Automatic testing of OCE, a human-centered reinforcement learning system for automated software composition

Maxence Demougeot, Kevin Delcourt, Jean-Paul Arcangeli, Sylvie Trouilhet and Françoise Adreit

Institut de Recherche en Informatique de Toulouse, Université de Toulouse, UT3, UT2J, France

Abstract

More and more applications rely on Machine Learning (ML) techniques, e.g., to automate software engineering. Like other applications, they need to be tested and validated. Testing ML-based software differs from testing software which do not rely on AI and ML: non-determinism, lack of oracle, high dependence on training and evaluation data are hard points. The problem is even more complicated when humans are involved in the process. In this paper, we analyze the problem of evaluating OCE, a human-centered intelligent system based on reinforcement learning that automatically builds user-tailored software. We present a test environment composed of two tools which are based on the notions of "scenario" and "ideal assembly": OCE Scenario Maker to edit scenarios and OCE Scenario Runner to automate and repeat their execution.

Keywords

Test automation, testing tools, automated software composition, reinforcement learning, human in-the-loop

1. Introduction

More and more applications rely on Machine Learning (ML) techniques, namely to automate software engineering. As part of our project on *Opportunistic Software Composition*, we are designing an intelligent solution based on reinforcement learning (RL) to automatically build applications in ambient contexts, and we are developing several working prototypes of an *Opportunistic Composition Engine* (OCE). With OCE, the user is put in the loop and provides feedback on the built applications.

For ML-based solutions, as with any software, development teams need to address validation issues. Testing is a potential solution. However, testing human-centered ML-based systems is quite different from testing traditional software, i.e., those which do not rely on ML techniques.

For our part, we need to assess the prototype versions of OCE in different use cases. Beyond the common problems posed by the evaluation of ML-based systems, our issues lie in the dynamics of the learning environment, including the human user. To answer, we have developed a tooling that supports the definition of test scenarios with simulated users, their execution and the evaluation of the results, which use is illustrated in a video[1].

The purpose of this paper is to present our work on testing and evaluating OCE, more specifically its learning mechanism. It describes the principles of our solution and the tooling we have developed.

Sec. 2 summarizes the principles of Opportunistic Software Composition and our needs for testing. Sec. 3 analyzes the main issues of testing ML-based systems in particular with humans "in the loop", then focuses on OCE. Sec. 4 briefly presents several tools that allow testers to carry out experiments on ML-based solutions, then describes the principles of our solution, the prototype tools we have developed and integrated with OCE, and a demonstration. Sec. 5 concludes and discusses some open issues.

2. Background

2.1. Opportunistic Software Composition

Today's users live in ambient environments invaded by connected objects. These environments are open, complex and dynamic: at any time, objects may appear and others may disappear. Besides, the user needs may change depending on the situation. One of the issues is how to manage the variability of such unpredictable environments, and propose relevant services to the user [1].

To tackle this issue, Opportunistic Software Composition is a human-centered approach based on reinforcement learning (RL) [2] that aims at automatically build applications that are tailored to the user and the ambient context. It plans on-the-fly assemblies of software components that are available in the ambient environment. Like objects, software components [3] expose the services they provide through an interface; in addition, they expose at the same level the services they require

1st International Workshop on Intelligent Software Engineering, APSEC 2022, 6 December, Gyeongsang National University (Hybrid), South Korea

Maxence.Demougeot@irit.fr (M. Demougeot); Kevin.Delcourt@irit.fr (K. Delcourt); Jean-Paul.Arcangeli@irit.fr (J. Arcangeli); Sylvie.Trouilhet@irit.fr (S. Trouilhet); Francoise.Adreit@irit.fr (F. Adreit)

[1] https://www.irit.fr/OppoCompo/automatic-testing/

from other components. This makes software components easier to assemble, replace and reuse. Technically, they can be assembled by connecting their services, the assembly implementing an application.

The goal of opportunistic software composition is to provide the user with the right application at the right time, but without them having expressed needs and preferences due to the dynamics and unpredictability of the environment, which make their specification difficult. User-adapted applications are designed in bottom-up way and emerge from the environment, relying on automatically on-the-fly learnt knowledge about the user's needs and preferences.

To realize opportunistic software composition, an *Opportunistic Composition Engine* (OCE) has been designed: every service is locally managed by an "agent" that learns by reinforcement the connection preferences according to the context and decides on its own connections. Building an application consists in three steps, called an *OCE cycle*, and involves the user:

1. OCE probes the ambient environment to detect the available components.
2. According to their learnt knowledge on the user's preferences, the agents decide on their connections to others to collaboratively build up an assembly plan that defines an application and present it to the user.
3. Using the Interactive Control Environment (ICE), the user accepts, modifies or rejects the application.

User's actions in the last step are used as feedback for OCE. From this feedback, a reinforcement signal is computed from which OCE's agents learn by reinforcement the user's needs and preferences. As the cycles are repeated, OCE gains knowledge that it can then use to create more relevant applications.

Readers who wish to know more about OCE and ICE can refer to [4] and [5], and to [6] for a demo.

2.2. Need for testing OCE

A working prototype of OCE has been developed. However, it must be tested and evaluated, with a focus on the learning mechanism, in order to check that OCE builds assembly plans that fit the user needs and preferences. There are also alternative versions of OCE whose behavior must be compared.

Moreover, OCE is parameterizable, and the parameter values have to be adjusted. Among the parameters, some are ML-specific (for instance the amount of exploration vs. exploitation), others are user-specific and must be tailored to the individual: for instance, the coefficient that defines the user's preference for components or applications not yet encountered.

3. Testing human-centered ML-based systems

In a general way, the testing activity consists in defining test cases, running them, and asserting the correctness of the outputs against a specification. Usually, tests are used to check the behavior of a system by finding bugs or highlighting anomalies but they can not prove their absence [7].

3.1. Issues in testing ML-based systems

The growing interest in AI and ML is driving development teams to focus on testing and evaluation issues. However, testing ML-based systems is quite different from testing traditional software, i.e., those that do not rely on AI and ML.

A first point is that learning is assessed indirectly. When testers evaluate a ML-based solution, they seek to evaluate the correctness of the learning mechanism. But learning can hardly be isolated from the rest of the application, which is tested as a whole including the decision process. Therefore, the observed outputs are those of the whole application, not those of the only learning process.

Secondly, testers struggle to deal with randomness and non-determinism [8, 9], whether it comes from learning or decision. When an anomaly is detected, does it come from the non-deterministic nature of ML-based systems or is there a design or implementation error [10]? Therefore, to get significant results, experiments must be repeated a certain number of times to reduce the effects of non-determinism.

Moreover, in many cases, testers can not predict the produced outputs for specified inputs: there are not always oracles [11] or it is difficult to design ones [12]. To assess the accuracy of decisions, a way is to ask humans to perform the same tasks as the system with the same input data and observe the differences in the outputs [10].

Finally, the results are highly dependent on the training data [9]. Indeed, the choice of relevant data and test cases is critical for the verification of the properties of ML-based systems. However, training data spaces are often infinite and it is difficult but necessary to select the most relevant portions to train and evaluate a system [10].

The problem is even more complicated when using reinforcement learning because of the interactions between the learning agent and the environment [13] and the incremental building of knowledge. In this case, the system learns over time and makes decisions depending on both its current knowledge and the state of the environment. Thus, the dependence on the environment and its variability over time can lead to outdated or obsolete learnt knowledge.

3.2. Issues in testing interactive ML-based systems

Human-centered ML-based systems are interactive systems. In the field of interactive systems, it is common to carry out test campaigns involving real or potential users to evaluate interaction criteria, such as usability [14]. Such campaigns may require a large number of participants to produce significant results and be costly in time and money.

To alleviate these costs, designers often model synthetic users, e.g., personas, and simulate their actions. This raises some challenges:

- The potential introduction of biases [15]: how to insure the quality of this modelization?
- The need for specific design methods and tools.

In this paper, we focus on this second point and propose tools to take advantage of synthetic users and apply human-centered methodologies to ML-based systems [16].

In the case of interactive ML-based systems [17], humans are the source of learning data. The main issue is the complexity and the diversity of user profiles. Knowledge built from the interactions between the user and the learning system is personalized and may suit one user but not another. Moreover, the human users may be imprecise or change their opinions over time. It is thus difficult to assert the behavior of the learning system.

3.3. Issues in testing OCE learning process

OCE runs a distributed human-centered RL process where the human user is an integral part of the learning environment: as they accept, modify, or reject an assembly proposed by OCE, they provide feedback that OCE transforms into a reinforcement signal. This way, learnt knowledge is personalized and differs from one user to another. It also depends on the dynamics, whether it comes from the ambient environment or from the changing user's needs and preferences depending on the current situation and time. OCE decisions are then dependent on the knowledge gained.

Thus, the evaluation of the OCE learning process faces the issues of both ML-based and interactive systems: in particular, non-determinism and the challenge of assessing the quality of OCE's decisions on the one hand, dependence on the user, their profile and the variability of their needs and preferences on the other hand. In addition, the multiple forms of the ambient environment must be taken into account with the variety and number of components, as well as the dynamics.

4. OCE testing principles and tools

4.1. Related work

In the literature, there are several tools that allow testers to carry out experiments on ML-based solutions. We can cite the following:

- Arcade Learning Environment (ALE) [18] seeks to test the genericity of a learning algorithm, which must be able to play dozens of Atari 2600 games. The goal of a learning agent is to maximize the scores obtained on each game. ALE thus proposes evaluation metrics to compare and understand the performances of learning agents on the different games.
- OpenAI GYM [19] is a Python library that addresses the lack of normalization of reinforcement learning environments by offering a set of standard ones. It allows developers to test and compare their learning agents in these environments to benchmarks.
- The DotRL platform [20] is a framework that enables rapid development and test of reinforcement learning solutions. To carry out an experiment, testers select environments and learning agents among those proposed by the platform, or develop their own, and compare the performances.
- Cogment [21] facilitates the definition of complex human-centered machine learning architectures, with human users and/or automated agents interacting with each other with the aim of training humans and AI together and improve ML results. For each experiment, Cogment generates logs to allow the tester to analyze the results.

These tools offer a wide range of features to perform experiments. The Cogment framework is fairly new but seems to be the most attractive since it focuses on the presence of humans in the learning process and fits well the human-centered reinforcement learning paradigm. However, generally speaking, the gap between what these tools provide and the needs for testing OCE seems quite important and expensive to fill. Indeed, in order to test OCE using these tools, it would be necessary to make the predefined environments and agents interoperable with those of OCE in particular to use the available benchmarks. Among the problems, some seem difficult or even impossible to address such as taking into account the distribution nature of OCE learning. However, some concepts could be useful for the evaluation of OCE, e.g., the ALE evaluation metrics.

4.2. A tooling for OCE testing

Currently, we are exploring the state of the art in more depth. At the same time, we have undertaken to develop

our own tooling to automate the testing of OCE learning process, which consists of a pair of tools:

- OCE Scenario Maker, which supports the definition of tests,
- OCE Scenario Runner, which allows to automate and repeat their execution.

These are based on *(i)* a formalism to describe test scenarios with the ambient environment and the user's preferences and their possible variations over time, *(ii)* a way to repeat tests any number of times, and *(iii)* scores that measure the quality of OCE decisions.

A benefit of these tools is that they allow to carry out test campaigns more easily and at a lower cost, without having to develop concrete components and involve real users.

4.3. Principles

We define a *scenario* as a series of OCE cycles but without the results that could be provided by OCE: each cycle is described both by a set of software components of the ambient environment and the user's reaction to the proposal that OCE would make in this context.

To define the user's reactions without knowing the results provided by OCE, we introduce the concept of *ideal assembly*: an ideal assembly is the one that the user would prefer in a given context. Ideal assemblies model the user preferences in the different situations.

A scenario includes training cycles followed by assessment cycles. In a training cycle, the ideal assembly allows OCE to learn the user's need and preferences in the stated context. In an assessment cycle, the ideal assembly is used to verify that OCE has correctly learnt: to do so, a distance between this ideal assembly and the proposition of OCE is computed.

Defining a test consists thus in specifying a sequence of cycles with a variety of components, a more or less important number of them, and a more or less dynamic ambient environment. By specifying the ideal assembly for a cycle, the tester behaves as an oracle. Besides, specifying ideal assemblies allows to simulate different user profiles: users with more or less stable preferences, more or less open to new applications… Note that the selection of adequate training and assessment cycles (i.e., the design of test cases) is another challenge that we do not address in this paper.

In the following sections, we first present OCE Scenario Maker and OCE Scenario Runner. Then, we describe the architecture of the testing environment which associates OCE Scenario Maker, OCE Scenario Runner, and OCE (the engine) to run and test the latter.

4.4. OCE Scenario Maker

To describe a test scenario, the tester has to indicate, for each training and assessment cycle, both which components populate the environment and the ideal assembly.

For that, OCE Scenario Maker is a tool composed of:

- A library of dummy software components.
- A component creator that allows the tester to define dummy components by indicating their name and associated services, and add them to the library.
- A cycle creator that allows the tester to specify a cycle by choosing the participating components and expressing the ideal assembly (i.e., giving a list of connections between components).

In practice, OCE Scenario Maker is a single-page Web application. A GUI (see Sec. 4.7) assists the tester to graphically manipulate the scenario elements (components and connections). Once defined, the test scenario description is saved in a JSON file.

4.5. OCE Scenario Runner

Once the test scenario is defined with OCE Scenario Maker, it has to be executed to carry out automatic testing. OCE Scenario Runner is a Java desktop application, which handles these experiments and their repetition as many times as required to reduce the impact of non-determinism.

To start an experiment, several parameters must be set such as the test scenario JSON file produced by OCE Scenario Maker, OCE parameters, the version of OCE and the number of repetitions of the experiment. For each cycle, OCE Scenario Runner compares the assembly proposed by OCE and the ideal assembly and calculates a Jaccard Similarity index between the set of the connections proposed by OCE and the one of the ideal assembly; then, it computes an average score over all the cycles. This provides a measure of the distance between OCE's propositions and the ideal assemblies, and so indicates whether the OCE decisions make relevant applications emerge according to the learnt user's preferences. This measurement only makes sense for the assessment cycles as the previous cycles are used to create knowledge. It therefore makes the tester able to analyze and understand how OCE behaves.

4.6. Software architecture

Figure 1 shows how OCE Scenario Maker and OCE Scenario Runner fit in OCE runtime environment to create the testing environment. In the production environment, OCE interacts with the ambient environment by probing available components. Then, according to its knowledge

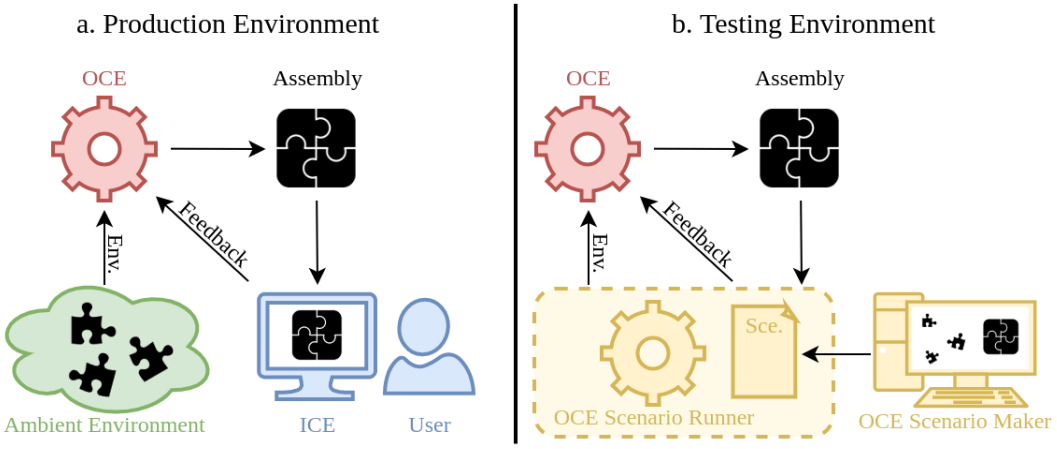

Figure 1: Architecture overview in production and testing configurations

about the user's preferences and needs, it plans an assembly which is displayed on ICE. Last, through ICE, the user gives a feedback to OCE that is converted into reinforcement signals for OCE learning agents.

OCE's modular architecture allows to seamlessly replace ICE and the ambient environment with OCE Scenario Runner, thus allowing simulation and automation of the interactions with the user and the environment. When running a scenario, at each cycle, OCE retrieves the set of the available components provided by OCE Scenario Runner, then proposes an assembly. Finally, OCE Scenario Runner returns the ideal assembly that is used by OCE to create feedback for its agents. Therefore, OCE Scenario Runner plays both the role of the ambient environment and the role of the user, in accordance with the scenario description produced using OCE Scenario Maker.

4.7. Testing OCE's behavior: a demonstration

The following video[2] shows the case of a tester who wants to check OCE's behavior in a simple use case where new components appear dynamically and the user preferences are stable. In this test scenario, the user is virtually surrounded by devices like switches and lamps, a switch can be connected to a single lamp, and OCE builds applications from them as they appear or disappear. Initially, three lamps (but only two of them work) and a switch are available in the user's surrounding environment, and the user prefers to use the first lamp. After a while, the third lamp works again and the user prefers to use it. An other component appears but do not affect the user's needs.

[2]https://www.irit.fr/OppoCompo/automatic-testing/

The video shows how the tester defines the component-based ambient environment and the ideal assemblies cycle by cycle, then runs the resulting multi-cycle scenario and gets similarity scores to check the relevance of OCE's decisions.

5. Conclusion

In this paper, we have addressed the problem of testing an intelligent solution for automated software composition. We have presented both a reflection about testing human-centered ML-based systems and a pair of tools that we have developed to automate the testing of our Opportunistic Composition Engine OCE. When defining scenarios using OCE Scenario Maker, the tester behaves as an oracle by designing different situations to simulate the environment dynamics and the user behavior. Then, experiments can be executed and repeated using OCE Scenario Runner. Repetition smoothes out the results altered by the presence of randomness in learning and decision.

Of course, questions arise concerning the design of the test sets. For now, we only propose tools that facilitate specification, execution, repetition and analyze of tests, but not a methodology: how to design test sets and how to assert their relevance? As it is highly dependent on the users, one way would be to involve them more in the design, e.g., build scenarios based on observations of their behavior or ask them about their preferences in different situations. Another way would be to improve usability of our tools in order to involve end-users in the implementation and the execution of tests. These are tracks to be explored further.

Although our work is still in progress, we hope that our approach can inspire other practitioners in the field

to develop tools for testing intelligent human-centered automated solutions.

References

[1] F. Sadri, Ambient intelligence: A survey, ACM Computing Surveys (CSUR) 43 (2011) 1–66.

[2] R. Sutton, A. Barto, Reinforcement Learning: An Introduction, 2nd ed., MIT Press, 2018.

[3] I. Sommerville, Component-based software engineering, in: Software Engineering, 10^{th} ed., Pearson Education, 2016, pp. 464–489.

[4] W. Younes, S. Trouilhet, F. Adreit, J.-P. Arcangeli, Agent-mediated application emergence through reinforcement learning from user feedback, in: 29th IEEE Int. Conf. on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), IEEE Press, 2020, pp. 3–8.

[5] S. Trouilhet, J.-P. Arcangeli, B. J.-M., M. Koussaifi, Model-Driven Engineering for End-Users in the Loop in Smart Ambient Systems, Journal of Universal Computer Science 27 (2021) 755–773.

[6] K. Delcourt, F. Adreit, J.-P. Arcangeli, K. Hacid, S. Trouilhet, W. Younes, Automatic and Intelligent Composition of Pervasive Applications, in: 19th IEEE Int. Conf. on Pervasive Computing and Communications (PerCom), Kassel (virtual), Germany, 2021.

[7] E. W. Dijkstra, On the reliability of programs, 1971. URL: https://www.cs.utexas.edu/users/EWD/transcriptions/EWD03xx/EWD303.html.

[8] D. Marijan, A. Gotlieb, M. K. Ahuja, Challenges of Testing Machine Learning Based Systems, in: Proc. of the 1st IEEE Artificial Intelligence Testing Conf., IEEE, San Francisco, CA, USA, 2019.

[9] K. Sugali, Software testing: Issues and challenges of artificial intelligence & machine learning, Int. J. of Artificial Intelligence & Applications 12 (2021) 101–112.

[10] C. Wan, S. Liu, S. Xie, Y. Liu, H. Hoffmann, M. Maire, S. Lu, Automated Testing of Software that Uses Machine Learning APIs, in: IEEE/ACM Int. Conf. on Software Engineering (ICSE), 2022. To appear.

[11] E. J. Weyuker, On testing non-testable programs, The Computer Journal 25 (1982) 465–470.

[12] S. Nakajima, Generalized Oracle for Testing Machine Learning Computer Programs, in: Software Engineering and Formal Methods - SEFM 2017, volume 10729 of *LNCS*, Springer, 2017, pp. 174–179.

[13] P. van Wesel, A. E. Goodloe, Challenges in the Verification of Reinforcement Learning Algorithms, Technical Report TM-2017-219628, NASA, 2017.

[14] J. C. Bastien, Usability testing: a review of some methodological and technical aspects of the method, Int. Journal of Medical Informatics 79 (2010) e18–e23.

[15] J. Salminen, S.-G. Jung, B. J. Jansen, Detecting demographic bias in automatically generated personas, in: Extended Abstracts of the 2019 CHI Conf. on Human Factors in Computing Systems, 2019, pp. 1–6.

[16] S. Amershi, D. Weld, M. Vorvoreanu, A. Fourney, B. Nushi, P. Collisson, J. Suh, S. Iqbal, P. N. Bennett, K. Inkpen, et al., Guidelines for human-AI interaction, in: Proceedings of the 2019 CHI Conf. on Human Factors in Computing Systems, 2019, pp. 1–13.

[17] S. Amershi, M. Cakmak, W. B. Knox, T. Kulesza, Power to the people: The role of humans in interactive machine learning, AI Magazine 35 (2014) 105–120.

[18] M. G. Bellemare, Y. Naddaf, J. Veness, M. Bowling, The arcade learning environment: An evaluation platform for general agents, Journal of Artificial Intelligence Research 47 (2013) 253–279.

[19] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba, OpenAI Gym, CoRR (2016). URL: https://arxiv.org/abs/1606.01540.

[20] B. Papis, P. Wawrzyński, dotRL: A platform for rapid Reinforcement Learning methods development and validation, in: 2013 Fed. Conf. on Computer Science and Information Systems (FEDCSIS), IEEE, 2013, pp. 129–136.

[21] AI Redefined, S. K. Gottipati, S. Kurandwad, C. Mars, G. Szriftgiser, F. Chabot, Cogment: Open Source Framework For Distributed Multi-actor Training, Deployment & Operations, CoRR abs/2106.11345 (2021). URL: https://arxiv.org/abs/2106.11345.