CHALMERS
UNIVERSITY OF TECHNOLOGY

Keeping intelligence under control

Keeping Intelligence under Control

Piergiuseppe Mallozzi
Chalmers University of Technology | University of Gothenburg
Sweden
mallozzi@chalmers.se

Patrizio Pelliccione
Chalmers University of Technology | University of Gothenburg
Sweden
patrizio.pelliccione@gu.se

Claudio Menghi
Chalmers University of Technology | University of Gothenburg
Sweden
menghi@chalmers.se

ABSTRACT

Modern software systems, such as smart systems, are based on a continuous interaction with the dynamic and partially unknown environment in which they are deployed. Classical development techniques, based on a complete description of how the system must behave in different environmental conditions, are no longer effective. On the contrary, modern techniques should be able to produce systems that autonomously *learn* how to behave in different environmental conditions.

Machine learning techniques allow creating systems that learn how to execute a set of actions to achieve a desired goal. When a change occurs, the system can autonomously learn new policies and strategies for actions execution. This flexibility comes at a cost: the developer has no longer full control on the system behaviour. Thus, there is no way to guarantee that the system will not violate important properties, such as safety-critical properties.

To overcome this issue, we believe that machine learning techniques should be combined with suitable reasoning mechanisms aimed at assuring that the decisions taken by the machine learning algorithm do not violate safety-critical requirements. This paper proposes an approach that combines machine learning with run-time monitoring to detect violations of system invariants in the actions execution policies.

KEYWORDS

Autonomous systems, Safety-critical, Reinforcement learning, Machine learning, Run-time verification

ACM Reference Format:
Piergiuseppe Mallozzi, Patrizio Pelliccione, and Claudio Menghi. 2018. Keeping Intelligence under Control. In *Proceedings of International Workshop on Software Engineering for Cognitive Services (SE4COG2018).* ACM, New York, NY, USA, Article 4, 4 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

There is a need for the systems to be "*smart*". Smart systems are systems that continuously sense the environment in which they operate, are able to detect changes, and react to those changes.

The environment in which smart systems operate is usually dynamic, uncontrollable, and partially known. For example, in the automotive domain, drivers behaviours are sometimes unpredictable, animals unexpectedly can cross roads, etc. Smart systems have to deal with such unpredictability and uncertainty in a "*self-adaptive*" manner. Self-adaptation refers to the capability, performed at run-time without human intervention, of a system in autonomously changing its behaviour in response to changes [1, 2].

Classical development techniques require to fully describe the system behaviour in *all* the different environmental conditions. This is unpractical — if not even impossible — in smart systems where there is a high number of environmental conditions to be considered for adaptation. For this reason, modern development techniques for smart systems must rely on techniques that allow creating systems that autonomously *learn* how to behave in different environmental conditions.

Machine learning techniques are able to autonomously learn how to act on a running system to achieve a desired goal. Such techniques are based on models trained on data and examples rather than logic programs with predefined rules. The programmer is replaced by a machine that can continuously update its models as new data comes from the environment. After training them, machine learning models allow for the effective handling of changes i.e., when a change occurs the system autonomously learns new policies for actions execution. The use of machine learning drastically changes how software systems are developed, i.e., the choice of which the actions to be executed in the different environmental condition is no longer in the developer's hands but is rather automatic. In this paper, we use reinforcement learning since it is a powerful machine learning technique for decision making.

The automatic support provided by machine learning techniques moves control from the developers to the system hands. Thus, the developer is no longer able to ensure that the system will not violate important properties, i.e., invariants, that may be used to represent for example safety-critical properties. In the automotive domain, for example, the developer has no longer control on ensuring whether a self-driving car is going to take a decision that is safe for the particular situation or not. Wrong decisions performed by the machine learning engine may result in car accidents and serious injuries for the passengers. Thus, we believe that while there is a need for systems to be self-adaptive, there is also a need to "*keep intelligence under control*".

We envision a new approach, in the following named WiseML, to ensure that machine learning decisions do not cause the violation of a set of "important" properties. This approach combines machine learning, and specifically Reinforcement Learning (RL) [3], with *run-time monitoring* techniques which aim at ensuring the preservation of important safety-critical requirements. On one side, WiseML allows the systems to adapt through the use of machine learning techniques. On the other side, WiseML employs run-time monitoring to continuously check that the policies suggested by

SE4COG'18, May 28–29, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5740-1/18/05…$15.00
https://doi.org/10.1145/3195555.3195558

reinforcement learning will not violate a set of safety-critical requirements.

The paper is structured as follows. Section 2 describes some of the challenges of reinforcement learning when used as instrument to enable run-time adaptation. Section 3 describes WiseML, our envisioned approach. Section 4 presents reinforcement learning: a machine learning technique that can be used within WiseML to enable self-adaptation. Section 5 presents monitoring techniques that can be used within WiseML. Section 6 discusses the proposed approach. Section 7 concludes with final remarks.

2 CHALLENGES OF REINFORCEMENT LEARNING

At each step, a reinforcement learning agent perceives observation (the state of the environment), applies actions, and receives a reward. The goal that the agent has to achieve is expressed by conveying a reward signal for each action it applies. The agent will eventually learn a policy, i.e. the action to be applied to the observed state, which maximises the cumulative expected reward. In this section, we describe some of the challenges of reinforcement learning as also pointed out by Koopman [4].

Overfitting problem - Reinforcement Learning (RL) techniques require a set of training data that have to be independent of the validation data to avoid *overfitting*. One main problem with machine learning methods is that they are optimised for average cost function and they do not guarantee for corner cases. Challenges in this area are compromised by the fact that when using methods such as neural networks, it is difficult for humans to understand the rules that have been learned by simply looking at its weights.

Black swan - When the neural network learns the rules from a training set, if certain data is missing or wrongly correlated to the training data, the network may fail and potentially cause safety hazards. In other words, if there is a special case that the system has not experienced, it cannot correctly predict such case; this is known as the *black sworn problem* [5]. Hence, it is hard to detect and isolate bugs where the behaviour is not expressed with traditional lines of codes but entrusted to a neural network. The network would need to be retrained with also the potential risk to "unlearn" correct behaviours.

Reward hacking - An incorrect specification of the reward function can cause unexpected behaviours to the agent. One of the problems is *reward hacking* [6]: when the reward function is not exactly representing the designer's intention, the agent might optimise towards an imprecise reward function and behave unexpectedly meaning that the agent exploits the reward function and manages to get a high reward without achieving the designer's intentions, but instead optimising towards the rewards function that is indeed not exactly representing the designer's intentions. For example, in the case of a cleaning robot, the reward function might give a positive reward for not seeing any mess then the agent might learn to disable its vision rather than cleaning up. Instead, if the reward is given only when the robot actually cleans up then the robot might learn to make a mess first and then cleaning up so that it keeps receiving more and more rewards.

3 WISEML

Figure 1 shows an overview of WiseML.

WiseML considers both functional and non-functional requirements in the adaptation framework. Functional requirements are the *Goals (G)* WiseML must achieve. Non-functional requirements refer to the properties WiseML must ensure and they are represented in the form of *Invariants (I)*.

WiseML receives as input the goals G to be achieved ((2)) and a set of invariants I to be ensured ((3)). It perceives the state of the environment through a set of input variables ((1)) and performs the actions ((4)) to apply to the environment that aims at reaching the desired goals.

WiseML uses a reinforcement learning agent as machine learning engine, indicated in Fig. 1 using an appropriate component, and a monitor component ((7)). Once trained the reinforcement learning agent automatically computes the action to be executed. The monitor component blocks the actions that most probably will violate invariants and provides feedback to the machine learning component so that it will learn from mistakes intercepted by the run-time monitor ((8)). The interplay between the machine learning and the monitoring components is designed to enable the integration of enforcement techniques as well as other techniques able to manage the erroneous behaviours intercepted by the monitor. For instance, this would permit to switch to a safety mode in the case of the system is in a critical situation.

For example, an RL agent in charge of driving may receive rewards related with the following goals: *stay in the middle of the lane*, *drive to the desired speed*, *avoid obstacles* etc. The monitor will continuously check at runtime that the agent does produce actions that will violate important invariants of the system: *keep a certain distance to the other vehicles*, *do not go off road*, *do not crash*, etc.

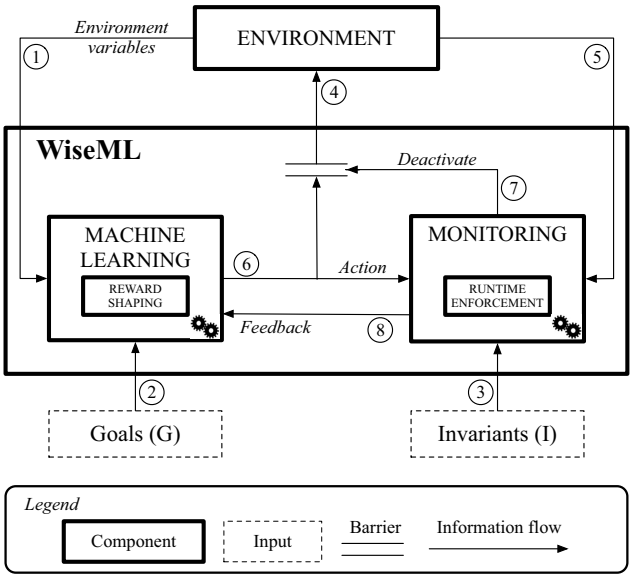

Figure 1: Overview of WiseML.

Machine learning. The reinforcement learning agent observes the state of the environment to detect how it reacts to the performed stimuli, i.e., the chosen actions (1). We envision the reinforcement learning agent to use a *Reward Shaping* component, that issues rewards to the agent to drive it towards the goals to be achieved, penalising it for acting bad and rewarding it for acting well. Based on the rewards it collects, it learns a policy, i.e. a function that maps the observations from the environment to actions to be performed on it. The reward function can also be affected by the feedback received by the monitor component, in case the selected action caused a violation of any invariant (8). In this sense, the monitor plays the role of a teacher for the learning algorithm. Eventually, the agent will learn a policy that maximises the cumulative reward by *trial and error* with the environment. Once an action is selected, it is then performed on the environment (4).

Monitoring. It aims at detecting whether the actions chosen by the machine learning component (6) are going to cause a violation of any invariant. In this sense, the monitoring algorithm should be *predictive*, i.e., it should detect violations of invariants before they will occur. The monitoring component relies on the *Runtime Enforcer* to ensure that the behaviour of the RL agent is compliant with certain properties, i.e., the invariants of the system. The Runtime Enforcer acts on the running system by allowing and forbidding actions of being executed.

The monitor evaluates the effect of actions on an abstract representation of the system, which is maintained updated using the information detected using the input variables (5). If the evaluation detects a violation, the monitoring component prevents the action for being executed (7) and sends feedback to the machine learning component (8). This feedback will be then integrated into the reward function by the reward shaping component. If no violation has been detected, then no barrier is activated (7) and the action is performed.

For example, referring to the automotive domain, functional safety standards such as the ISO26262 can be used as a guide for the definition of the monitored properties as in the approach of Heffernan et al. [7]. The usage of approaches that perform predictive runtime verification of timed properties may also be investigated [8, 9]. For instance, the approach in [9] exploits the structure of the property to predict faults before they will actually materialize. The verification monitor might also provide additional information such as the minimum (maximum) time when the property can be violated (satisfied) in the future.

4 MACHINE LEARNING

Reinforcement learning (RL) is an area of machine learning that deals with decision-making. An RL agent interacts with the environment by performing actions and it receives a reward. The agent will learn to choose actions that maximize the cumulative reward.

In model-based reinforcement learning, the RL agent computes an optimal policy on a model of the world, usually formalized as a Markov Decision Process (MDP) extended with the reward information associated to the transitions from one state to another.

The RL agent is also able to operate in a completely unknown environment, i.e., it is able to learn the best strategy to be employed when the only way to collect information about the environment is by interacting with it. Basically, the RL agent will learn a policy without knowing a priori a model of the environment, in this case, we refer to model-free reinforcement learning.

In our approach we will use a combination of *model-free* and *model-based* RL. First, we formalize the goals in terms of reward functions at *design-time*. Then, at *run-time*, the monitor can interfere with the reward function in case a violation of the invariants is detected. This mechanism is called *reward shaping* and it steers the RL agent to perform actions that will not trigger the monitor in the future, guiding it towards its goals [10].

5 MONITORING

Our envisioned approach (WiseML) uses monitoring techniques at run-time to prevent violations of invariants that will be caused by the actions chosen by the machine learning engine. Since the goal of monitoring is to avoid the execution of actions that will cause violations, monitoring should be *predictive*. According the available knowledge of the system and or the environment, different monitoring approaches might be conceived and/or adopted.

Given the current model of the environment, the invariant that must be ensured and the action a to be executed, the predictive monitoring engine aims at detecting whether an action execution will cause a violation of an invariant. In the case of no model of the environment is available, the predictive monitor can only make predictions on the structure of the property, on its current partial satisfaction, and on the distance, in terms of actions, to be performed in order to have a failure [9].

In general, the selection and design of predictive monitors open interesting challenges. These include the definition of appropriate semantics that consider whether a property *will* be satisfied or violated, the probability and distance from the potential failure, as well as whether it is possible to control the system and its environment in a way that satisfy the properties of interest (or avoid the failure). Since properties satisfaction must be verified at run-time, semantics must be defined taming the complexity of the corresponding verification algorithms.

Solutions for these problems may exploit multi-valued semantics [11]. These semantics are usually employed since two-valued semantics cannot be used to monitor all properties, such as liveness properties, and the satisfaction of these properties rely on how the system *will* behave. An example of these semantics is LTL_3. The semantics of LTL_3 is defined as follows: 1) satisfied; 2) violated; and 3) inconclusive. The same authors also extend LTL_3 with four-valued semantics: 1) satisfies the property, 2) violates the property, 3) will presumably violate the property, or 4) will presumably conform to the property in the future, once the system has stabilized.

Another aspect to be considered in the selection of the monitor is the type of invariants that should be ensured. Invariants describe guarantees that the system must ensure. These may include properties that predicate on explicit time as well as branching or linear notions of time. Examples of invariants that use a linear notion of time and implicit and explicit time are “if the left turn signal of the vehicle is on, it must eventually turn left" and “if the left turn signal of the vehicle is on, it must turn left within 10 seconds", respectively. Depending on the invariants to be analysed, different

run-time monitors can be considered and used. It is important to note that these monitors cannot predict exactly what will be the behaviour of the system and if specific failures can be prevented. However, they should be able to inform the system about the possibility and probability of a failure. The output of our monitors may be exploited to enhance the system with run-time mechanisms to avoid failures. Triggered by our monitors, these mechanisms may be able to act before the failure and take all the possible actions to prevent failures.

We plan to automatically generate a predictive monitor from the invariants using methods such as PREDIMO [12], a novel approach where the properties to be monitored are specified in terms of scenarios. This approach automatically synthesises a monitor by exclusively exploiting the structure of the property and a partial knowledge of the behaviour of the environment. By taking into account the actual status and also the foreseen possible evolutions of both system and environment in the near future, the generated monitors provide an estimate of a potential incoming failure, in terms of the distance to the failure and the degree of controllability of the system. This enables the definition of run-time mechanisms that, e.g. by avoiding specific actions or forcing other ones, might prevent failures in the near future.

We will also evaluate the possibility of realizing monitors based on game theory, like the approaches in [13–15].

6 DISCUSSION

WiseML is based on the idea to not program all the behaviours and adaptation that the system should perform at design-time but instead set up goals to achieve and train the system to achieve them. Reinforcement learning is a framework that fits perfectly with our needs of learning and adapting in order to reach the desired goal. However, as pointed out in Section 2, using machine learning techniques instead of traditional software imposes some challenges. Decisions are not driven anymore by software written by programmers following the requirements but on data and the reward signal. With runtime monitoring, we envision to create a *satefy envelope* around the machine learning system. The monitor will prevent the RL agent to choose an action that violates important invariants and it will train the agent to perform better in the future.

The correct selection of the reinforcement learning algorithm and of the monitoring engine is crucial for obtaining an implementation of WiseML that behaves as expected. The machine learning algorithm should be chosen depending on whether there is full or partial observability about the environment in which the system is deployed. Most likely the RL agent will have some initial knowledge about the environment and it will be trained so that it can learn about the environment as it explores it, in an online fashion. The monitoring engine should be chosen considering the type of invariants that need to be ensured as well as the desired effectiveness and velocity in reacting to changes. It must be noticed that machine learning and monitoring components cannot be chosen in isolation, but their composed behaviour should be considered in their selection and set-up.

Having an effective monitor may reduce performance. Indeed a deeper analysis of invariants' violations may cause a performance overhead and at the end invalidate verification results. This may occur when predictive monitoring is too slow and, before results are obtained, the model of the environment already changed drastically since other agents performed actions. For example, while the monitoring verifies whether the action "crosses the intersection within 2 seconds" can be performed (by also evaluating the effect of other agents actions) the semaphore may turn red and invalidate the obtained results. On the other hand, a less effective monitor may return not accurate results. For example, a monitor that checks whether a pedestrian is not crossing the road should also consider the volumes of the objects carried by the pedestrian. This check causes a performance overhead.

7 CONCLUSIONS

This paper envisions a new approach named WiseML. This paper envisions a new approach named that aims at creating systems that, on one hand are able to learn and adapt their behavior based on changes that occur in the environment, on the other are able to ensure that adaptation does not cause invariants violation. We discussed some challenges posed by reinforcement learning and how combining it with runtime monitoring might solve some of these challenges. We broadly discussed some possible solutions for the proposed envisioned approach that may use reinforcement learning as mechanisms to enable adaptation and predictive monitoring as instrument to detect invariants violations.

REFERENCES

[1] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 4, no. 2, p. 14, 2009.

[2] R. de Lemos et al., "Software Engineering for Self-Adaptive Systems: A Second Research Roadmap," in *Software Engineering for Self-Adaptive Systems II*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–32.

[3] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[4] P. Koopman and M. Wagner, "Challenges in Autonomous Vehicle Testing and Validation," *SAE International Journal of Transportation Safety*, vol. 4, no. 1, pp. 15–24, Apr. 2016.

[5] N. N. Taleb, *The Black Swan: The Impact of the Highly Improbable*. Random House Group, ISBN: 1400063515, 2007.

[6] T. Everitt, V. Krakovna, L. Orseau, M. Hutter, and S. Legg, "Reinforcement learning with a corrupted reward channel," *arXiv preprint arXiv:1705.08417*, 2017.

[7] D. Heffernan, C. MacNamee, and P. Fogarty, "Runtime verification monitoring for automotive embedded systems using the ISO 26262 functional safety standard as a guide for the definition of the monitored properties." *IET Software ()*, vol. 8, no. 5, pp. 193–203, 2014.

[8] S. Pinisetty, T. Jéron, S. Tripakis, Y. Falcone, H. Marchand, and V. Preoteasa, "Predictive runtime verification of timed properties," *Journal of Systems and Software*, vol. 132, pp. 353–365, 2017.

[9] P. Zhang, P. Pelliccione, H. Leung, and X. Li, "Automatic generation of predictive monitors from scenario-based specifications," *Information and Software Technology*, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584917301507

[10] S. Devlin and D. Kudenko, "Dynamic potential-based reward shaping," in *Proc. of International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, 2012, pp. 433–440.

[11] A. Bauer, M. Leucker, and C. Schallhart, "Runtime Verification for LTL and TLTL," *ACM Trans. Softw. Eng. Meth.*, vol. 20, no. 4, 2011.

[12] P. Zhang, P. Pelliccione, H. Leung, and X. Li, "Automatic generation of predictive monitors from scenario-based specifications," *Information and Software Technology*, 2018.

[13] L. de Alfaro and M. Stoelinga, "Interfaces: A game-theoretic framework for reasoning about component-based systems," *Electronic Notes in Theoretical Computer Science*, vol. 97, no. Supplement C, pp. 3 – 23, 2004, proc. of FOCLASA 2003.

[14] D. Harel and I. Segall, "Synthesis from scenario-based specifications," *J. Comput. Syst. Sci.*, vol. 78, no. 3, pp. 970–980, May 2012.

[15] R. Ehlers and B. Finkbeiner, "Monitoring realizability," in *Proc. of International Conference on Runtime Verification*. Springer-Verlag, 2012, pp. 427–441.