Towards Inverse Uncertainty Quantification in Software Development (Short Paper)[*]

Matteo Camilli[1], Angelo Gargantini[2], Patrizia Scandurra[2], Carlo Bellettini[1]

[1] Dept. of Computer Science, Università degli Studi di Milano, Milan, Italy
{camilli,bellettini}@di.unimi.it

[2] Dept. of Management, Information and Production Engineering (DIGIP), Università degli Studi di Bergamo, Bergamo, Italy
{angelo.gargantini,patrizia.scandurra}@unibg.it

Abstract. With the purpose of delivering more robust systems, this paper revisits the problem of *Inverse Uncertainty Quantification* that is related to the discrepancy between the measured data at runtime (while the system executes) and the formal specification (i.e., a mathematical model) of the system under consideration, and the value calibration of unknown parameters in the model. We foster an approach to quantify and mitigate system uncertainty during the development cycle by combining Bayesian reasoning and online Model-based testing.

1 Introduction

The problem of *uncertainty quantification* is recently gaining attention in the software engineering community since it has a significant impact on the ability of a software system to satisfy its objectives [1,2]. Preliminary works towards this direction aim at establishing a common vocabulary and taxonomy of uncertainty from the perspective of a software system (see works [2, 3] to name a few).

Sources of uncertainty can occur either at requirements, design, or execution phase, and propagate throughout all phases [3]. 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: 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), experimental uncertainty (due to the inherent variability of experimental measurements), etc. From a different perspective uncertainty can be classified taking into account the *nature* [2]. The nature concerns the uncertainty due to the lack of knowledge (i.e., *epistemic*) or because of inherent randomness of the observed phenomenon (i.e., *aleatory*). Both kinds of uncertainties often come up in practice, during the development of real world applications.

[*] This is a short paper accepted in the new ideas and work-in-progress section of SEFM 2017.

Uncertainty quantification, in this context, has two major problems: *Forward Uncertainty Propagation* (FUQ) and *Inverse Uncertainty Quantification* (IUQ) [4]. The first problem focuses on studying the quantification of uncertainties in system output(s) propagated from uncertain inputs. This is useful in reliability engineering and to assess the complete probability distribution of the outputs in order to calculate and optimize the utility function. The latter one is essentially the inverse problem. Given some experimental measurements of a system and some simulation outputs from its mathematical model, inverse uncertainty quantification estimates the discrepancy between the measured data at runtime and the mathematical model (i.e., *bias correction*) and estimates the values of unknown parameters in the model if there are any (i.e., *parameter calibration*). FUQ is easier and more studied [5], while IUQ is recently drawing increasing attention in the engineering design community, since uncertainty quantification of a model and its inference from the true system response(s) are of great interest in designing robust systems.

In this paper, we revisit the IUQ problem in software development and propose an approach for quantifying system uncertainty [4] before the deployment of a release build. We depart from the unrealistic assumption that the outputs as well as specific properties are known for a given system before accounting for evidence during the actual system's execution. In fact, mathematical models are often imperfect and measured data from a running system is subject to noise. Therefore, it is of extreme importance to quantify and reduce the uncertainty to determine how likely certain outcomes are if some aspects of the system are not exactly known at design-time. To this purpose, we propose an exploration methodology to quantify and mitigate uncertainty during system development by combining Bayesian reasoning [6] and online *Model-based testing* (MBT) techniques [7]. The intuition behind our envisioned approach is to leverage the capability of online MBT to explore the state space in a controlled way, while the given system is up and running. At the same time, we gather information about the uncertain aspects of the system to perform inference activity.

As specification formalisms, we adopt Markov models, such as Discrete/-Continuous Time Markov Chains (D/CTMCs) [8], that are a widely accepted stochastic formalisms able to support modeling of randomly changing systems (or probabilistic systems), as well as quantitative verification of requirements using probabilistic temporal logic (e.g., PCTL, or CSL) model checking [9].

This paper is organized as follows. In Section 2, we introduce our approach to IUQ based on Bayesian reasoning applied through online MBT. We discuss related work in Section 3, and conclude and discuss challenges ahead in Section 4.

2 Overview of the Approach

Our approach to IUQ aims at estimating the discrepancy between the measured data y^e at runtime and the model response $y^m(\Theta)$ that depends on different uncertain parameters Θ of the Markov model m. Starting from the approximation $y^m(\Theta) \simeq y^e$, we perform a sequence of observations in order to infer a prob-

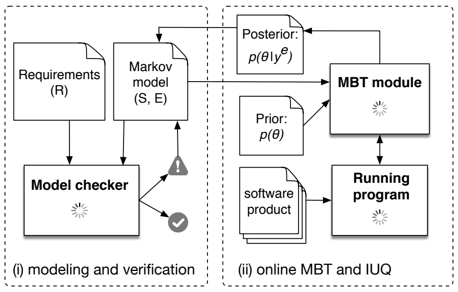

Fig. 1: IUQ methodology.

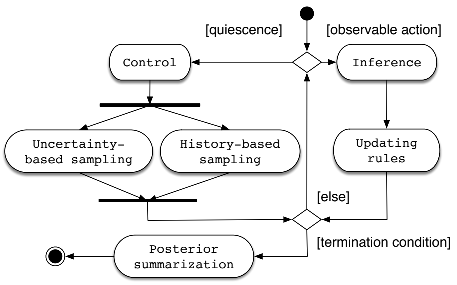

Fig. 2: Online MBT activity diagram.

ability distribution of Θ^* describing the best knowledge of the true parameter values, such that $y^m(\Theta^*) = y^e$.

The Bayesian methodology [6] provides a viable technique to incrementally update our *prior* uncertain knowledge (hypothesis) on a given phenomenon by observing its own behavior. The general formula is: $p(\Theta|y^e) \propto p(y^e|\Theta) \cdot p(\Theta)$, where $p(\Theta)$ represents the set of prior distributions for parameter set Θ and uses probability to express uncertainty about Θ before the data (i.e., the current evidence) is observed, $p(y^e|\Theta)$ is the likelihood function that expresses the compatibility of the evidence with the given hypothesis, and $p(\Theta|y^e)$ is the joint *posterior* distribution of the parameters after taking both the prior and the evidence into account. This formula basically links the degree of belief in the prior knowledge before and after accounting for evidence.

The inherent uncertainty of the system is explicitly modelled by means of *Prior* distributions of the parameters of interest of the Markov model. Observations to enable Bayesian reasoning are made at runtime during online MBT, where test strategies are created dynamically as testing goes on, taking advantage of the knowledge gained by exploring the model and by observing the evidence. A high-level overview of our methodology is shown in Figure 1. It relies on the iteration of two different phases, which are described below.

Design-time modeling and verification – This phase concerns the development of the mathematical model of the system under development. The model includes a formal representation of both the specification (S) and the environment (E). This separation is explicitly represented by disjointly partitioning the state transitions into *controllable* and *observable* ones. This choice is motivated by our problem domain of MBT. Thus, we follow the notation introduced in [7] to distinguish between full *controllable* behavior from the tester (i.e., the environment, such as user requests) and only *observable* behavior from the running software system (i.e., the specification, such as inter-components interaction). Markov models allow both the controllable and the observable behavior of system under development to be described in probabilistic terms from different perspectives, such as the architecture of the application, the response time of the components, or even the energy consumption (using for instance costs/rewards model extensions [10]).

Design-time model checking serves as a means to verify the desired requirements against the model of the system that contains our assumptions.

Online MBT and IUQ – This phase concerns the validation of the system and the inverse uncertainty quantification during testing activity. Fig. 2 shows the activity diagram of the main operations performed by our online MBT algorithm. Besides the *observation* until *termination* paradigm [7], usually applied in MBT and runtime verification, our approach relies on two additional steps: incremental `inference` and test scenarios `control` based on the design-time *uncertainty.*

– `Inference`: Given the natural conjugate priors for the uncertain parameters Θ of the Markov model, inference following the Bayesian approach reduces to the application of incremental `updating rules` [6, 8] for the posterior distributions based on the evidence that can be efficiently computed while the system is observed (i.e., foreach occurring *observable* action) at runtime. As an example, consider a video streaming web application, accessed from clients through HTTP requests via mobile application. Different components (e.g., data manager, cache, payment system, etc.) interact to satisfy users requests under different environmental conditions, such as workload (e.g., request rate) or user profiles (e.g., unregistered/registered users). Typical design-time uncertain parameters may include failure rates, and launch/response time of different video streaming servers that can be expressed for instance by means of independent *Dirichlet* and *Gamma* prior distributions describing the hypothesis on the rates and the probability matrix of a CTMC model, respectively [6, 8].

– `Control`: This step provides control over test scenarios by selecting actions during the test run based on the model *uncertainty.* In our application example, a wait condition (for user requests) can be controlled for instance by generating incoming requests at different rates, thus stressing the system in different workload conditions. In particular, if the running system is in a state of *quiescence* [11], the MBT algorithm chooses a legal *controllable* action such that the probability of this choice in the current state is governed by two weighted sampling methods.
The `uncertainty-based sampling` method is related to the likelihood of exploring uncertain regions of the model, choosing different controllable actions from the current state. It is grounded on the computation of the *maximum likelihood* trajectories [12] connecting the current state to states containing uncertain parameters of the Markov model.
The `history-based sampling` method takes advantage of the knowledge gained by exploring the model, thus allows the strategy to be configured based on the test run history. In particular *decrementing weight*s [7] can be adopted to call particular controllable actions a specific number of times in the test runs, in favor of unexplored regions.

Once termination has been reached, each uncertain parameter of interest can be described by summarizing the posterior distribution (i.e., the `summarization` activity) through the posterior *mode* and the *highest posterior density* (HPD) intervals [8]. Thus, the uncertainty can be numerically quantified by evaluating the discrepancy between the initial design-time parameter values and the mode values after accounting for evidence.

Estimated parameter values represent the basis of new verification phases and the prior knowledge for future evolutions of the software system.

3 Related work

In the community of self-adaptive systems, there have been several efforts focusing on studying the FUQ problem and on dealing with changing requirements and unpredictable environment (see [2,3,13,14], to name a few), some by employing Markov models. Our approach revisits the IUQ problem and focus mainly on the system uncertainty [4] in software development.

An interesting effort has been shown in [15]. It focuses primarily on design-time verification aspects to assure quality-of-service (QoS) properties of systems that exhibit stochastic behavior. The presented technique and toolchain aim at establishing confidence intervals for the QoS properties of a software system modeled as a Markov chain with uncertain transition probabilities.

Concerning testing techniques, a promising *Active Learning* query strategy to black-box test generation has been proposed in [16]. It aims at overcoming the problem of intractability in MBT and generating test cases which the inferred model is “least certain” about. The usage of machine learning algorithms and Bayesian reasoning represents an attractive approach to achieve reliable and efficient software testing and program analysis [17]. Despite the inherent potential of these methods, their employment in software testing and program analysis, to tackle the IUQ problem, is still in its early stages.

4 Conclusion

We proposed an approach to quantify and mitigate system uncertainty during system development life cycle, by combining Bayesian reasoning [6,18] and online *Model-based testing* (MBT) [7]. The key idea is to explicitly model the inherent uncertainty and provide a means to stress and observe the software product in order to quantify the design-time uncertainty before the deployment of a release build. In order to validate our current prototypal implementation, we are going to conduct several experiments with case studies of different size and complexity. Our experience in this context has been very positive. A great advantage of the underlying probabilistic representation and our incremental update scheme of the posterior knowledge is the robustness to unreliable/spurious observations (difficult to achieve with non-probabilistic techniques).

There are also challenges to be faced. A very critical issue, for example, is that stochastic techniques and Bayesian reasoning are computationally expensive, thus often unsuitable for use at run-time. However, in our approach, expensive probabilistic model checking is used only at design-time, while very efficient incremental inference steps are carried out at run-time during testing activity.

References

1. D. Garlan, “Software engineering in an uncertain world,” in *Proc. of the FSE/SDP Workshop on Future of Software Engineering Research*, 2010, pp. 125–128.
2. N. Esfahani and S. Malek, *Uncertainty in Self-Adaptive Software Systems.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 214–238.
3. A. J. Ramirez, A. C. Jensen, and B. H. C. Cheng, “A taxonomy of uncertainty for dynamically adaptive systems,” in *Proc. of the 7th Int. Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2012, pp. 99–108.
4. 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, 2012.
5. 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.
6. J. Berger, *Statistical Decision Theory and Bayesian Analysis*, ser. Springer Series in Statistics. Springer, 1985.
7. 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.
8. D. Insua, F. Ruggeri, and M. Wiper, *Bayesian Analysis of Stochastic Process Models*, ser. Wiley Series in Probability and Statistics. Wiley, 2012.
9. M. Kwiatkowska, G. Norman, and D. Parker, “PRISM 4.0: Verification of probabilistic real-time systems,” in *Proc. 23rd Int. Conference on Computer Aided Verification (CAV’11)*, ser. LNCS, vol. 6806. Springer, 2011, pp. 585–591.
10. M. Kwiatkowska, G. Norman, and A. Pacheco, “Model checking expected time and expected reward formulae with random time bounds,” *Computers & Mathematics with Applications*, vol. 51, no. 2, pp. 305–316, 2006.
11. J. Tretmans and A. Belinfante, “Automatic testing with formal methods,” in *7th European Int. Conf. on Software Testing, Analysis & Review*, 1999, pp. 8–12.
12. T. J. Perkins, “Maximum likelihood trajectories for continuous-time markov chains,” in *Proceedings of the 22Nd Int. Conf. on Neural Information Processing Systems*, 2009, pp. 1437–1445.
13. 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 Int. Conference on Performance Engineering*, 2014, pp. 3–14.
14. I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli, “Model evolution by run-time parameter adaptation,” in *2009 IEEE 31st International Conference on Software Engineering*, May 2009, pp. 111–121.
15. R. Calinescu, C. Ghezzi, K. Johnson, M. Pezzè, Y. Rafiq, and G. Tamburrelli, “Formal verification with confidence intervals to establish quality of service properties of software systems,” *IEEE Trans. Reliability*, vol. 65, no. 1, pp. 107–125, 2016.
16. N. Walkinshaw and G. Fraser, “Uncertainty-driven black-box test data generation,” in *IEEE Int. Conf. on Software Testing, Verification and Validation*, 2017.
17. A. S. Namin and M. Sridharan, “Bayesian reasoning for software testing,” in *Proc. of the FSE/SDP Workshop on Future of Soft. Eng. Research*, 2010, pp. 349–354.
18. J. Bernardo and A. Smith, *Bayesian Theory*, ser. Wiley Series in Probability and Statistics. John Wiley & Sons Canada, Limited, 2006.