JOT
JOURNAL OF
OBJECT TECHNOLOGY

Next Generation Context-oriented Programming: Embracing Dynamic Generation of Adaptations

Nicolás Cardozo* **and Ivana Dusparic†**
*Systems and Computing Engineering Department, Universidad de los Andes, Colombia
†School of Computer Science and Statistics, Trinity College Dublin, Ireland

ABSTRACT Context-oriented Programming (COP) first appeared in 2005 as a way to enable the dynamic adaptation of software systems to specific situations in their surrounding environment. Multiple COP languages have since been proposed, and used in numerous adaptive systems areas, enabling dynamic swapping and composition of adaptive behavior at run-time. However, until recently, all approaches relied on the offline pre-definition of adaptive behavior, limiting the adaptations to only those foreseen at design time. Auto-COP recently emerged as an approach to shift adaptation definition to run-time, if and when the need for adaptations to new contexts arises, by utilizing reinforcement learning techniques. In this paper, we use Auto-COP as a starting point to discuss the research path to achieve a completely dynamic adaptive system. We discuss the potential benefits of such an automated AI-based approach, present several application domain categories where dynamic adaptation definition would enable adaptivity breakthroughs, and discuss open challenges in developing such a fully automated approach.

KEYWORDS Context-oriented programming, Reinforcement learning, Dynamic software adaptation

1. Introduction

Context-oriented Programming (COP) (Appeltauer et al. 2009; Salvaneschi et al. 2012) is a programming language technique designed to enable dynamic adaptations of software systems to their surrounding execution environment. COP has been envisaged to "*bring a similar degree of dynamicity to the notion of behavioral variations that object-oriented programming brought to ad-hoc polymorphism*" where one of its essential features is "*dynamic representation of layers and their scoped activation and deactivation in arbitrary places of the code*" (Hirschfeld et al. 2008).

Since its appearance in 2005 the work on COP has focused in proposing new programming languages and language-level abstractions for the definition of modular adaptations with a clean separation of concerns. The proposed abstractions differ across COP languages, but generally consist of a way to represent the information gathered from the environment (*i.e., contexts*), the specialized behavior to exhibit in a particular context (*i.e., behavioral variations*), and the means to activate (*i.e.,* incorporate behavioral variations with the running system) and deactivate (*i.e.,* withdraw behavioral variations from the system) contexts.

COP has been positioned as a suitable paradigm for complex self-adaptive systems (Cardozo & Dusparic 2020a; Cardozo & Mens 2022) and has been applied in the domains of pedestrian navigation systems, home automation (González et al. 2011), and autonomic storage (Ghezzi et al. 2010). Recent work has explored COP extensions to enable the specification and management of interactions between multiple remote entities (Fandiño de la Hoz et al. 2019), and to dynamically manage interactions (*e.g.,* conflicts or composition) between multiple context-triggered adaptations (Cardozo et al. 2017; Cardozo & Dusparic 2020b).

However, while the behavior of COP-based systems is effectively adapted at run-time, these adaptations are not taken from the surrounding execution environment, but rather are predefined by developers of COP systems (Cardozo & Clarke 2015). As a consequence, despite the original vision, COP-based system are not yet fully dynamic or adaptive to their context, but constrained to adapt to the situations foreseen at design time.

In parallel to recent developments in COP, advancements in

JOT reference format:
Nicolás Cardozo and Ivana Dusparic. *Next Generation Context-oriented Programming: Embracing Dynamic Generation of Adaptations*. Journal of Object Technology. Vol. 21, No. 2, 2022. http://dx.doi.org/10.5381/jot.2022.21.2.a5

a range of Artificial Intelligence (AI), and in particular Machine Learning (ML) techniques, have given rise to increasing utilization of ML techniques in programming language development and software engineering, for example, source code generation, bug fixing, and program synthesis (Watson et al. 2020). Reinforcement Learning (RL), as one of such ML techniques, enables the adaptation of a system at run time, based on the information gathered from the environment (*i.e.,* the reward obtained for achieving the system's goal). Similar to the case of COP, in RL the adaptations are restricted to the predefined set of actions of the learning agent. Moreover, the definition of such adaptations lack modularity and hinders the system's maintenance and evolution.

To counter the aforementioned problems, and to reuse the adaptation models from both COP and ML, the Auto-COP (Cardozo & Dusparic 2021) framework has been recently proposed, which uses ML to detect, learn, and incorporate adaptations (*i.e.,* contexts and their associated behavioral variations) into the system at run-time. Auto-COP represents the first proposal/proof of concept for the possibility to achieve full dynamic behavior through the use of AI techniques.

In this paper, we position a research path to achieve fully dynamic adaptations as envisioned by COP, using the Auto-COP framework as a starting point to achieve such goal. Auto-COP aims to achieve full run-time adaptivity, by using RL options (Stolle & Precup 2002) to monitor the system surrounding environment, continuously extracting the current system state from a set of sensed or monitored variables, and by lifting such state as possible execution contexts. Similarly, Auto-COP associates sequences of actions arising at run-time to a particular state and lifts them as the behavioral variation associated with the detected context. With these two components extracted at run-time, we are able to effectively adapt the system behavior without any prior knowledge of specific situations.

In the rest of this paper we first recap the main features of Auto-COP, then propose potential novel applications that could be enabled by using full run-time adaptation in COP. Finally, we present challenges and research questions associated with the development of a fully automated and dynamic approach, in the context of further development of Auto-COP, as well as wider use of ML to increase run-time adaptivity of self-adaptive systems.

2. Auto-COP

Auto-COP (Cardozo & Dusparic 2021) is a new framework proposed to tackle the problem of adapting software systems to the dynamic situations of its surrounding environment. As the surrounding system environment changes, the system's requirements and observed behavior should change as well. Therefore, we must be able to incorporate unannounced changes to unknown situations. To do so, we identify four characteristics the system should exhibit:

(C1) Must interact with users or have autonomous agents to take decisions about the actions to take for different run-time situations

(C2) Must have a defined goal, and the means to monitor the system actually moves towards such a goal

(C3) Must have a finite set of states

(C4) Must have a finite set of actions

If these four characteristics are satisfied by a software system, then we can use Auto-COP to generate and automate adaptive behavior for specific run-time situations dynamically. The Auto-COP framework is defined by the process and components depicted in Figure 1.

As an example of the type of systems targeted with Auto-COP, let us take a warehouse delivery system. In our example, a delivery robot moves warehouse goods from their storage location (expressed as a two dimensional grid) to the designated drop-off point in the warehouse. Expressed using our proposed requirements we see that: (Characteristic **(C2)**) the goal of the robot is to transport goods to the designated location, (Characteristic **(C3)**) the system state is captured by the grid representing the warehouse, the state of the robot (*i.e.,* wether is currently moving an object), and (Characteristic **(C4)**) the set of actions that the robot executes includes moving in the direction of any of the four cardinal points, picking up an object and dropping-off and object. Finally, Characteristic **(C1)** is present in the system, as the warehouse robot can be controlled by human operators or as an autonomous agent.

The Auto-COP process begins by first monitoring relevant variables of the system's surrounding execution environment to capture specific system states (possible from Characteristic **(C2)**). These states are sensed information from the environment or monitored system variables that are captured dynamically. In Figure 1, as part of the same monitoring process, together with every state, the system adaptation engine continuously monitors the traces of executed actions, captured by the System primitive actions component in Figure 1 (possible from Requirements **(C3)** and **(C4)**). The captured states and actions in the process are shown as the environment component in the rightmost part of Figure 1, using a generic set of captured state - action pairs. Note that the captured state can refer to a single sensed variable or to a combination of variables, as required to decide on the actions to take to satisfy the system goal. For example, in the warehouse system, the state can be represented by the current position of the robot and whether its carrying an object or not. The actions correspond to the specific robot movements that the robot can take from the given state (*e.g.,* move north).

The second step in the process, depicted as the Adaptation Engine in Figure 1 is to extract contexts and behavioral variations from the captured execution traces. To do so, Auto-COP implements a technique based on RL options (Stolle & Precup 2002). RL options, unlike basic RL which learns a single action to execute in each state, focuses on observing sequences of actions that often repeat in the same order during the execution; those learnt sequences are called options and are executed instead of individual actions when a state in which they're suitable for execution is observed. For example, one such sequence in Figure 1 is *{Action1, Action2, Action3}* executed starting from

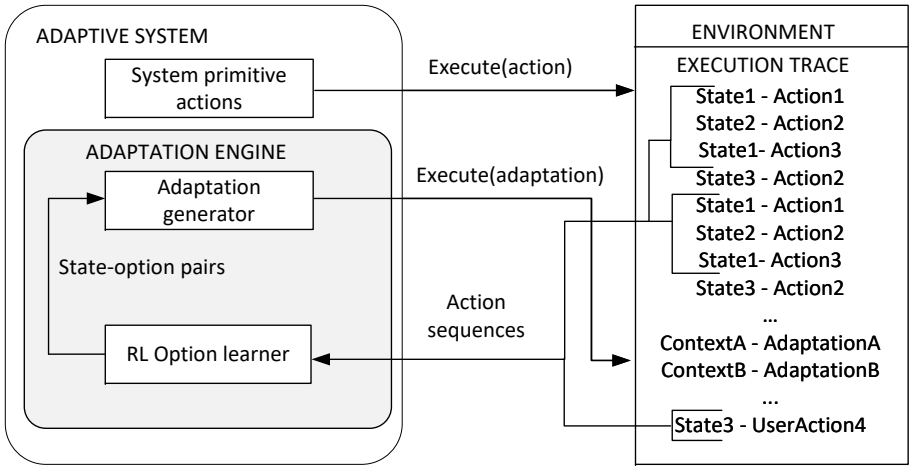

Figure 1 Adaptation generation process using Auto-COP

State1. To learn these sequences, Auto-COP considers all possible options (action sequences) observed in execution traces, and measures the progress of each of them towards reaching the system goal (possible from Characteristic **(C2)**), using RL rewards. For example, in our warehouse system, if taking an action from a state takes the robot closer to the goal (of delivering and object), then it obtains a large reward. If the action does not take the robot closer to the goal, then the reward would be zero, or negative. For RL options, the reward achieved by the option under consideration, is the sum of individual action rewards within the sequence. Each action sequence is then mapped to a particular state, the starting state of the action sequence. In Auto-COP, this state is extracted as a particular context object. Then, the option with the largest reward is used as the action sequence to extract into the behavioral variation associated with the context. For further details of the process, and the algorithm listing, we refer the reader to Cardozo & Dusparic (2021). Note that Auto-COP is agnostic to the source of actions, *i.e.,* it behaves it same regardless whether the action source is a user interacting with the system (*e.g.,* executing an action *UserAction4* in Figure 1), an autonomous agent, or a sequence of predefined actions for specific situations (possible from Characteristic **(C1)**).

The final step in the process, the Adaptation generator component part of the Adaptation Engine in Figure 1, is to extract the options with the highest reward at a given state as full behavioral adaptations. The effect of this process is that as the system executes further, when a particular state extracted as a context is sensed, we can now proceed with the activation of the context, making its associated behavioral variations available in the system. In consequence, this will trigger the execution of the generated adaptation (*i.e.,* the action sequence defined by the RL option). Once the state is no longer sensed, the context is deactivated and the base behavior is restored. In our example, the behavior of the system could adapt the warehouse robot to autonomously take an object from its storage location to the delivery point, without requiring any user intervention, or decision making from an autonomous agent.

In such a way, Auto-COP-based systems can adapt to a changing environment without requiring the upfront definition of adaptations and their associated situations, nor requiring user interventions in order to introduce new adaptive behavior. Adaptations are learned at run-time, but unlike pure ML-based systems, it enables run-time adaptation while retaining the modularity aspects offered by COP.

3. Application Scenarios for Generated Adaptations

Based on the aforementioned requiremetns and use-cases explored for Auto-COP, we propose different application domains in which run-time adaptivity requirements match the proposed capabilities of Auto-COP, and therefore could benefit from using Auto-COP to automate their behavior to unknown environment conditions.

Self-healing systems

Self-healing systems (Ghosh et al. 2007) are able to autonomously detect faults in their execution and recover from them. These approaches use diagnosis analysis (*e.g.,* root cause analysis) to identify the cause of errors, and to determine the execution points (usually specific states in which failures can occur **(C3)**) in which a sequence of recovery actions **(C4)**, normally pre-defined by developers, can be initiated to ensure error-free continuation of the running system **(C2)**. The behavior of self-healing systems is a good complement to our proposed approach. Generated adaptations can be seen as sequences of recovery actions and root causes of the failure can be extracted as contexts. Using Auto-COP, self-healing systems can become context-aware and able to autonomously perform recovery actions to disruptive behavior not foreseen at design time. Adaptations, *i.e.,* sequences of recovery actions, can be learned from corrective behavior observations at run time (*e.g.,* those executed by a user or administrator **(C1)**). This would

result in fully dynamic and smart self-healing software systems, increasing the tolerance/resiliency of complex software systems to unanticipated run-time disruptions. Such behavior, would be beneficial, for example, to add corrective behavior to reactive and distributed systems.

Lively systems

Lively systems are those that do not stop their execution in face of errors, modifications, or updates (*e.g.,* the Squeak/Smalltalk system (Ingalls et al. 1997), Lively Kernel (Ingalls et al. 2008)). The objective of such systems is to be highly adaptable **(C2)**; therefore all system entities can be modified, reused, and composed, even for purposes not intended originally **(C1)**. Modifications in lively systems follow the idea behind Auto-COP in which the changes are unknown beforehand, and the complete system execution must adapt to the new conditions. As such, using Auto-COP, the system state in which changes take place can be extracted as contexts **(C3)**, and the sequence of changes as the associated behavioral variations **(C4)**. In this way, we can automate the required changes for any system entity. Moreover, with the automation of changes, we can revert changes according to the system state autonomously, offering even more flexibility and possible compositions between the system entities.

Autonomous service composition

In service composition, interaction between service components is prescribed by dedicated models or predefined hook-functions (**(C4)** and **(C3)**). However, with the advent of IoT systems where different service consumers and providers are available in the environment, and they can appear and disappear unannounced (Ariza et al. 2021), building such models is not straightforward. Previously unknown service components should be able to interact spontaneously with each other if their input and output interfaces correspond (Cardozo 2016) **(C2)**. Interaction between components could be managed online by a user, but could also emerge from components in the environment querying their interfaces **(C1)**. Once a match is found the corresponding link between the components is established. Such dynamic environments are another application area for Auto-COP. In this case, we could extract the available service components currently available in the environment and their interfaces as the context, and adapt the composition mechanism as the sequence of actions required to link the components. This can result in a fully automated emergence of services in a highly dynamic and unknown environment.

4. Open Challenges in Automating Adaptations

While Auto-COP makes a substantial step towards generating adaptations dynamically and removing the need for their explicit manual definition, multiple open research issues need to be addressed before fully automated adaptations can be achieved. We discuss three main challenges here.

Fully Automated Adaptation Engine

Using AI to generate adaptations in general, and specifically using RL in the case of Auto-COP, if not fully automated, comes with the risk of removing the need for manual specifications of adaptations only for it to be replaced by the need for manual specifications of AI processes generating those automated adaptations. Currently, in Auto-COP, this process is only semi-automated –that is, only parts of the RL process are generic to all Auto-COP applications, while parts have to be tailored to each particular implementation, by an experienced RL developer. Each RL process relies on specifying four main constructs: states, actions, rewards, and learning hyperparameters (*e.g.,* learning rate). In Auto-COP, states and actions are extracted from the execution traces of the underlying adaptive system (as illustrated in Figure 1) and can therefore be generated automatically. However, unless the underlying adaptive system also uses RL to learn the original adaptations, rewards (based on the goals of underlying adaptive system), and hyperparameters (based on the characteristics of the environment in which system operates) these would need to be specified manually. To automate this process, we propose that COP-style languages need to be enriched with abstractions that enable declarative-style specifications of systems' high level goals. Details about the RL process, such as parameter selection or sensor discovery, should be, not only separated from the application code, but ideally also hidden away from the developer. This will enable development of COP systems without requiring developers to have knowledge of particular ML techniques. Additionally, the system should be capable of integrating new sensors, data sources, atomic actions and goals at run-time. Some of the early work in this area enables dynamically generating state space representations (Guériau et al. 2019), and learning reward functions rather than specifying them (Edwards & Isbell Jr. 2019). This needs to be adapted and integrated with Auto-COP to achieve full automation.

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 the system will not significantly deviate from the expected safe behavior. 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 (Katz et al. 2017), and formal verification of RL (Fulton & Platzer 2018)). Significant research is needed in this area addressing a wider variety of ML techniques to ensure consistency of ML-generated adaptations.

However, additional issues in providing behavior guarantees using Auto-COP-based systems arises from the use of RL options (*i.e.,* fixed sequences of actions) to encapsulate adaptations. Once learned, such sequences are designed to be executed fully and uninterrupted. In more complex systems, where multiple adaptations are executed simultaneously affecting different parts of the system, the potential for a conflict during the execution of action sequences increases. Currently, in self-adaptive systems with manual adaptations, checks for such conflicts are performed at design-time; in Auto-COP, conflicts would need

to be detected and resolved automatically at run-time after adaptations are generated (such as Cardozo & Dusparic (2020b)), adding an additional layer of complexity to the learning process.

Impact on Development and Maintenance Cost

While learning adaptations instead of pre-specifying them can reduce development complexity and effort, as well as increase the run-time adaptability of the system, the use of ML to generate adaptations could increase long-term software maintenance cost (Sculley et al. 2014). These issues arise from the 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 run-time 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. Similarly to the challenge of fully automating adaptations, we propose that COP languages need to be enriched to specifically cater for ML constructs, to retain the modularity of the underlying adaptive system, not just to enable Auto-COP integration, but to tackle increasing proliferation of use of ML in all aspects of adaptive system development.

5. Conclusion

Context-oriented Programming (COP) has proved effective in enabling run-time adaptation of software systems to their surrounding execution environment in a modular way. However, the adaptation of software systems using COP techniques is not fully dynamic, as the system is only able to adapt to those situations previously foreseen by the system developers at design time. In parallel, Reinforcement Learning (RL) techniques enable systems to autonomously learn specific actions, or sequences of actions to optimize the system behavior from a given state. However, these techniques lack modularity, hindering the maintainability and evolution of RL-based systems.

To achieve true run-time adaptation, automatic generation of adaptations is required. In this paper, we use a recent proposal to achieve such adaptation, named Auto-COP, to present a discussion into the chatachteristics of the underlying systems' kind suitable for automated adaptation generation. We further present three promising application scenarios that would benefit from fully automated adaptation, and discussed the main open research challenges. These challenges constitute the roadmap and research agenda to foster both COP and RL, for dynamic and autonomous adaptations in general, as well as the further development of Auto-COP.

Acknowledgments

This work was supported, in part, by the Science Foundation Ireland (SFI) Grant No. 16/SP/3804 (Enable) as well as a research grant from SFI and the National Natural Science Foundation of China (NSFC) under the SFI-NSFC Partnership Programme Grant Number 17/NSFC/5224.

References

Appeltauer, M., Hirschfeld, R., Haupt, M., Lincke, J., & Perscheid, M. (2009). A comparison of context-oriented programming languages. In *International workshop on context-oriented programming* (pp. 1–6). New York, NY, USA: ACM. doi: 10.1145/1562112.1562118

Ariza, J., Garcés, K., Cardozo, N., Sánchez, J. P. R., & Vargas, F. J. (2021). IoT architecture for adaptation to transient devices. *Journal of Parallel and Distributed Computing*, *148*, 14–30.

Cardozo, N. (2016). Emergent software services. In *Proceedings of the 2016 acm international symposium on new ideas, new paradigms, and reflections on programming and software* (pp. 15–28). New York, NY, USA: ACM.

Cardozo, N., & Clarke, S. (2015, July). Context slices: Lightweight discovery of behavioral adaptations. In *Proceedings of the context-oriented programming workshop* (pp. 2:1–2:6). ACM. doi: http://dx.doi.org/10.1145/2786545.2786554

Cardozo, N., & Dusparic, I. (2020a). Language abstractions and techniques for developing collective adaptive systems using context-oriented programming. In *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)* (p. 133-138). doi: 10.1109/ACSOS-C51401.2020.00044

Cardozo, N., & Dusparic, I. (2020b, Jun). Learning run-time compositions of interacting adaptations. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. ACM.

Cardozo, N., & Dusparic, I. (2021). Auto-COP: Adaptation generation in context-oriented programming using reinforcement learning options. *arXiv preprint arXiv:2103.06757*.

Cardozo, N., Dusparic, I., & Castro, J. H. (2017). Peace COrP: Learning to solve conflicts between contexts. In *International workshop on context-oriented programming*. New York, NY, USA: ACM. doi: 10.1145/3117802.3117803

Cardozo, N., & Mens, K. (2022). Programming language implementations for context-oriented self-adaptive systems. *Information and Software Technology*, *143*, 106789. Retrieved from https://www.sciencedirect.com/science/article/pii/S0950584921002305 doi: https://doi.org/10.1016/j.infsof.2021.106789

Edwards, A. D., & Isbell Jr., C. L. (2019). Perceptual values from observation. *CoRR*, *abs/1905.07861*.

Fandiño de la Hoz, J. D., Sosa, J. S., & Cardozo, N. (2019). Distributed Context Petri Nets. In *Proceedings of the Workshop on Context-Oriented Programming* (pp. 24–31). New York, NY, USA: ACM.

Fulton, N., & Platzer, A. (2018, February). Safe reinforcement learning via formal methods: Toward safe control through proof and learning. In *AAAI'18*.

Ghezzi, C., Pradella, M., & Salvaneschi, G. (2010). Programming language support to context-aware adaptation: a case-study with Erlang. In *Proceedings of the international symposium on software engineering for adaptive and self-managing systems* (pp. 59–68). New York, NY, USA: ACM.

doi: 10.1145/1808984.1808991
Ghosh, D., Sharman, R., Rao, H. R., & Upadhyaya, S. (2007). Self-healing systems—survey and synthesis. *Decision support systems*, *42*(4), 2164–2185.
González, S., Cardozo, N., Mens, K., Cádiz, A., Libbrecht, J.-C., & Goffaux, J. (2011). Subjective-C: Bringing context to mobile platform programming. In B. Malloy, S. Staab, & M. van den Brand (Eds.), *Proceedings of the international conference on software language engineering* (Vol. 6563, pp. 246–265). Springer-Verlag. doi: 10.1007/978-3-642-19440-5_15
Guériau, M., Cardozo, N., & Dusparic, I. (2019, June). Constructivist approach to state space adaptation in reinforcement learning. In *International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*. IEEE.
Hirschfeld, R., Costanza, P., & Nierstrasz, O. (2008, March–April). Context-oriented programming. *Journal of Object Technology*, *7*(3), 125–151. Retrieved from http://www.jot.fm/issues/issue_2008_03/article4/
Ingalls, D., Kaehler, T., Maloney, J., Wallace, S., & Kay, A. (1997). Back to the future: The story of Squeak, a practical Smalltalk written in itself. *ACM SIGPLAN Notices*, *32*(10), 318–326. doi: 10.1145/263700.263754
Ingalls, D., Palacz, K., Uhler, S., Taivalsaari, A., & Mikkonen, T. (2008). The Lively Kernel A Self-supporting System on a Web Page. In R. Hirschfeld & K. Rose (Eds.), *Self-Sustaining Systems* (pp. 31–50). Berlin, Heidelberg: Springer Berlin Heidelberg.
Katz, G., Barrett, C. W., Dill, D. L., Julian, K., & Kochenderfer, M. J. (2017, July). Reluplex: An efficient SMT solver for verifying deep neural networks. In *International conference on computer aided verification* (pp. 97–117).
Salvaneschi, G., Ghezzi, C., & Pradella, M. (2012, August). Context-oriented programming: A software engineering perspective. *Journal of Systems and Software*, *85*(8), 1801–1817. doi: 10.1016/j.jss.2012.03.024
Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., … Young, M. (2014). Machine learning: The high interest credit card of technical debt. In *Workshop on Software Engineering for Machine Learning*.
Stolle, M., & Precup, D. (2002). Learning options in reinforcement learning. In *Proceedings of the International Symposium on Abstraction, Reformulation and Approximation* (pp. 212–223). Berlin, Heidelberg: Springer-Verlag.
Watson, C., Cooper, N., Nader-Palacio, D., Moran, K., & Poshyvanyk, D. (2020). A systematic literature review on the use of deep learning in software engineering research. *CoRR*, *abs/2009.06520*. Retrieved from https://arxiv.org/abs/2009.06520

About the authors

Nicolás Cardozo Nicolás Cardozo is an Assistant Professor and research lead at FLAGlab in the Systems and Computing Department at Universidad de los Andes. Nicolás specializes in the design and implementation of programming languages for distributed adaptive software systems. Nicolás has worked in the implementation of dynamic distributed adaptations in the smart cities domain from different perspectives, such as automated personalized assistants and evolutionary models for dynamic adaptations. Before becoming an Assistant Professor, Nicolás was a postdoctoral fellow at Trinity College Dublin, and Vrije Universiteit Brussel. Nicolás received his doctoral diploma from the Université catholique de Louvain, and Vrije Universiteit Brussel in Belgium. You can contact the author at n.cardozo@uniandes.edu.co or visit http://flaglab.github.io.

Ivana Dusparic Ivana Dusparic is an Ussher Assistant Professor in the School of Computer Science and Statistics at Trinity College Dublin since 2016. She holds a BSc from La Roche College, PA, USA (2003), and MSc (2005) and PhD (2010) from Trinity College Dublin. Her expertise and research interests lie in the use of Artificial Intelligence techniques (reinforcement learning, multi-agent systems) to achieve autonomous optimization of large-scale heterogeneous infrastructures, applied to smart cities and sustainable urban mobility. You can contact the author at ivana.dusparic@tcd.ie.