Online Model-based Testing Under Uncertainty

Matteo Camilli and Carlo Bellettini
Dept. of Computer Science
Università degli Studi di Milano
Milano, Italy
Email: {camilli,bellettini}@di.unimi.it

Angelo Gargantini and Patrizia Scandurra
Dept. of Management, Information and Production Engineering
Università degli Studi di Bergamo
Bergamo, Italy
Email: {angelo.gargantini,patrizia.scandurra}@unibg.it

***Abstract*—Modern software systems are required to operate in a highly uncertain and changing environment. They have to control the satisfaction of their requirements at run-time, and possibly adapt and cope with situations that have not been completely addressed at design-time. Software engineering methods and techniques are, more than ever, forced to deal with change and uncertainty (lack of knowledge) explicitly.**

For tackling the challenge posed by uncertainty in delivering more reliable systems, this paper proposes a novel online Model-based Testing technique that complements classic test case generation based on pseudo-random sampling strategies with an uncertainty-aware sampling strategy. To deal with system uncertainty during testing, the proposed strategy builds on an Inverse Uncertainty Quantification approach that is related to the discrepancy between the measured data at run-time (while the system executes) and a Markov Decision Process model describing the behavior of the system under test. To this purpose, a conformance game approach is adopted in which tests feed a Bayesian inference calibrator that continuously learns from test data to tune the system model and the system itself. A comparative evaluation between the proposed uncertainty-aware sampling policy and classical pseudo-random sampling policies is also presented using the Tele Assistance System running example, showing the differences in achieved accuracy and efficiency.

***Index Terms*—Uncertainty Quantification, Reliability under Uncertainty, Bayesian Calibration, Online Model-based Testing.**

I. Introduction

Modern software systems are required to work in different and dynamically changing environments. They have to control the satisfaction of their requirements at run-time, and possibly adapt to cope with situations that have not been fully understood or anticipated at design-time. More importantly, they have to deal with uncertainty introduced both at design-time and at run-time due to the lack of knowledge, such as the lack of control over third-party system components (e.g., native cloud services), humans in the loop, highly configurable environments (such as cloud-based systems), and complex interactions among software, hardware infrastructures and physical phenomena. Today, endowing conventional software engineering methods and techniques with a means to model, quantify, and manage uncertainty explicitly is becoming a crucial and challenging task [1], [2]. In particular, Model-based Testing (MBT) [3] techniques that explicitly consider uncertainty in testing the expected behavior of a system is an unexplored research direction.

This paper presents a novel *online* MBT approach that adopts Bayesian reasoning [4] as learning technique to manage uncertainty. Bayesian and probabilistic techniques really come into their own in domains where uncertainty must be taken into account. In our view, the sources of uncertainty can be mitigated by a incremental learning approach during online MBT. Posterior beliefs can be inferred and used to update the hypothesis about uncertain Quality of Service (QoS) parameters of the formal model describing the behavior of the system. Concretely, we introduce a *uncertainty-aware sampling strategy* that complements classic test case generation (based on pseudo-random sampling strategies) in order to deal with system uncertainty. The proposed strategy maximizes the probability to reach the uncertain components of the system under test (SUT) during test case generation. The online MBT activity uses a *Inverse Uncertainty Quantification* (IUQ) approach [5], [6] to assess the discrepancy between measured data at run-time (while the system executes) and a Markov Decision Process (MDP) model describing the expected behavior (including uncertainty) of the SUT. To this purpose, tests feed a Bayesian inference calibrator that continuously learns from test data to tune the uncertain components of the system model. New estimations, after the online MBT activity, represent the basis of new verification phases and the prior knowledge for future evolutions of the software system. Developers and testers would greatly benefit from such a technique, which could be used to test system's functionality and ensure the system will handle uncertainty during its operation, thus improving its reliability in the presence of uncertainty.

A preliminary sketch of our approach, to organize and share the idea, appeared in the position paper [6]. Here, we also present a toolchain supporting our methodology, and report the results of a comparative evaluation between the proposed uncertainty-aware sampling policy and classical pseudo-random sampling policies using the Tele Assistance System (TAS) running example [7].

The main contributions of this paper can be summarized as follows:

1) We introduce and formalize a novel online MBT technique that complements classic test case generation with an uncertainty-aware sampling strategy.

2) We describe how online MBT can be used to feed a Bayesian inference calibrator that continuously learns from test data to perform IUQ.

3) We present the evaluation of both the accuracy and

the efficiency of our IUQ process depending on the selected sampling strategy, thus showing the effectiveness of our uncertainty-aware technique.

The remainder of this paper is structured as follows. Sect. II briefly recalls background concepts on MDPs, the Probabilistic Computation Tree Logic (PCTL) extended with reward properties, and Bayesian inference. Sect. III presents the TAS running example, used throughout the paper to illustrate and validate the approach. Sect. IV introduces the workflow and toolchain of our online MBT approach. Sect. V formalizes the underlying framework and the uncertainty-aware sampling strategy. Sect. VI reports our experience in validating and comparing our uncertainty-aware sampling strategy with classical pseudo-random sampling strategies. Sect. VII discusses related work, and Sect. VIII reports our conclusions and future work.

II. Background concepts

This section provides some background concepts related to Markov Decision Processes, the temporal logic PCTL with rewards, and the Bayesian inference.

A. Markov Decision Processes with rewards

We adopt MDPs for probabilistic modeling of systems with uncertainty and rewards. MDPs [8], [9] represent a widely used formalism for modeling systems exhibiting both probabilistic and nondeterministic behavior. Formally, a MDP is defined as a tuple $\mathcal{M} = (S, s_0, A, \delta, L)$ where: S is a finite set of states; $s_0 \in S$ is an initial state; A is a finite alphabet of actions; $\delta : S \times A \rightarrow Dist(S)$[1] is a partial probabilistic transition function; and $L : S \rightarrow 2^{AP}$ 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 actions available from state s: $A(s) = \{a \in A : \exists \delta(s, a)\}$; (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 from s to s' occurs. Note that δ satisfies $\sum_{s'} \delta(s, a)(s') = 1$, foreach s, a and successor state s'.

To perform quantitative analysis on MDP models, a probability space over infinite paths must be constructed. This is achieved by solving nondeterminism using a specific *policy*, which chooses an action for each state of the MDP, usually depending on the execution history. Formally, given the set of all finite paths $FPath_{\mathcal{M}}$ of a MDP $\mathcal{M}$, a *policy* is a function $\sigma : FPath_{\mathcal{M}} \rightarrow Dist(A)$, such that $\sigma(\rho)(a) > 0$ only if $a \in A(last(\rho))$, where $last(\rho)$ is the last state in the path ρ. Thus, an action is chosen (from the actions available in the current state) randomly depending on the full history of the MDP. A policy σ is *deterministic* if $\sigma(\rho)$ is a *point distribution*[2] for all $\rho \in FPath_{\mathcal{M}}$.

MDPs can be augmented with *reward structure*s, useful for representing quantitative information about the system such as response time or energy consumption. A reward structure for a MDP $\mathcal{M}$ is defined as a pair $r = (r_s, r_a)$ composed of a *state* reward function $r_s : S \rightarrow \mathbb{R}_{\geq 0}$ and an *action* reward function $r_a : S \times A \times S \rightarrow \mathbb{R}_{\geq 0}$ that assign reward values to states and transitions, respectively.

B. Probabilistic Computation Tree Logic

PCTL [10] is a well-known probabilistic extension of the temporal logic CTL [11] for specifying properties over MDPs. In particular, PCTL adds the *probabilistic path operator* that takes as parameter a path formula ψ. Intuitively, a state s satisfies the formula $\mathcal{P} \bowtie p[\psi]$ if, under any policy, the probability of taking a path from s satisfying ψ is in the interval specified by $\bowtie p$, where $\bowtie \in \{\leq, <, \geq, >\}$ and $p \in [0, 1]$. The properties verified on the TAS (Sect. III) are written as reward-based properties in PCTL by using the *reward operator* $\mathcal{R}^r \bowtie x[\phi]$. Intuitively, $\mathcal{R}^r \bowtie x[\phi]$ holds in s if the expected reward cumulated along the path originating from s expressed by ϕ and considering the reward structure r, meets the bound $\bowtie x$, where $x \in \mathbb{R}_{\geq 0}$.

A comprehensive theoretical treatment of PCTL extended with reward-based properties can be found in [11].

C. Bayesian Inference

This section provides a summary of the main key points of this approach. We refer the reader to [4] for further details on this topic. The Bayesian approach represents a very popular framework for inference and prediction. In particular, a very common goal in statistics is to learn about one (or more) uncertain/unknown parameters θ describing some details of a stochastic phenomenon of interest. To learn about θ, we observe the phenomenon of interest and we collect a data sample $y = (y_1, y_2, ..., y_n)$ in order to compute the conditional density $f(y|\theta)$ of the observed data given θ (i.e., 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 [12]. This information is represented by the *Prior* distribution $f(\theta)$. By combining the Prior and the likelihood using the Bayes' theorem (see Equation 1) we obtain the *Posterior* distribution $f(\theta|y)$, describing the best knowledge of the true value of θ, given the data sample y.

$$f(\theta|y) \propto f(\theta) \cdot f(y|\theta) \tag{1}$$

The Posterior distribution can be used in turn to perform point and interval estimation of the uncertain parameters. This is typically addressed, in the multivariate case, by summarizing the distribution through the Posterior *mean* (Equation 2) and the shortest possible region of probability 0.95 (Equation 3), that is the *Highest Posterior Density* (HPD) region [13] defined by the set of θ-values $\mathcal{C}$.

$$E[\theta|y] = \int \theta \cdot f(\theta|y) d\theta \tag{2}$$

$$\mathcal{C} = \{\theta : f(\theta|y) \geq 0.95\} \tag{3}$$

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

[2] The point distribution over a countable set S, is the distribution that assigns probability 1 to a single element $s \in S$.

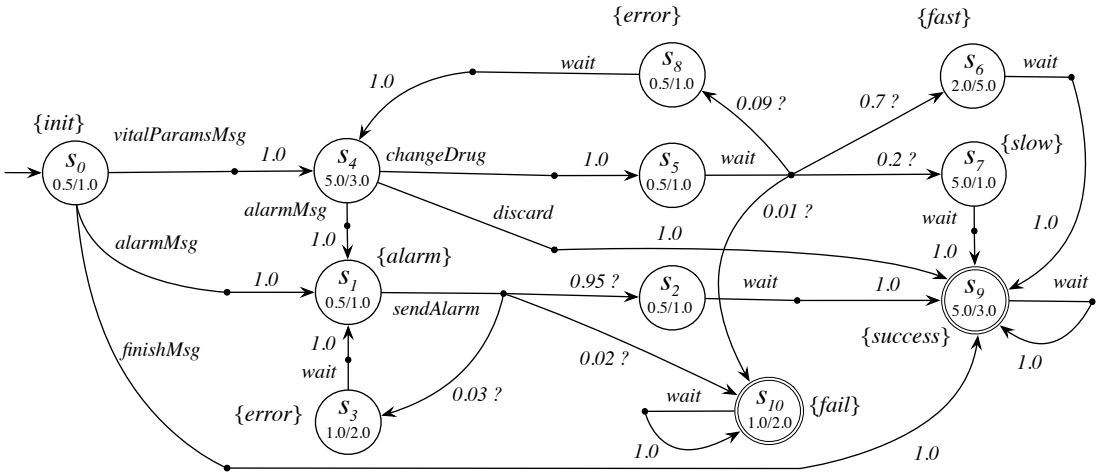


Fig. 1: Markov Decision Process of the TAS behavior.

III. Running Example

In this section we revisit the Tele Assistance System (TAS) exemplar [7]. We use it throughout the paper as running example. The TAS is a service-based system (SBS) 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. The TAS is an example of a wide category of SBSs (e.g., e-commerce, e-health, online banking, taxi-hailing, etc.) characterized by a workflow of interactions with distributed components [14] (e.g., web-services or microservices) owned by multiple third-party providers with different QoS (e.g, reliability, performance, cost, etc.). In this context, our approach to IUQ can be used to calibrate the initial design-time model of the system behavior and verify that the running implementation does not violate specific quality requirements. Therefore, the overall functional and non-functional quality of the final system also depend on the capability of the external services to comply with the assumptions made to design the application workflow.

Figure 1 shows a formalization of the TAS behavior by means of a MDP. The TAS workflow 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 invocation of a pharmacy service (i.e., *changeDrug* action) to add/deliver new medication to the patient or to change the medication dose, or the invocation of an alarm service eventually leading to dispatch a first aid team to the patient home (i.e., *alarmMsg* action). The same 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 (with probability 0.95), or exhibit a faulty behavior such as data loss on the communication channel or an unexpected exception (with probability 0.03), or exhibit a failing behavior such as unreachability of the alarm service (with probability 0.02). 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), or succeed very fast (probability 0.7), or fail with a recoverable error (probability 0.09), or fail in an unrecoverable manner (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 SBSs: *network data loss*, *service failure*, *service response time*, *inadequate design*, and so forth [15]). Uncertain transition probabilities, identifying the set of uncertain/unknown parameters θ, are explicitly represented in Figure 1 by numeric values followed by question marks. Lists of labels associated with states (e.g., $\{alarm\}$) represent the labeling function L. Moreover, 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.

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 [16]. However, the uncertainty, discussed early on, can harm 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

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	$\mathcal{P}_{\geq 0.91}(\neg \mathit{error}\ \mathtt{U}\ (\neg \mathit{error}\ \&\ \mathit{success}))$
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	$\mathcal{R}_{\mathit{energy}<15.0}(C^{\leq 10})$

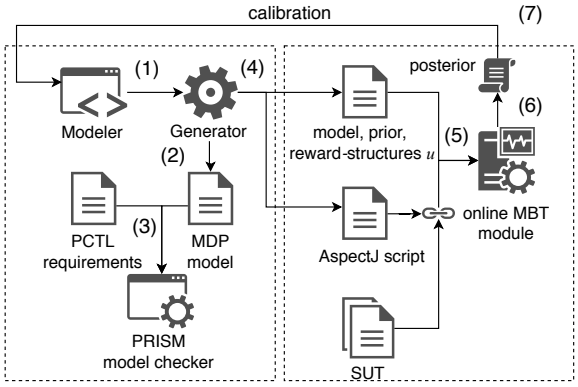


Fig. 2: Online MBT under uncertainty workflow and toolchain.

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.

IV. Online MBT under uncertainty overview

This section provides an overview of our online MBT under uncertainty framework. Fig. 2 sketches the workflow and the toolchain of the proposed framework. It has two main components: the design-time *modeling & verification front end*, and the online *MBT & Inference back end*.

A prototype [3] of the proposed toolchain has been developed using the Java programming language and the frameworks Xtext/Xtend [17] and AspectJ [18]. The main steps supported by the toolchain along with technical details on the front end and back end components are reported in the following.

A. Front end: Modeling & verification

The front end contains the *Modeler*, the *Generator*, and the PRISM model checker. The Modeler is an Eclipse IDE plug-in supporting standard editing features (syntax highlighting, error checking, auto-completion, etc.). The Modeler allows the user to define a MDP model of the system's behavior including the uncertain parameters, the Prior distributions, and the connection to the SUT. All these concepts can be defined using a textual Domain Specific Language (DSL) defined by means of the grammarware framework Xtext. As an example, Listing 1 reports an extract of the TAS MDP specification using the DSL. The language allows the MDP to be defined intuitively by means of the keywords `actions`, `states`, and `arcs`. Moreover, states can be augmented with Prior distributions (using the keyword `Dir`) describing the hypothesis on the uncertain parameters θ attached to the outgoing edges. Here, we abstract from the details of the Prior definition. This topic will be discussed later on in Sect. V.

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` and `control`. Essentially, we follow a common notation introduced in [3] 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 of the methods). The `control` keyword is used to define a mapping between states of the model and states of *quiescence* [19] of the SUT, i.e., states where the SUT expects inputs from the external environment using the available SUT APIs. Inputs depend on the actions available from the quiescent states. Inputs must be declared in the `actions` section by mapping actions to Java objects (e.g., Strings "`v`" and "`a`", associated to the actions `vitalParamsMsg` and `alarmMsg` actions, respectively).

The Generator has been implemented using the Xtend language. It translates the textual MDP model into the PRISM model checker input file, and generates the software artifacts used as input by the online MBT module. These last include: the AspectJ instrumentation script (a file `.aj`) that allows the SUT to be linked to the MBT structure taking into account the binding, and a concise textual representation of the MDP model (a file `.jmdp`) equipped with the priors and the reward structures needed by the MBT module to carry out Bayesian inference while observing the behavior of the SUT.

As part of the front end, desired properties of the system can be expressed in PCTL and verified against the MDP model

[3]The toolchain has been released as open source software 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 example.

```
model "tas−model"
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.

of the system using the probabilistic model checker PRISM. We focus on the verification of non-functional requirements such as reliability and performance (see Table I). Design-time model checking serves therefore as a means to verify the desired requirements against the system model with some degree of uncertainty that becomes quantified later on, during the online MBT and inference phase.

B. Back end: Online MBT & Inference

The back end has been implemented as a JAVA program and it is distributed along with a JAVA MDP simulator to allow users to easily try out the capability of our IUQ approach. Here, we assume that the system implementation, reifying the MDP model, is already available. A complete description of this phase is beyond the scope of this paper. The customized test instrumentation is realized by generating a collection of ASPECTJ pointcuts and advices that allow controllable/observable methods to be handled at runtime by the MBT module. More precisely, the instrumentation provides a serialized view of the execution of observable methods in order to gather data used to perform the inference activity. Controllable methods are handled by supplying external inputs using the available APIs. This is realized by injecting specific input arguments upon the execution of the controllable methods during testing.

The online MBT module derives tests from the MDP specification by stochastically sampling the state space at runtime. Nondeterminism in the MDP is used to capture the possible ways that a *controller* has to influence the behavior of the system by means of *controllable* actions. Thus, our algorithm choses over different available actions using specific *sampling policie*s (selected by the user) which determine the exploration strategy of the SUT.

In particular, we introduce a *uncertainty*-aware sampling policy that allows the probability to reach the uncertain regions of the SUT to be maximized during test case generation.

During testing, the back end collects data and performs a Bayesian inference activity to compute the Posterior distributions associated with the uncertain parameters θ. The online MBT module gives as output the summarization of the posterior distributions computed during testing activity. Information given by the summarization can be used in turn to perform point and interval estimation and calibrate the initial MDP model thus updating the knowledge on the uncertain parameters.

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

V. FORMAL FRAMEWORK

In this section we provide a formalization of the proposed online MBT approach. After providing some preliminary definitions about expressing the SUT behavior as a MDP, we describe how the conformance relation is realized in our MBT technique, the supported sampling policies for test case generation, and the mathematical machinery involved during the model calibration process including the termination conditions.

A. Preliminary definitions

Typically a SBS is composed of a collection of APIs allowing the actual SUT to receive inputs from the external environment. The APIs are connected to a customized test instrumentation that provides a particular high-level view of the behavior of the SUT matching the abstraction level of the MDP specification. The instrumentation provides also a view of the observable actions (i.e., execution of tasks or subroutines of interest) resulting from the stimulation of the SUT by means of the external inputs. In this sense the SUT is an *open system* [20]. This mathematical abstraction reflects the classic transition-based interpretation of the behavior of open systems by considering states as configurations or some functional status of the system, in terms of working and/or failed (distributed) components and transitions as task executions causing changes in the configuration.

The MDP model corresponding to the SUT is called *program model* and denoted by $\mathcal{M}_{SUT}$. This model is defined

exploiting the notion of *binding* declared by the user by means of the DSL introduced in Sect. IV. 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 $\mathcal{M}$ and a set of subroutines $H \subseteq P$, a *binding* is a tuple of partial functions $(\mathcal{H}, \mathcal{I}, \mathcal{P}re, \mathcal{P}ost)$ with domain $S \times A$ s.t.,

- $\mathcal{H}(a)$, 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 binding allows the behavior of the SUT to be viewed in the same way as that of the MDP specification. In particular, $\mathcal{M}_{SUT}$ is defined taking into account the specification $\mathcal{M}$ as follows.

Definition 2 (Program model): 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 program model $\mathcal{M}_{SUT} = (S', s'_0, A', \delta', L')$ is a MDP, s.t.,

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

The intuition 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.

B. 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 program model) using the notions of *alternating simulation* and *refinement* as used in [20], [21].

Definition 3 (Alternating simulation): An alternating simulation between $\mathcal{M}$ and $\mathcal{M}_{SUT}$ is a binary relation $\pi \subseteq S \times S'$, s.t. for all $(s, s') \in \pi$,

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

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

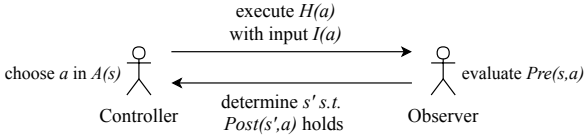


Fig. 3: Conformance game.

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 of any controllable action enabled in the model.

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

The notion of refinement can be explained in terms of a *conformance game* [21], [22] between two players: a *controller* and an *observer* (Figure 3). The game 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.

C. Sampling policies

Online MBT is a technique able to derive tests from a model program 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 (usually performed by classical MBT approaches).

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 sampling policies and termination conditions. The sampling policy is governed by a probabilistic function that has the following form:

$$\mathcal{P}(s, a) = \begin{cases} 0 & \omega(s, a) = 0 \\ \omega(s, a) / \sum_{a' \in A(s)} \omega(s, a') & \text{otherwise} \end{cases} \quad (4)$$

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 to configure the generation of test cases depending on different model-based exploration strategies.

Our framework provides three different sampling policies grounded on *random-based*, *history-based*, and *uncertainty-aware* weight functions, respectively.

1) Random sampling policy: It allows the actions to be selected depending on a statically defined weight function ω^r that maps a pair (s, a) to a fixed value. If $\omega^r(s, a)$ maps to the same value $k \geq 0$ for each $a \in A(s)$, the action is chosen by using a discrete uniform distribution whereby all the available actions are equally likely to be observed. Otherwise, weights can be used to selectively increase or decrease the probability associated with specific state-action pairs. This could be useful to favor the execution of new released functionalities that we may want to stress during testing.

2) History sampling policy: By using this policy, the online MBT algorithm keeps the selected actions balanced during test generation, by taking into account the history of the test run. It is governed by the following weight function.

$$\omega^h(s, a) = \sum_{a' \in A(s)} \#(a') - \#(a) \tag{5}$$

where $\#(a)$ denotes the number of times the action a has been chosen during a test run. Intuitively, this policy is typically adopted when we want to increase the coverage of the available actions by dynamically increasing and decreasing the probability associated with uncovered and covered state-action pairs, respectively. In fact, by using the function ω^h in Equation 4, $\mathcal{P}(s, a)$ becomes inversely proportional to the number of times a has been chosen in s.

3) Uncertainty sampling policy: This policy takes into account the uncertainty explicitly modeled by the user-supplied Prior distributions associated with the θ parameters. In the following we formalize how our algorithm handles the uncertain parameters to solve nondeterminism and guide the generation of the test cases.

The main objective of such a policy is 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.,

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

where $k \in \mathbb{N}_{>0}$.

The intuition of the structure u is to assign a high reward value (k) to transitions of the model associated with uncertain parameters, and a low reward value (0) to the other transitions.

Given an uncertainty-aware reward structure u, the optimal deterministic policy σ^* that maximizes the expected sum rewards can be constructed by computing the optimal value function $V^* : S \to \mathbb{R}_{\geq 0}$ for the *discounted infinite horizon* problem[4], which satisfies the Bellman equation [23]. Namely, V^* is used to produce the optimal policy σ^* as follows.

$$\sigma^*_u(s) = \arg\max_{a \in A(s)} \sum_{s'} \delta(s, a)(s') \cdot (u_a(s, a, s') + \gamma V^*(s')) \tag{6}$$

where $V^*(s')$ represents the expected reward accumulated when starting from state s' and acting optimally along a infinite horizon; $\gamma \in [0, 1]$ represents a discount factor that alleviates the contribution of future rewards in favor of present rewards. The best policy $\sigma^*_u(s)$ returns for each state s the action that allows the reward, considering the uncertainty-aware reward structure u, to be maximized.

The subsets θ_i are partitions of θ constructed by grouping θ-parameters attached to transitions sharing the same source state and action. Namely, $\biguplus_i \theta_i = \theta$ such that,

$$\theta_i = \{\alpha \in \theta \text{ s.t. } \exists a \in A(s_i), s_j \in S : \delta(s_i, a)(s_j) = \alpha\} \tag{7}$$

The intuition of this operation is as follows. We partition θ in order to identify different uncertain regions of the model and then we compute the set of best 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: θ_1 contains the parameters attached to transitions starting from s_1 when choosing the action *sendAlarm*; θ_2 contains the parameters attached to transitions starting from s_5 when choosing the action *wait*.

The set of best policies $\{\sigma^*_{u_i}\}$, is used in turn to construct the uncertainty weight function ω^u as follows.

$$\omega^u(s, a) = \begin{cases} 1 & \exists i : \sigma^*_{u_i}(s) = a \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

The weight function ω^u makes the controller able to stochastically sample the available actions maximizing the probability to reach the uncertain parameters θ.

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

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

[4]A detailed and comprehensive treatment of the discounted infinite horizon problem for MDPs can be found in [8].

D. Inference & Calibration

During testing activity, we set up a Bayesian framework, 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 [13] for more details.

A natural conjugate Prior for the uncertain transition probabilities of a MDP model is defined by letting $p_i^a = (p_{i,j}^a, ..., p_{i,k}^a)$ have a *Dirichlet distribution* [24], where $p_{i,j}^a$ is the probability to observe a transition from s_i to s_j when the action a is chosen.

$$p_i^a \sim Dir(\alpha_i), \text{ where } \alpha_i = (\alpha_{i,j}, ..., \alpha_{i,k}) \tag{9}$$

During the conformance game, the observer component collects $n_{i,j}^a$ 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.

$$p_i^a | y \sim Dir(\alpha_i'), \text{ where } \alpha_i' = (\alpha_{i,j} + n_{i,j}^a, ..., \alpha_{i,k} + n_{i,k}^a) \tag{10}$$

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

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 1000 observations.

At termination, our online 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 online 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 (< 0.05 for each parameter). The model 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.

The accuracy and the efficiency of the calibration will be discussed in detail in Sect. VI.

E. Termination conditions

Our online MBT algorithm can use different termination conditions. Two first conditions are classic 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. These termination conditions are typically used in combination with random and history-based sampling policies. The intuition is to test the SUT by generating a pseudo-random permutation of actions until either a conformance failure is found, or the desired level of confidence has been reached.

In presence of uncertainty, the online MBT algorithm can also use a termination condition based on the precision reached by the inference process. This condition is grounded on a model selection criteria 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.

$$K = \frac{f(y|\theta)}{f(y|\theta')} \tag{11}$$

The two terms of the ratio K represent the likelihood that data y are produced under different assumptions θ and θ'. A positive value of K means that the data under consideration supports more the assumption θ than θ'. The usual interpretation of this value considers $K \in [10^0, 10^{1/2}]$ as not substantial. So, the termination condition reduces to check the K value in this interval. More precisely, given a sample size N, the online MBT algorithm computes the Bayes factor K, for each Posterior under consideration, every N observations. Thus, the convergence of the values K 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.

VI. Comparative Evaluation

The approach has been evaluated through a large simulation campaign using the TAS example. We assessed the accuracy and the efficiency of the calibration of our online MBT technique, configuring the SUT with the desired true θ-values. 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. Each experiment is discussed in the following by reporting the most significant results.

1) Inference activity: A fundamental aspect to discuss is the *sample size*. In fact, it influences both the number of tests needed to achieve termination (based on the convergence of the Bayes factor) and the accuracy of the calibration process. The graph reported in Figure 4 shows the number of tests needed to achieve convergence (left y axis) and the *relative error*[5] (RE) (right y axis) of the Posterior. Data has been extracted by running the inference activity on the TAS example several times varying the sample size (from 200 to 4000). For each different sample size value, we considered three

[5]The relative error describes the discrepancy between an exact value and some approximation of it. It has been computed in a standard way: dividing the absolute error between the true and the inferred θ-values by the magnitude of true θ-values.

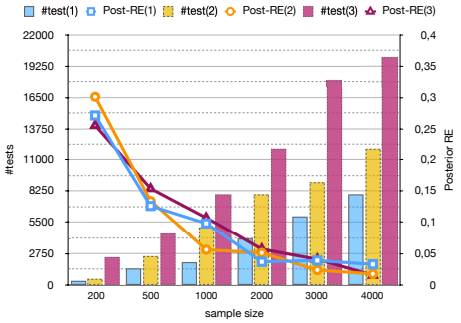
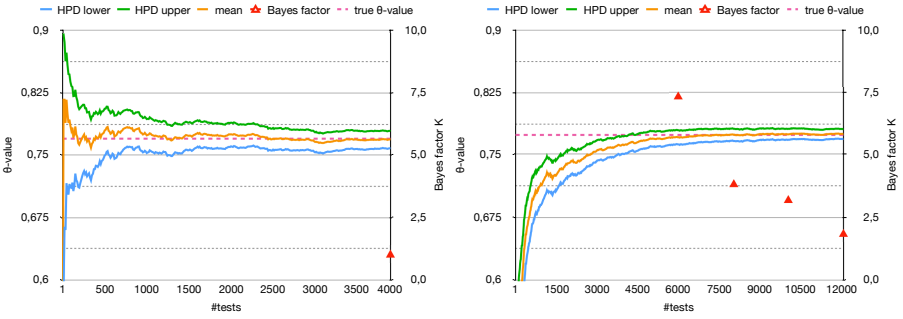


Fig. 4: Tests and Posterior RE depending on the Prior RE and the sample size.



(a) 0.8 HPD region size scenario

(b) 0.1 HPD region size scenario

Fig. 5: Convergence and Bayes Factor depending on the Prior HDP region size.

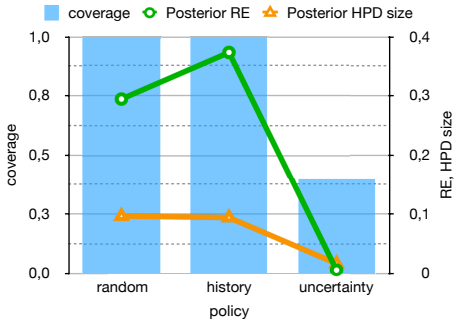
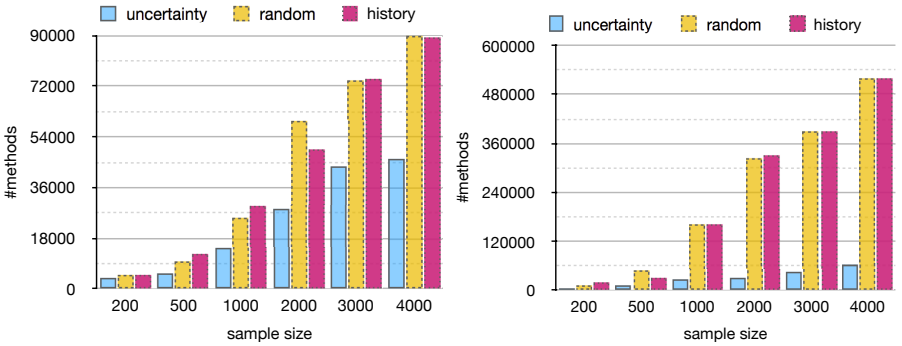


Fig. 6: Inference accuracy depending on the sampling policy.



(a) Simple scenario

(b) Complex scenario

Fig. 7: Inference efficiency depending on the sampling policy and the sample size.

scenarios keeping the Prior HPD region fixed, but varying the Prior RE: (1) 0.4, (2) 0.8, and (3) 1.6. The objective is to evaluate the inference activity starting from different initial hypothesis for the θ-values. In this comparison, we can observe that by increasing the sample size we obtained a smaller Posterior RE. In fact, more data usually leads to compute a more significant Bayes factor, thus allowing for better model selection supported by the data under consideration. We also observed that the calibration achieved almost the same precision, although varying the Prior RE. The number of tests needed to achieve termination increases proportionally over the Prior RE. We empirically found that a sample size of 2000 represents a good give-and-take setting in terms of number of tests and accuracy in terms of Posterior RE (~ 0.05).

Figure 5 shows two graphs where we can observe the convergence of the θ-values and the Bayes factor, during the Bayesian inference process. The two graphs have been drawn by considering two different scenarios where we kept fixed the Prior RE (0.8) and the sample size (2000), but we changed the HPD region size. In particular, the scenario 5a adopts a large HPD region (i.e., low confidence in the hypothesis), while the scenario 5b adopts a small HPD region (i.e., high confidence in the hypothesis). In particular, the two graphs report the convergence of a θ-parameter (with true value set to 0.77), along with the HDP region. We observed that the smaller is the Prior HPD region, the higher is the number of tests needed to achieve termination. For instance, the second scenario (Figure 5b) takes $\sim$ 8000 more tests to achieve termination. The convergence in the first scenario is faster because it relies more on the data under consideration. The Bayes factor K, in this case, falls inside the convergence interval at the second check (i.e., after 4000 samples), while at the first check (i.e., after 2000 samples) $K >> 10$. This means that during the first 2000 samples, the data under consideration make the Posterior change a lot very rapidly. Considering the new estimation (after the first check), the sampled data does not make the Posterior change significantly, thus the inference process stops at the second check. Figure 5b, shows instead that the latter scenario takes 6 iterations to converge. In fact, a very informative (but wrong) Prior determines a slowdown in the Bayesian inference process because its relies more on the hypothesis.

2) Sampling policies: Another important aspect the comparison between the available exploration policies in terms of accuracy and efficiency of the IUQ process performed during the online MBT activity. Figure 6 shows the accuracy in terms of Posterior RE and Posterior HPD region by varying the sampling policy and the termination condition. In particular we ran the online MBT module along with the inference back end to calibrate a set of θ-parameters, starting from a Prior with 0.4 RE and 0.1 HPD region size, and using a sample size of 2000. We considered three different runs as follows. In the first two runs we adopted the classic random-based and history-based sampling policy, respectively. Moreover, we adopted a

classic termination condition based on the total coverage of the available state-action pairs. In the latter run, we used our proposed uncertainty-aware sampling policy coupled with the termination condition based on the convergence of the Bayes factor. Although, we achieve in general higher coverage by using the classic pseudo-random exploration policies, the IUQ activity is way more accurate by adopting our uncertainty-aware approach. In fact, it allows the uncertain parts of the SUT to be stressed until the desired level of confidence has been reached. By using the the uncertainty-aware approach, we observed, in this experiment, an increase in the Posterior accuracy by a factor of 8.04 ± 0.22.

Figure 7 shows the efficiency of the Bayesian inference process when varying the sampling policy and using the convergence of the Bayes factor as termination condition. The two graphs have been drawn by considering two different scenarios where we considered two different uncertain sets of θ-parameters in the TAS example. The two scenarios 7a and 7b consider the parameters θ_1 and θ_2, respectively (introduced in Sect. V). The former scenario is simpler in terms of number of actions and alternative paths leading from the initial state to the θ-parameters. Namely, the probability to take the correct sequence of actions leading to θ_1 (using a discrete uniform distribution for each choice) is 0.33. In the latter scenario, the probability to reach θ_2 (with the same setting) is 0.11. The two graphs show the number of executed methods during the online MBT activity until termination, varying the sampling policy and the sample size. The number of methods executed using the random-based and the history-based sampling policy is very similar. Using instead the uncertainty-aware sampling policy, the convergence is way more efficient. In particular, we can observe from graph 7a that we need on average 42% fewer method executions to achieve termination. Considering the graph 7b, the convenience of using our sampling policy in presence of uncertainty is even more evident. In fact, the uncertainty-aware approach takes on average 80% fewer method executions to achieve termination in this scenario.

VII. Related Work

Uncertainty mitigation has a lot of attention in different fields of software engineering. A popular technique to deal with uncertainty at design-time is parametric model checking [25]. It follows a *Forward Uncertainty Propagation* [26] 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 works, inside the community of self-adaptive systems, aim at making adaptation decisions taking into account the sources of uncertainty. Notable examples can be found in [27], [15], [2], [28]. Most of them employ Markov Models to express uncertain QoS properties by using probability.

The usage of Bayesian reasoning has been mainly influenced by different existing approaches [29], [30]. Bayesian estimators [12], [13] have recently gained high interest for online calibration thanks to the ease with which the basic ideas are put into place. Moreover, convergence is usually fast and the Bayesian approach allows expert knowledge to be embedded in the inference framework. The approaches in [29], [30] 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 can be found in [31], [32]. They aim at alleviating the negative effect of historical data on the estimation, by using aging mechanisms [33] to discard old information.

The employment of these methods in software testing to tackle the IUQ problem is still in its early stages. The *Active Learning* strategy to black-box test case generation has been proposed in [34]. 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 enumerate them. The basic idea of online/on-the-fly MBT is not new. It has been introduced in the context of labeled transition systems using both *ioco theory* [22] and *alternating simulation* [21].

MBT under uncertainty has been considered in the test modeling framework *UncerTum* [35] in the context of Cyber-Physical Systems. It provides a UML profile and model libraries to capture uncertainty explicitly in UML-based test ready models, but it does not provide yet a MBT technique guided by such models.

In summary, to the best of our knowledge, the approach proposed in this paper is a pioneer work in addressing the IUQ problem by combining Bayesian inference and online MBT guided by uncertainty-aware strategies.

VIII. Conclusion and Future Work

In this paper, we introduced a novel approach to quantify and mitigate the sources of uncertainty, before the deployment of a software release build, by combining Bayesian reasoning and online Model-based testing.

The uncertainty quantification, in our view, is part of a stepwise incremental process for building understanding and confidence in system reliability. In this process, our online MBT technique can be applied to: (i) validate the current SUT against the certain components of the formal specification, and (ii) perform interface activity to calibrate the uncertain QoS parameters. To this end, it dynamically generates test cases using a specific *uncertainty*-aware exploration policy. We performed a comparative evaluation between the *uncertainty*-aware policy and other classic pseudo-random test case generation strategies, showing the effectiveness of our proposal both in terms of accuracy and efficiency of the IUQ process.

We plan to enhance the toolchain that supports the proposed approach with the ability to perform *sensitivity analysis* [4], to study how the variability of the uncertain parameters in the model can be apportioned to different experimental designs, i.e., values assigned to one or more independent variables such as hardware/software settings, traffic condition in the communication channels, load of different distributed components.

REFERENCES

[1] D. Garlan, "Software engineering in an uncertain world," in *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, ser. FoSER '10. New York, NY, USA: ACM, 2010, pp. 125–128. [Online]. Available: http://doi.acm.org/10.1145/1882362.1882389

[2] N. Esfahani and S. Malek, *Uncertainty in Self-Adaptive Software Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 214–238.

[3] M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, *Model-Based Testing of Reactive Systems: Advanced Lectures (Lecture Notes in Computer Science)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.

[4] J. Berger, *Statistical Decision Theory and Bayesian Analysis*, ser. Springer Series in Statistics. Springer, 1985.

[5] P. D. Arendt, D. W. Apley, and W. Chen, "Quantification of model uncertainty: Calibration, model discrepancy, and identifiability." *J. Mech. Des.*, vol. 134, no. 10, p. 100908, 2012.

[6] M. Camilli, A. Gargantini, P. Scandurra, and C. Bellettini, "Towards inverse uncertainty quantification in software development (short paper)," in *Software Engineering and Formal Methods*, A. Cimatti and M. Sirjani, Eds. Cham: Springer International Publishing, 2017, pp. 375–381.

[7] D. Weyns and R. Calinescu, "Tele assistance: A self-adaptive service-based system examplar," in *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 88–92.

[8] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, 1st ed. New York, NY, USA: John Wiley & Sons, Inc., 1994.

[9] V. Forejt, M. Kwiatkowska, G. Norman, and D. Parker, *Automated Verification Techniques for Probabilistic Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 53–113.

[10] A. Aziz, V. Singhal, F. Balarin, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "It usually works: The temporal logic of stochastic systems," in *Computer Aided Verification*, P. Wolper, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 155–165.

[11] C. Baier and J.-P. Katoen, *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[12] C. P. Robert, *The Bayesian Choice: From Decision-Theoretic Foundations to Computational Implementation*, 2nd ed. Springer, May 2007.

[13] D. Insua, F. Ruggeri, and M. Wiper, *Bayesian Analysis of Stochastic Process Models*, ser. Wiley Series in Probability and Statistics. Wiley, 2012.

[14] M. Camilli, C. Bellettini, L. Capra, and M. Monga, "A formal framework for specifying and verifying microservices based process flows," in *Software Engineering and Formal Methods*, A. Cerone and M. Roveri, Eds. Cham: Springer International Publishing, 2018, pp. 187–202.

[15] A. J. Ramirez, A. C. Jensen, and B. H. C. Cheng, "A taxonomy of uncertainty for dynamically adaptive systems," in *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '12. Piscataway, NJ, USA: IEEE Press, 2012, pp. 99–108. [Online]. Available: http://dl.acm.org/citation.cfm?id=2666795.2666812

[16] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, ser. LNCS, G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806. Springer, 2011, pp. 585–591.

[17] M. Eysholdt and H. Behrens, "Xtext: Implement your language faster than the quick and dirty way," in *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*, ser. OOPSLA '10. New York, NY, USA: ACM, 2010, pp. 307–309. [Online]. Available: http://doi.acm.org/10.1145/1869542.1869625

[18] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold, "An overview of aspectj," in *ECOOP 2001 — Object-Oriented Programming*, J. L. Knudsen, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 327–354.

[19] J. Tretmans and A. Belinfante, "Automatic testing with formal methods," in *7th European Int. Conf. on Software Testing, Analysis & Review*, 1999, pp. 8–12.

[20] L. de Alfaro, *Game Models for Open Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 269–289. [Online]. Available: https://doi.org/10.1007/978-3-540-39910-0_12

[21] M. Veanes, C. Campbell, W. Schulte, and N. Tillmann, "Online testing with model programs," in *Proceedings of the 10th European Software Engineering Conf. / 13th ACM Int. Symp. on Foundations of Software Engineering*, 2005, pp. 273–282.

[22] M. van der Bijl, A. Rensink, and J. Tretmans, "Compositional testing with ioco," in *Formal Approaches to Software Testing*, A. Petrenko and A. Ulrich, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 86–100.

[23] R. E. Bellman, *Dynamic Programming*. Dover Publications, Incorporated, 2003.

[24] P. Diaconis and D. Ylvisaker, "Conjugate priors for exponential families," *Ann. Statist.*, vol. 7, no. 2, pp. 269–281, 03 1979. [Online]. Available: https://doi.org/10.1214/aos/1176344611

[25] E. M. Hahn, H. Hermanns, B. Wachter, and L. Zhang, "Param: A model checker for parametric markov models," in *Computer Aided Verification*, T. Touili, B. Cook, and P. Jackson, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 660–664.

[26] S. H. Lee and W. Chen, "A comparative study of uncertainty propagation methods for black-box-type problems," *Structural and Multidisciplinary Optimization*, vol. 37, no. 3, p. 239, 2008.

[27] M. Camilli, A. Gargantini, and P. Scandurra, "Zone-based formal specification and timing analysis of real-time self-adaptive systems," *Science of Computer Programming*, vol. 159, pp. 28 – 57, 2018.

[28] D. Perez-Palacin and R. Mirandola, "Uncertainties in the modeling of self-adaptive systems: A taxonomy and an example of availability evaluation," in *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '14. New York, NY, USA: ACM, 2014, pp. 3–14. [Online]. Available: http://doi.acm.org/10.1145/2568088.2568095

[29] A. Filieri, C. Ghezzi, and G. Tamburrelli, "A formal approach to adaptive software: Continuous assurance of non-functional requirements," *Form. Asp. Comput.*, vol. 24, no. 2, pp. 163–186, Mar. 2012. [Online]. Available: http://dx.doi.org/10.1007/s00165-011-0207-2

[30] A. Filieri, G. Tamburrelli, and C. Ghezzi, "Supporting self-adaptation via quantitative verification and sensitivity analysis at run time," *IEEE Transactions on Software Engineering*, vol. 42, no. 1, pp. 75–99, Jan 2016.

[31] R. Calinescu, Y. Rafiq, K. Johnson, and M. E. Bakir, "Adaptive model learning for continual verification of non-functional properties," in *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '14. New York, NY, USA: ACM, 2014, pp. 87–98.

[32] A. Filieri, L. Grunske, and A. Leva, "Lightweight adaptive filtering for efficient learning and updating of probabilistic models," in *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ser. ICSE '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 200–211. [Online]. Available: http://dl.acm.org/citation.cfm?id=2818754.2818781

[33] K. J. Astrom and B. Wittenmark, *Computer-controlled Systems: Theory and Design (2Nd Ed.)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1990.

[34] N. Walkinshaw and G. Fraser, "Uncertainty-driven black-box test data generation," in *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, March 2017, pp. 253–263.

[35] M. Zhang, S. Ali, T. Yue, R. Norgren, and O. Okariz, "Uncertainty-wise cyber-physical system test modeling," *Software & Systems Modeling*, Jul 2017. [Online]. Available: https://doi.org/10.1007/s10270-017-0609-6