Towards testing self-adaptive software for cyber-physical systems

Claudio Mandrioli, Martina Maggio
Department of Automatic Control, Lund University

I. Introduction

Many cyber-physical systems change their behaviour depending on environmental data and internal states. This is the case of *control systems*, that compute a control signal that depends on input values like a desired position, measured values like the current position, and internal states like the previous control action. This is also the case of systems embedding *machine learning algorithms*, that receive new samples and incorporate what they learnt using these new samples into a policy that determines how to behave in new conditions. All these systems are *adaptive*, in that their behaviour changes over time in a prescribed – but *a priori* unpredictable – way. This paper discusses *testing* and *comparing* systems that incorporate some adaptivity.

Testing systems whose behaviour varies over time is difficult [4], [7]. Think of a machine learning algorithm: how many and which samples should we give to the system before we can consider its behaviour *testable*? And what is the *correct* outcome? Of course we can apply unit testing to each function in the code, check for coverage, select a few cases in which the ideal behaviour of the code is known. But this does not give us any guarantee that the code is behaving *correctly* for the task it has to complete in the physical environment.

Comparing systems whose behaviour varies over time is difficult. Consider a physical process and three different controllers implemented for that process, C_1, C_2, and C_3. In all cases, we execute the system asking for the single output to reach a setpoint of one unit. A simulation of the system's behaviour is shown in Figure 1. Both C_1 and C_2 are able to drive the system output to the desired value. The first controller, C_1, is slower in achieving convergence, but never exceeds the setpoint. The second controller, C_2, is faster in reaching a value close to the objective, but overshoots. The simulation with the third controller, C_3, converges very fast, but the controller does not succeed in driving the output to the requested value. While C_3 failed the test, depending on what we choose to be our desired behaviour, either C_1 or C_2 behaves better. Furthermore, if we add some disturbance to the system, we could see a difference in how the two controllers handle it. Depending on the entity of the disturbance and on where we introduce it (e.g., a load disturbance or a measurement error) we could prefer one controller to the other.

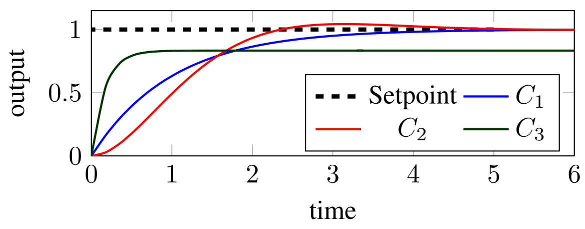

Fig. 1. Simulation of the execution of different controllers.

Moreover, learning algorithms in general present an unprecedented challenge, when it comes to guaranteeing specific properties for the environments they operate in. In fact, the learning process is used precisely to avoid explicitly considering all the possible instances in which the environment can be found. The exploration and the analysis of a significant portion of the potential environment space is not only infeasible but also goes against the precise objective for which these algorithms are employed.

We advocate that a *formal* and *rigorous* methodology is needed to test systems with adaptivity. This methodology should be used in conjunction with other forms of testing (e.g., unit testing) to provide guarantees on the cyber-physical system behaviour.

II. Proposal

When learning is involved, it is impossible to provide any deterministic guarantees, since the function to be learnt may not have been explored [1]. In such cases, drawing any general conclusion is impossible (and undesirable), unless probabilistic guarantees are targeted. We are convinced that this is true also for adaptive software and a paradigm shift is necessary for its testing: guarantees deriving from the tests' execution should be provided in the probabilistic space rather than in the deterministic one.

In the probabilistic space, we plan to investigate three alternatives methods to analyze testing data and provide guarantees:

- Monte Carlo experiments [8],
- Extreme Value Theory [6],
- Scenario Theory [5].

In physics, the Monte Carlo method [8] is used to simulate systems with many degrees of freedoms that interfere with one another and to model phenomena with significant uncertainty. In principle, it is possible to use Monte Carlo methods to solve any problem that has a probabilistic interpretation. Generally speaking, Monte Carlo methods are based on the definition of the domain of input variables and the generation of possible realizations drawn from the corresponding probability distributions. Then, one can compute values (in a deterministic way) based on the drawn samples. The larger the sample set size is, the more likely it is to cover the space of all possible realizations. Monte Carlo methods are very easy to use and implement. On the downside, they can be heavy from a computational perspective. Furthermore, their aim is to estimate true values of quantities (like failure probabilities) without guaranteeing an exhaustive exploration of the uncertainty space.

In statistics, the Extreme Value Theory [6] answers the question of estimating worst case circumstances based on observations and data. From samples of a random variable, it estimates what are the most extreme events that we can expect from a given process. This is formulated in terms of estimating the maximum of a stochastic process. When paired with the execution of many tests on the combination of software and environment, the Extreme Value Theory could provide us with probabilistic bounds for the worst case scenario.

In control, the *scenario theory* [5] is used to design controllers that are robust in the presence of uncertainty. The problem of testing the behaviour of adaptive software (and cyber-physical systems) can be reformulated as a problem of handling uncertainty. While in robust control the controller should be designed to withstand the widest possible class of disturbances, the adaptive software should be tested in many different conditions, the aim being to provide guarantees that hold also in the more general sense (i.e., also on new data) with a certain probability. We propose to draw a parallel between the task of robust control and *robust* testing and re-interpret the scenario theory to derive strategies to formally and rigorously test self-adaptive software. The scenario theory could not only provide us probabilistic bounds on worst case events, but also give us a measure of the confidence we have in the analysis, based on the uncertainty characterization.

III. Preliminary Results

We conducted some preliminary exploration, using the Tele Assistance System (TAS) case study [3], [13] as a self-adaptive software system. TAS is a medical assistance system for elderly people. Measurements of vital signs are taken from the person, and used to query a service that analyzes the data. This service can be chosen among many, each having different characteristics (for example different probability of failing, different costs, and so on). An adaptation layer uses adaptation strategies to determine which service should be queried. If the results of the analysis indicates problems, two different countermeasures can be taken: drugs can be used, or an ambulance can be alerted. Also in this case, the computation can be offloaded to different services, with different execution characteristics.

The adaptation strategy in TAS is used to provide guarantees on the response times and failures of requests in presence of changes in the environment. For example, network connectivity affects service invocation, causing the failure of one service. This can result in cascaded problems. One research problem here is how to design an adaptation strategy that handles different objectives.

A few alternatives strategies have been proposed [12], [2] or can be adapted [9]. While there have been attempts to compare these strategies – like [10] and [11] – the resulting comparisons are not based on formal testing of the software and do not provide guarantees that cases that are not covered in the comparison could in fact be very common. The results are therefore very *ad hoc* and only valid for the specific workload that has been tested.

In our talk, we will present our preliminary results and show the probabilistic guarantees that it is possible to obtain using Monte Carlo simulations, Extreme Value Theory, and Scenario Theory.

Acknowledgements: This work was partially supported by the Wallenberg Artificial Intelligence, Autonomous Systems and Software Program (WASP) funded by Knut and Alice Wallenberg Foundation, and by the Nordforsk Nordic Hub on Industrial IoT (HI2OT).

References

[1] Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin. *Learning From Data*. AMLBook, 2012.

[2] K. Angelopoulos, A. V. Papadopoulos, V. E. S. Souza, and J. Mylopoulos. Model predictive control for software systems with cobra. In *Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 35–46, 2016.

[3] L. Baresi, D. Bianculli, C. Ghezzi, S. Guinea, and P. Spoletini. Validation of web service compositions. *IET Software*, 1(6):219–232, December 2007.

[4] L. Briand, S. Nejati, M. Sabetzadeh, and D. Bianculli. Testing the untestable: Model testing of complex software-intensive systems. In *International Conference on Software Engineering Companion*, ICSE Companion, pages 789–792, 2016.

[5] G. C. Calafiore and M. C. Campi. The scenario approach to robust control design. *IEEE Transactions on Automatic Control*, 51(5):742–753, 2006.

[6] L. de Haan and A. Ferreira. *Extreme Value Theory: An Introduction*. Springer, 2010.

[7] C. A. González, M. Varmazyar, S. Nejati, L. C. Briand, and Y. Isasi. Enabling model testing of cyber-physical systems. In *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, MODELS '18, pages 176–186. ACM, 2018.

[8] D. P. Kroese, T. Brereton, T. Taimre, and Z. I. Botev. Why the monte carlo method is so important today. *Wiley Interdisciplinary Reviews: Computational Statistics*, 6(6):386–392, 2014.

[9] M. Maggio, A. V. Papadopoulos, A. Filieri, and H. Hoffmann. Automated control of multiple software goals using multiple actuators. In *Foundations of Software Engineering*, pages 373–384, 2017.

[10] G. A. Moreno, A. V. Papadopoulos, K. Angelopoulos, J. Cámara, and B. R. Schmerl. Comparing model-based predictive approaches to self-adaptation: Cobra and PLA. In *Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 42–53, 2017.

[11] S. Shevtsov, M. U. Iftikhar, and D. Weyns. Simca vs activforms: comparing control- and architecture-based adaptation on the TAS exemplar. In *International Workshop on Control Theory for Software Engineering*, pages 1–8, 2015.

[12] S. Shevtsov and D. Weyns. Keep it SIMPLEX: satisfying multiple goals with guarantees in control-based self-adaptive systems. In *International Symposium on Foundations of Software Engineering*, pages 229–241, 2016.

[13] D. Weyns and R. Calinescu. Tele assistance: A self-adaptive service-based system examplar. In *Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS, pages 88–92. IEEE Press, 2015.