HYPpOTesT: Hypothesis Testing Toolkit for Uncertain Service-based Web Applications

Matteo Camilli, Angelo Gargantini, Rosario Madaudo and Patrizia Scandurra

December 25, 2019

HYPpOTesT: Hypothesis Testing Toolkit for Uncertain Service-based Web Applications

Matteo Camilli[⋆], Angelo Gargantini[†], Rosario Madaudo[‡], and Patrizia Scandurra[†]

[⋆] Faculty of Computer Science, Free University of Bozen-Bolzano, Italy
matteo.camilli@unibz.it
[†] Dept. of Management, Information and Production Engineering (DIGIP), Università degli Studi di Bergamo, Italy
{angelo.gargantini,patrizia.scandurra}@unibg.it
[‡] Altran Italia SPA, Milan, Italy
rosario.madaudo2@altran.it

Abstract. This paper introduces a model-based testing framework and associated toolkit, so called HYPpOTesT, for uncertain service-based web applications specified as probabilistic systems with non-determinism. The framework connects input/output conformance theory with hypothesis testing in order to assess if the behavior of the application under test corresponds to its probabilistic formal specification. The core component is a (on-the-fly) model-based testing algorithm able to automatically generate, execute and evaluate test cases from a Markov Decision Process specification. The testing activity feeds a Bayesian inference process that quantifies and mitigates the system uncertainty by calibrating probability values in the initial specification. This paper illustrates the structure, features, and usage of HYPpOTesT using the U-Store exemplar, i.e., a web-based e-commerce application that exhibits uncertain behavior.

Keywords: Model-based Testing · Probabilistic systems · Service-based Web applications · Bayesian Inference · Uncertainty Quantification.

1 Introduction

Modern software systems operate in a complex ecosystem of protocols, libraries, services, and execution platforms that change over time in response to: new technologies; repairing activities (due to faults and vulnerabilities); varying resources/services availability; and reconfiguration of the environment. Predictability is very hard to achieve since modern software-intensive systems are often situated in complex ecosystems that can be hard or even impossible to fully understand and specify at design-time. Namely, these systems are often exposed to multiple sources of uncertainty that can arise from an ambiguous specification not completely known before the system is running. Common examples of applications affected by uncertain and probabilistic behavior are control policies in robotics, speech recognition, security protocols, and service-based web applications (e.g., e-commerce, e-health, online banking, etc.). In this latter case, highly dynamic and changing ecosystems influence a workflow of interacting distributed components (e.g., web-services or microservices) owned by multiple third-party providers with different Quality of Service (QoS) attributes (e.g, reliability, performance, cost, etc.). Testing is the most common validation technique, seen as it generally represents a lightweight and vital process to establish confidence on the developed software systems. Nevertheless, there is little work in the scientific community that focuses on executable testing frameworks for uncertain systems, with notable exceptions in the context of cyber physical systems [11, 12].

As part of our ongoing research activity on testing under uncertainty [2, 3], this paper introduces HYPpOTesT: a model-based HYPOthesis Testing Toolkit for service-based web applications that considers uncertainty as a first-class concern. Namely, we focus on statistical *hypothesis testing* [5] of uncertain QoS parameters of the System Under Test (SUT), modeled by a Markov Decision Processes (MDP) [7]. MDPs represent a widely adopted formalism for modeling systems exhibiting both probabilistic and nondeterministic behavior. As described in [4], hypothesis testing (differently from functional testing) represents a fundamental activity to assess whether the frequencies observed during model-based testing (MBT) processes correspond to the probabilities specified in the model. Thus, HYPpOTesT has been tailored to deal with the *uncertainty quantification* problem [3] by means of hypothesis testing while executing a model-based exploration of the SUT. The MDP specification, along with assumptions on the uncertain QoS parameters, guides the automatic generation of the test cases so that the probability to stress the uncertain components of the application is maximized. Testing feeds a *Bayesian inference* [5] process that calibrates the uncertain parameters depending on the observed behavior. Bayesian inference is used to compute the Posterior density function associated with specific uncertain parameters θ of the MDP model. As described in [5], Bayesian inference represents an effective technique used to update belief about θ. A Prior density function (or simply Prior) is the probability distribution that would express one's beliefs about θ parameters before some evidence (i.e., experimental data) is taken into account.

This paper focuses on engineering aspects of HYPpOTesT, such as design and implementation concerns. To sum up, our toolkit supports: (*i*) modeling of a service-based web application (SUT) in terms of a MDP using a simple domain-specific language; (*ii*) explicit elicitation of the uncertain QoS parameters using Prior probability distributions; (*iii*) automatic generation, execution, and evaluation of the test cases using an uncertainty-aware (on-the-fly) model-based testing algorithm. Throughout the paper, we adopt an uncertain web-based e-commerce application, U-Store (Uncertain Store), as running example to illustrate the main feature of the toolkit and as exemplar for a preliminary validation of our testing approach.

Related Work. In [1] MDPs are used to model systems exhibiting stochastic failures. The proposed approach aims at finding input-selection (i.e., testing) strategies which maximizes the probability of hitting failures. The approach introduced in [10] is based on Machine Learning and it aims at inferring a behav-

Table 1. Services composing the U-Store web-based application.

Service	Description
`frontend`	Exposes an HTTP server to serve the website. Generates session IDs for all users.
`user`	Provide Customer login, sign up, as well as user's information retrieval.
`cart`	Stores selected items in the user's shipping cart (persistent memory) and retrieves it.
`catalog`	Provides the list of products from a SQL database, the ability to search products, and get individual products.
`shipping`	Gives shipping cost estimates based on the shopping cart and the given address (mock component).
`payment`	Charges the given credit card info (mock component) with the given amount and returns a transaction ID.

ioral model of the SUT to select those tests which the inferred model is "least certain" about. Results suggest that such a test case generation outperforms conventional random testing. In [12] test case generation strategies based on the uncertainty theory and multi-objective search are proposed in the context of cyber-physical systems. Results in this work showed that this test strategy increases the likelihood to observe more uncertainties due to unknown behaviors of the physical environments. In [11] a testing method that takes into account uncertainty in timing properties of embedded software systems is proposed. This method improves the fault detection effectiveness of tests suites derived from timed automata compared to traditional testing approaches. Summarizing, there are few and recently defined approaches that deal with testing driven by uncertainty awareness. Notable examples has been briefly described. The topic definitely needs further investigation in the area of service-based applications, where QoS attributes are influenced by highly dynamic and uncertain ecosystems.

This paper is organized as follows. Sect. 2 introduces the running example; Sect. 3 describes our testing toolkit; Sect. 4 reports some experimental results of a preliminary validation of our toolkit; and Sect. 5 concludes the paper.

2 Running Example: the U-Store Web Application

U-Store[1] consists of a number of services that implement specialized pieces of functionalities and interact with each other using HTTP resource APIs. Table 1 lists the services of the U-Store and provides a brief description of them. From the user perspective, the application behavior can be viewed as a number of *functional statuses* (or states), each one of them with a number of feasible *input*s that cause services to execute specific tasks. Services tasks generate *output*s and allow the current state to be changed accordingly.

In this context, both functional and non-functional quality attributes of the web application depend on parameters (e.g., performance, bandwidth, available memory, etc.) typically subject to different sources of uncertainty (e.g., jobs arrival, fault tolerance, scalability, etc.) [6]. As an example, suppose the user

[1] Sources and testing results are publicly available at https://github.com/SELab-unimi/ustore-exemplar.

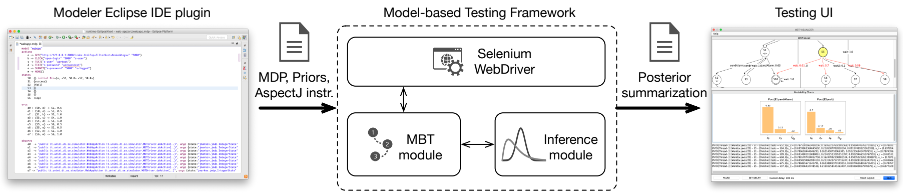
Modeler Eclipse IDE plugin
MDP, Priors, AspectJ instr.
Model-based Testing Framework
Selenium WebDriver
MBT module
Inference module
Posterior summarization
Testing UI

Fig. 1. Main components of the HYPPOTESTtoolkit.

navigates the U-STORE towards the *Checkout* web page. After selecting the payment method, the user submits the *buy* request. At this stage, the U-STORE asks the external `payment` service to execute the proper task. The outcome of this operation is inherently influenced by several sources of uncertainty (as those mentioned above), and from the user perspective uncertainty reflects common types of failure or undesired behavior upon the *buy* request, such as unexpected errors and/or high latency.

3 The Hypothesis Testing Toolkit

HYPPOTEST (see Fig. 1) is tailored to perform hypothesis testing of uncertain service-based web applications by combining an uncertainty-aware MBT and Bayesian inference. A description of the three major components follows.

Modeler. The modeler is an ECLIPSE IDE plugin[2] that allows the MDP specification to be created using a textual Domain Specific Language (DSL). As described in [7], a MDP model is composed of finite sets of states, transitions, and actions. Transitions between states occur by taking a nondeterministic choice among the available actions from the current state and then a stochastic choice of the successor state, according to a partial probabilistic transition function. Fig. 2 reports an extract of the U-STORE MDP design-time model using our DSL. The keywords reflect the structural elements of the model: actions (line 2), states (line 6), and arcs (line 14). Fig. 3 contains a visual representation of this MDP extract. It is worth noting that upon the submit action from state S_6 (readyToPay) we can have multiple responses from the system. Each one of them is associated with a different probability value reflecting our assumption about the behavior of the `payment` service, typically based on past experience or previous studies. To express uncertainty on the assumptions, the DSL allows the Prior probability density functions to be specified (e.g., line 10). In particular, modelers use *Dirichlet* as conjugate Priors for the uncertain transition probabilities of the MDP model. In fact, as described in [5], the Dirichlet distribution $Dir(\alpha_i)$ is the natural conjugate prior of the categorical distribution, with $\alpha_i = (\alpha_1, ..., \alpha_n)$

[2] Publicly available at https://github.com/SELab-unimi/mdp-generator/tree/web-app. The repository contains sources and the complete specification of the U-Store.

```
 1  model "UStore"
 2  actions
 3   select -> click("s-credit-card" "1000" "s-pay-button")
 4   submit -> submit_form("s-pay" "5000" "s-pay-result")
 5   ...
 6  states
 7   S0 { } initial
 8   ...
 9   S5 {checkoutPage}
10   S6 {readyToPay} Dir(submit, <S7, 95.0> <S8, 3.0> <S9, 2.0>)
11   S7 {success}
12   S8 {error}
13   S9 {failure}
14  arcs
15   ...
16   a5 : (S5, select) -> S6, 1.0
17   a6 : (S6, submit) -> S7, 0.95
18   a7 : (S6, submit) -> S8, 0.03
19   a8 : (S6, submit) -> S9, 0.02
20  observe
21   ...
22   a6 -> "result.getUIElement().findElement(By.id(\"s-success\")).isDisplayed()"
23   a7 -> "result.getUIElement().findElement(By.id(\"s-error\")).isDisplayed()"
24   a8 -> "result.timeout()"
```

Fig. 2. U-Store MDP extract in our DSL.

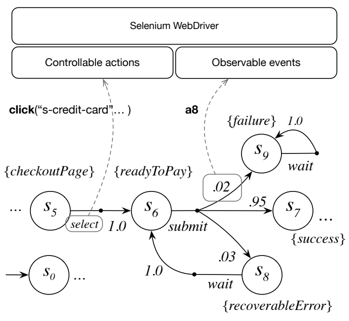

Fig. 3. Visual representation of the U-Store MDP extract and of mapping to SUT components.

vector of concentration parameters. Priors are used to express uncertainty on model parameters describing QoS attributes, such as reliability of the services or the communication channels, response time, and cost in terms of resources usage or energy consumption.

To make our framework able to carry out model-based generation and execution of tests, the structural elements in the model are bound to components in the SUT as informally sketched in Fig. 3. Such a binding is defined by the modeler at design-time. MDP actions are mapped to *controllable action*s while arcs are mapped to *observable event*s. Controllable actions are user inputs supplied to the application using the available web UI. A wide range of inputs typically seen in web applications are supported by our DSL, such as click on different UI elements (e.g., link, button, checkbox, etc.), filling in text fields, submit forms, navigating back and forth, and more. As an example, the submit_form controllable action (line 4) allows a form to be submitted. Arguments specify the form *id*, a *timeout*, and the *id* of a specific UI element which contains the result of the executed task. Each arc in the model is mapped to an observable event (e.g., line 22), that is an arbitrary Java boolean expression, where we typically make assertions on the resulting UI element (i.e., a `WebElement` object of the Selenium [8] library package `org.openqa.selenium`). After the execution of a controllable action, HYPpOTesT waits until one of the suitable observable events happens and performs the selected transition. These operations are executed by an automatically generated *test harness* (AspectJ instrumentation). The MBT module generates test cases using the MDP specification and makes them executable upon the SUT though the test harness.

Model-based Testing Framework. The main components of the testing framework[3] are: the *MBT module* responsible for test cases generation; the *Selenium*

[3] Sources and instructions are publicly available at https://github.com/SELab-unimi/mbt-module/tree/web-app.

WebDriver responsible for test cases execution; and the *Inference module* responsible for hypothesis testing.

The MBT module dynamically generates test cases from the MDP specification according to an *uncertainty*-aware test case generation strategy. Essentially, this strategy solves a *dynamic decision problem* [7] to compute the *best exploration policy* π^* that returns, for each state s, the actions that maximize the probability to reach the uncertain θ parameters in the model. More technical details on this strategy can be found in [2]. Thus, the testing process stochastically samples the state space by choosing those inputs that allow the uncertain components of the SUT to be stressed out. To this end, the *test harness* provides a high-level view of the SUT behavior matching the abstraction level of the MDP specification. Technically, it allows the actions selected by the exploration policy π^* to be translated into valid inputs for the SUT by means of the *Selenium WebDriver* that interacts directly with the web UI. At the same time, observable events provide a serialized view on the SUT behavior to keep track of the execution trace and extract meaningful data to perform hypothesis testing. From a theoretical perspective, the MBT module uses the test harness to conduct a input/output *conformance game* [2, 9] between the model and the SUT. During the conformance game, hypothesis testing is carried out by the Inference module that incrementally updates beliefs on θ parameters by using the Bayesian inference formulation: $Posterior \propto Likelihood \cdot Prior$. In our context, the Prior and Posterior are conjugate distributions and the Posterior can be obtained by applying a very efficient updating rule [2, 5]. In fact, the Posterior is distributed as $Dir(\alpha')$, where $\alpha' = \alpha + (c_1, ..., c_n)$ with c_i number of observations in category i. At termination, the MBT module summarizes the Posteriors (by computing the mean values) and calibrates the θ parameters.

Two termination conditions are currently supported by our testing framework: a traditional condition based on the *number of executed tests*; and termination based on the convergence of the *Bayes factor* [5]. This latter condition, in particular, is a model selection method that allows the testing activity to be terminated when the θ parameters do not substantially change during the inference process. So, by using this latter method the Inference module decides when inferred θ parameters are strongly supported by the data under consideration.

Testing UI. The UI allows information about hypothesis testing to be visualized for human consumption. Three different canvas in the main window show: the MBT model, the Posterior density functions, and a log produced by the MBT module. The MDP model 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 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 canvas shows textual information generated by the MBT module for each uncertain parameter: the number of executed tests, the summarization of the Posterior density functions, and the Bayes factor.

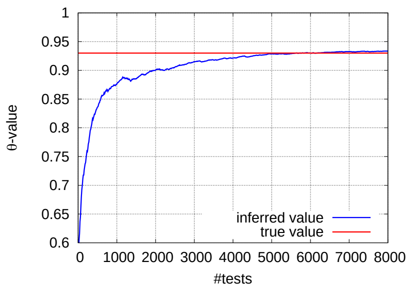

Fig. 4. Inference of a θ parameter.

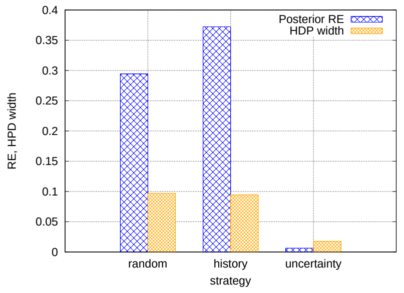

Fig. 5. Inference effectivenes.

4 Evaluation

We are evaluating our testing framework by conducting a large testing campaign of the U-Store application. Here we briefly discuss some significant results and we refer the reader to our implementation for the replicability of the presented data. To measure the effectiveness of HYPpOTesT, we artificially induced abnormal conditions due to sources of uncertainty. Namely, the services composing the U-Store application have been configured to simulate service degradation by means of failure rates and random delays. As an example, we forced the uncertain `payment` component to have specific failure/error rates and we executed our testing framework starting from wrong hypothesis (i.e., using an informative Prior having a relative error of 1.5). Fig. 4 shows how the uncertain success rate associated with the `payment` service varies during hypothesis testing of the U-Store. The uncertainty-aware strategy allows the probability to test the `payment` component to be maximized during model-based exploration. As long as evidence is collected, the Posterior knowledge is incrementally updated. The termination condition based on the Bayes factor allows the uncertain θ parameter to be inferred with high precision (i.e., the order of magnitude of the Posterior relative error is 10^{-2}) after executing $\sim 8k$ tests.

We also compared the effectiveness of our uncertainty-aware test case generation strategy with two traditional model-based exploration strategies: a completely random walk approach; and a history-based exploration approach. Fig. 5 shows the accuracy in terms of Posterior relative error and Posterior Highest Probability Density (HPD) region width, very often used as a measure of the confidence gained after the inference activity (i.e., the smaller the region width, the higher the accuracy). This comparative evaluation assumes equal effort (i.e., $5k$ tests) spent using different strategies. For each strategy, we started the hypothesis testing activity using a Prior with 0.5 relative error and 0.1 HPD region width. In our running example, the uncertainty-aware strategy used by HYPpOTesT allows inference to be always more precise. On average, we measured a decreased Posterior relative error by a factor of ~ 50.

5 Conclusion

In this paper we presented HYPpOTesT, a model-based testing toolkit for uncertain web service applications modeled in terms of MDPs. HYPpOTesT adopts an online MBT technique that combines test case generation guided by uncertainty-aware strategies and Bayesian inference. Namely, we focused on statistical hypothesis testing of uncertain QoS parameters of the SUT. The U-Store application was used throughout the paper to illustrate the features of the toolkit and as validation benchmark.

As future work, we plan to study different fine grained uncertainty-aware testing methods in order to assess whether delivered confidence out from testing may be better if looking at specific and uncertain model-based properties of interest. We also plan to enhance the toolchain that supports the proposed approach with the ability to perform sensitivity analysis can be apportioned to different experimental designs (e.g., traffic condition, frequency of requests, workload of services).

References

1. Aichernig, B.K., Tappler, M.: Probabilistic black-box reachability checking. In: Lahiri, S., Reger, G. (eds.) Runtime Verification. pp. 50–67. Springer International Publishing, Cham (2017)
2. Camilli, M., Bellettini, C., Gargantini, A., Scandurra, P.: Online model-based testing under uncertainty. In: 2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE). pp. 36–46 (Oct 2018)
3. Camilli, M., Gargantini, A., Scandurra, P., Bellettini, C.: Towards inverse uncertainty quantification in software development (short paper). In: Cimatti, A., Sirjani, M. (eds.) Software Engineering and Formal Methods. pp. 375–381. Springer International Publishing, Cham (2017)
4. Gerhold, M., Stoelinga, M.: Model-based testing of probabilistic systems. Formal Aspects of Computing **30**(1), 77–106 (Jan 2018)
5. Insua, D., Ruggeri, F., Wiper, M.: Bayesian Analysis of Stochastic Process Models. Wiley Series in Probability and Statistics, Wiley (2012)
6. Perez-Palacin, D., Mirandola, R.: Uncertainties in the modeling of self-adaptive systems: a taxonomy and an example of availability evaluation. In: International Conference on Performance Engineering. pp. 3–14 (2014)
7. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley & Sons, Inc., New York, NY, USA (1994)
8. Selenium HQ: WebDriver. https://www.seleniumhq.org/ (2019), online; accessed June 2019
9. Veanes, M., Campbell, C., Schulte, W., Tillmann, N.: Online testing with model programs. SIGSOFT Softw. Eng. Notes **30**(5), 273–282 (2005)
10. Walkinshaw, N., Fraser, G.: Uncertainty-driven black-box test data generation. In: International Conference on Software Testing, Verification and Validation. pp. 253–263 (2017)
11. Wang, C., Pastore, F., Briand, L.: Oracles for testing software timeliness with uncertainty. ACM Trans. Softw. Eng. Methodol. **28**(1), 1:1–1:30 (Nov 2018)
12. Zhang, M., Ali, S., Yue, T.: Uncertainty-wise test case generation and minimization for cyber-physical systems. Journal of Systems and Software **153**, 1 – 21 (2019)