Generating Software Adaptations using Machine Learning

Nicolás Cardozo
Systems and Computing Engineering Department
Universidad de los Andes, Colombia
n.cardozo@uniandes.edu.co

Ivana Dusparic
CONNECT Research Centre
Trinity College Dublin, Ireland
ivana.dusparic@scss.tcd.ie

ABSTRACT

Recent availability of large amounts of sensor data from Internet of Things devices opens up the possibility for software systems to dynamically provide fine-grained adaptations to the observed environment conditions, rather than executing only static hard-coded behaviors. However, in current adaptive systems such adaptations still need to be specified beforehand, making the development process cumbersome as well as restricting the system adaptations only to those situations foreseen by the developers. We propose that adaptations should instead be generated by machine learning techniques at run time. Adaptive systems should incorporate an adaptation engine, which, through a mix of supervised and unsupervised learning, learns adaptive behaviors, and packages them into reusable software adaptations. We illustrate this idea with a simple proof-of-concept example using Context-oriented Programming, and focus on the challenges of implementing such an approach in the development of adaptive systems.

CCS CONCEPTS

• **Computing methodologies → Machine learning algorithms**;
• **Software and its engineering → Language features**;

ACM Reference format:
Nicolás Cardozo and Ivana Dusparic. 2018. Generating Software Adaptations using Machine Learning. In *Proceedings of ML4PL'18, Amsterdam, the Netherlands, July 16-23, 2018,* 2 pages.
https://doi.org/

1 INTRODUCTION

Adaptive software systems' goal is to seamlessly modify their behavior to detected situations in their surrounding execution environment. Such situations can be gathered from internal system monitors (*self*) or external sensors (*context*). There are several ways to realize behavior adaptations [6]. The Context-oriented Programming (COP) paradigm [3] offers a programmatic approach to dynamic adaptations. In COP, adaptations are conceived as the association of specialized behavior (*i.e., behavioral adaptations*) with a *context* object, representing information gathered from sensors. The systems dynamically adapts whenever the context object is sensed, triggering the activation of the behavioral adaptation *i.e.,* composing this behavior with the running system. If the context is no longer sensed, the system is recomposed excluding the contexts' associated behavioral adaptations.

However, in COP, as well as in other adaptation approaches, adaptive systems require the explicit definition of both the situations to which they can adapt, and the adaptation behavior. To remove this constraint, we propose to incorporate Machine Learning (ML) techniques within the programming language, able to identify situations in which adaptation is required, as well as to dynamically generate suitable adaptations continuously, based on executed behaviors. This will reduce the development effort of defining adaptations, and enable the system to react to situations unforeseen at development time. Packaged adaptations could also be reused across similar contexts or transferred to other entities within the system.

2 COP ADAPTATIONS USING ML

ML is incorporated into a COP language by means of an adaptation engine, shown in Figure 1, defining a process to generate adaptations. To generate adaptations, the system must be aware of both its surrounding environment (environment states), and the *atomic actions* taken –that is, the basic set of actions implemented in the system. Furthermore, the system must associate sequences of actions to a particular situation from the surrounding execution environment. Unsupervised learning algorithms (*e.g.,* Reinforcement Learning (RL) [5]) drive the selection of atomic action sequences that associate to a given environment state. These sequences can be learned based on their frequency and the contribution towards the system's goal. Once a suitable sequence is identified, an adaptation is generated by extracting the specific situation gathered from the system as the *context*, and the action sequence as the associated *behavioral adaptation*. Extracted adaptations are then executed whenever their associated context is sensed.

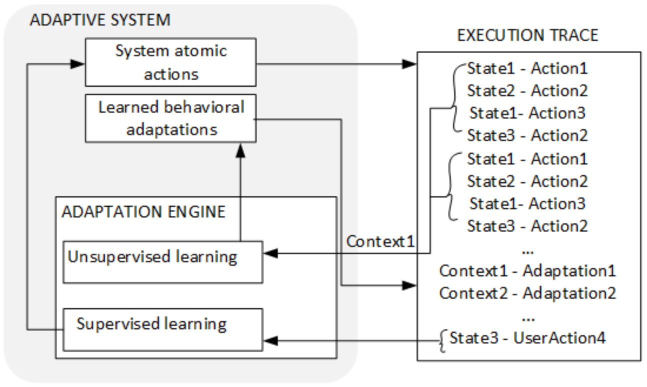

Figure 1: ML-based adaptation engine

ML4PL'18, July 16-23, 2018, Amsterdam, the Netherlands
© 2018 Copyright held by the owner/author(s). Publication rights licensed to Association for Computing Machinery.
ACM ISBN . . . $15.00
https://doi.org/

Additional to this process, the adaptation engine also offers a supervised learning module to take into account actions explicitly triggered by users. These actions are then incorporated to generate new (improved) adaptations from existing ones. The system behavior then consists of the mix of atomic actions and generated/learned adaptations, continuously evolving if and when new contexts and behavioral adaptations are detected.

We implemented a proof-of-concept RL-based adaptation engine integrated with the Context Traits [2] COP language. Snippet 1 shows an example of the adaptation generation by the adaptation engine in Context Traits. Both the context (Line 1) and behavioral adaptation (Lines 2-5) objects are generated at run time. We tested the adaptation engine using a cruise control driving assistant for autonomous vehicles. The goal of the assistant is to drive while respecting the traffic rules (speed limit and driving at the correct side of the road), and to avoiding crashes. The atomic actions are: `changeLane`, `speedUp`, and `slowDown`.

```
1 Context1 = new cop.Context({name:"Context1"});
2 Context1Adaptation = Trait({
3   systemGoal = function() {
4     action1(); action2(); action3(); action2();
5   } });
6 Context1.adapt(baseSystem,Ctx1Adaptation);
```

Snippet 1: Behavior generation stub

Figure 2 shows the results of preliminary evaluation of this approach in two environment situations requiring adaptation: changing the speed limit and changing the side of the road on which a vehicle should drive (switching from driving on the right-hand side of the road to the left-hand side of the road, as it might be the case, for example, when traveling from France to the UK). We observe that before the environment change, the base and adapted systems behave correctly (*base behavior*). When the speed limit or the desired driving side of the road change, the non-adapted system violates both goals, until the new behaviors are learned (*learned atomic actions*), and new adaptations generated (*learned adaptations*), leading once again to almost full compliance with the system goals. However, we also note a single instance in which adapted behavior violated the speed limit and crashed, evidencing a reliability problem with learned adaptations, discussed further in the next section.

3 CHALLENGES

ML can be successfully exploited to generate adaptations dynamically, tackling the problem of explicit definition of adaptations in adaptive systems. However, several issues need to be addressed before a full implementation can be considered. Here, we highlight three of those issues.

Full Adaptation Engine. The dynamic generation of adaptations is currently only semi-automated. To have full integration between ML and the language, a better interfacing to the adaptation engine is needed. The programming language needs to be enriched with abstractions that enable declarative-style specifications of system's atomic actions and high level goals (*e.g.,* `drive`) to generate

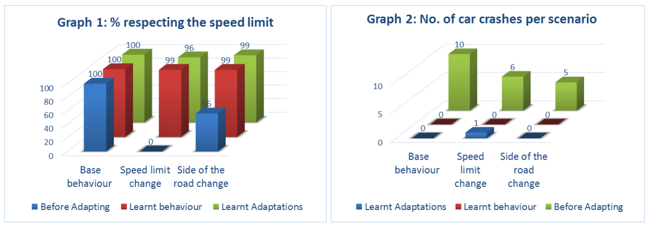

Figure 2: Performance of ML-generated Adaptations

adaptations. Details of the ML implementation, such as as parameter selection or sensor discovery, should be hidden away from the developer. Additionally, the system should be capable of integrating new sensors, data sources, atomic actions and goals at run time. Otherwise, the development effort of manually specifying the adaptations will not be removed but only shifted to specifying ML parameters in the adaptation engine.

Providing Behavior Guarantees. One of the main open challenges in adaptive systems that utilize ML is the difficulty of providing behavioral guarantees, not just in terms of convergence to optimality, but more crucially guaranteeing that system will not significantly deviate from the expected safe behavior (as illustrated by our crash example). This challenge is present in any programming language utilizing ML-generated code, as it requires testing and verification of yet unwritten code. Research on theoretical guarantees of ML implementations is in its infancy, with examples addressing *e.g.,* SMT solvers for verifying deep neural networks [4], and formal verification of RL [1]). Significant research is needed in this area addressing a wider variety of ML techniques to ensure consistency of ML-generated adaptations.

Impact on Development and Maintenance Cost. While learning rather than pre-specifying adaptations reduces development complexity and user effort, there is a growing concern that the use of ML could increase long-term software maintenance cost [7]. These issues arise from tight coupling of ML code and external data sources, and feedback loops from the environment; changes in the external world might make models behave unexpectedly and require continuous runtime monitoring. ML development is prone to a number of identified anti-patterns, such as excessive glue code and entangled handling of multiple data streams, that adaptive systems developers would need to be aware of and avoid.

REFERENCES

[1] Nathan Fulton and André Platzer. 2018. Safe Reinforcement Learning via Formal Methods: Toward Safe Control Through Proof and Learning. In *AAAI'18.*

[2] Sebastián González, Kim Mens, Marius Colacioiu, and Walter Cazzola. 2013. Context Traits: dynamic behaviour adaptation through run-time trait recomposition. In *Proceedings of International Conference on Aspect-Oriented Software Development (AOSD '13).* New York, NY, USA, 209–220.

[3] Robert Hirschfeld, Pascal Costanza, and Oscar Nierstrasz. 2008. Context-Oriented Programming. 7, 3 (March–April 2008), 125–151. http://www.jot.fm/issues/issue_2008_03/article4/

[4] Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. [n. d.]. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks.

[5] Sutton R. S. and Barto A. G. 1998. *Reinforcement Learning: An Introduction.* Bradford Book. The MIT Press, Cambridge, Massachusetts.

[6] Mazeiar Salehie and Ladan Tahvildari. 2009. Self-Adaptive Software: Landscape and Research Challenges. *ACM Transactions on Autonomous and Adaptive Systems* 4, 2 (May 2009), 14:1–14:42.

[7] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, and M. Young. 2014. Machine Learning: The High Interest Credit Card of Technical Debt. In *SE4ML: Software Engineering for Machine Learning, NIPS'14.*