using weight-based genetic algorithms. *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, GECCO '13, ACM: New York, NY, USA, 2013; 1493–1500, doi:10.1145/2463372.2463545. URL http://doi.acm.org/10.1145/2463372.2463545.
33. de Oliveira Barros M, Dias-Neto AC. Threats to validity in search-based software engineering empirical studies. *RelaTe-DIA* 2011; **5**(1).
34. Hahn EM, Hermanns H, Wachter B, Zhang L. Param: A model checker for parametric markov models. *Computer Aided Verification*, Touili T, Cook B, Jackson P (eds.), Springer Berlin Heidelberg: Berlin, Heidelberg, 2010; 660–664.
35. Camilli M, Gargantini A, Scandurra P. Zone-based formal specification and timing analysis of real-time self-adaptive systems. *Science of Computer Programming* 2018; **159**:28 – 57, doi:https://doi.org/10.1016/j.scico.2018.03.002.
36. Calinescu R, Ghezzi C, Johnson K, Pezzè M, Rafiq Y, Tamburrelli G. Formal verification with confidence intervals to establish quality of service properties of software systems. *IEEE Trans. Reliability* 2016; **65**(1):107–125.
37. Ma Z, Zhang F, Yan L. Fuzzy information modeling in UML class diagram and relational database models. *Applied Soft Computing* 2011; **11**(6):4236 – 4245, doi:https://doi.org/10.1016/j.asoc.2011.03.020. URL http://www.sciencedirect.com/science/article/pii/S1568494611001165.
38. Ma Z, Zhang F, Yan L, Cheng J. Representing and reasoning on fuzzy uml models: A description logic approach. *Expert Systems with Applications* 2011; **38**(3):2536 – 2549, doi:https://doi.org/10.1016/j.eswa.2010.08.042. URL http://www.sciencedirect.com/science/article/pii/S0957417410008389.
39. Filieri A, Ghezzi C, Tamburrelli G. A formal approach to adaptive software: Continuous assurance of non-functional requirements. *Form. Asp. Comput.* Mar 2012; **24**(2):163–186, doi:10.1007/s00165-011-0207-2. URL http://dx.doi.org/10.1007/s00165-011-0207-2.
40. Filieri A, Tamburrelli G, Ghezzi C. Supporting self-adaptation via quantitative verification and sensitivity analysis at run time. *IEEE Transactions on Software Engineering* Jan 2016; **42**(1):75–99, doi:10.1109/TSE.2015.2421318.
41. Gagniuc PA. *Markov Chains: From Theory to Implementation and Experimentation*. John Wiley & Sons, Inc., 2017, doi:10.1002/9781119387596.
42. Calinescu R, Rafiq Y, Johnson K, Bakir ME. Adaptive model learning for continual verification of non-functional properties. *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering*, ICPE '14, ACM: New York, NY, USA, 2014; 87–98, doi:10.1145/2568088.2568094. URL http://doi.acm.org/10.1145/2568088.2568094.
43. Filieri A, Grunske L, Leva A. Lightweight adaptive filtering for efficient learning and updating of probabilistic models. *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ICSE '15, IEEE Press: Piscataway, NJ, USA, 2015; 200–211. URL http://dl.acm.org/citation.cfm?id=2818754.2818781.
44. Astrom KJ, Wittenmark B. *Computer-controlled Systems: Theory and Design (2Nd Ed.)*. Prentice-Hall, Inc.: Upper Saddle River, NJ, USA, 1990.
45. Dias Neto AC, Subramanyan R, Vieira M, Travassos GH. A survey on model-based testing approaches: a systematic review. *International Workshop on Empirical Assessment of Software Engineering Languages and Technologies*, 2007; 31–36.
46. Kästner C, Von Rhein A, Erdweg S, Pusch J, Apel S, Rendel T, Ostermann K. Toward variability-aware testing. *Proceedings of the International Workshop on Feature-Oriented Software Development*, 2012; 1–8.
47. Fraser G, Walkinshaw N. Assessing and generating test sets in terms of behavioural adequacy. *Software Testing, Verification and Reliability* 2015; **25**(8):749–780.
48. Arcaini P, Gargantini A, Riccobene E. Improving model-based test generation by model decomposition. *Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2015; 119–130.
49. Zhang M, Ali S, Yue T, Norgren R, Okariz O. Uncertainty-wise cyber-physical system test modeling. *Software & Systems Modeling* Jul 2017; doi:10.1007/s10270-017-0609-6. URL https://doi.org/10.1007/s10270-017-0609-6.
50. Zhang M, Ali S, Yue T. Uncertainty-wise test case generation and minimization for cyber-physical systems. *Journal of Systems and Software* 2019; **153**:1–21, doi:10.1016/j.jss.2019.03.011. URL https://doi.org/10.1016/j.jss.2019.03.011.
51. Liu DB. Uncertainty theory. *Uncertainty Theory*. Springer Berlin Heidelberg: Berlin, Heidelberg, 2007; 205–234, doi:10.1007/978-3-540-73165-8_5. URL https://doi.org/10.1007/978-3-540-73165-8_5.
52. Nebro AJ, Durillo JJ, Garcia-Nieto J, Coello Coello CA, Luna F, Alba E. Smpso: A new pso-based metaheuristic for multi-objective optimization. *2009 IEEE Symposium on Computational Intelligence in Multi-Criteria Decision-Making(MCDM)*, 2009; 66–73, doi:10.1109/MCDM.2009.4938830.
53. Ma T, Ali S, Yue T, Elaasar M. Testing self-healing cyber-physical systems under uncertainty: a fragility-oriented approach. *Software Quality Journal* 2018; **27**:615–649, doi:10.1007/s11219-018-9437-3.
54. Walkinshaw N, Fraser G. Uncertainty-driven black-box test data generation. *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, 2017; 253–263, doi:10.1109/ICST.2017.30.
55. Ji R, Li Z, Chen S, Pan M, Zhang T, Ali S, Yue T, Li X. Uncovering unknown system behaviors in uncertain networks with model and search-based testing. *International Conference on Software Testing, Verification and Validation*, 2018; 204–214.
56. Wang C, Pastore F, Briand LC. Oracles for testing software timeliness with uncertainty. *ACM Trans. Softw. Eng. Methodol.* 2019; **28**(1):1:1–1:30, doi:10.1145/3280987. URL https://doi.org/10.1145/3280987.

Copyright © 2010 John Wiley & Sons, Ltd.
Prepared using stvrauth.cls

57. Nijssen S, Back T. An analysis of the behavior of simplified evolutionary algorithms on trap functions. *IEEE Transactions on Evolutionary Computation* Feb 2003; **7**(1):11–22, doi:10.1109/TEVC.2002.806169.

Copyright © 2010 John Wiley & Sons, Ltd.
Prepared using stvrauth.cls

DOI: 10.1002/stvr

Table I. TAS non-functional requirements.

label	description	category	PCTL formula
R_1	The probability of successfully handling a request must be at least 0.98	reliability	$\mathcal{P}_{\geq 0.98}(\mathtt{F}\mathit{success})$
R_2	The probability of successfully handling a request without errors must be at least 0.89	reliability	$\mathcal{P}_{\geq 0.89}(\neg \mathit{error}\ \mathtt{U}\ (\neg \mathit{error} \,\&\, \mathit{success}))$
R_3	The probability of encounter two errors in a single run must be less than 0.009	reliability	$\mathcal{P}_{<0.009}(\mathtt{F}\mathit{error} \,\&\, \mathtt{X}(\mathtt{F}\mathit{error}))$
R_4	The probability of of successfully handling a request between 5 and 7 operations must be at least 0.9	complexity bound	$\mathcal{P}_{\geq 0.9}(\mathtt{F}_{[5,7]}\mathit{success})$
R_5	The expected response time of a fast execution must be less than 2.0	response time	$\mathcal{R}_{\mathit{time}<2.0}(\mathtt{F}\ S_6)$
R_6	The expected energy consumption of a run with less than 10 operations must be less than 15.0	energy consumption	$\mathcal{R}_{\mathit{energy}<15.0}(C^{\leq 10})$

Table II. $\mathcal{P}(s,a)$ using ω^u evaluated on the TAS.

action \ state	S_0	S_1	S_4	S_5-S_{10}
alarmMsg	1/2	0	1/2	0
sendAlarm	0	1	0	0
vitalParamsMsg	1/2	0	0	0
changeDrug	0	0	1/2	0
wait	0	0	0	1

Table III. Design of the evaluation.

research question	independent variables	dependent variables
RQ1	Prior RE, Prior HPD width, sample size	Posterior RE
RQ2	Prior RE, Prior HPD width, sample size	#tests
RQ3	exploration policy, termination condition, sample size	%coverage, Posterior RE, Posterior HPD width, #methods

Copyright © 2010 John Wiley & Sons, Ltd.
Prepared using stvrauth.cls

DOI: 10.1002/stvr