Keeping Smartness under Control

Anonymous Author(s)

ABSTRACT

Today, software is becoming predominant in every system. Software is also a means to make systems smart, and continuously improvable. Example of systems made smart by software are self-driving cars, self-flying airplanes, self-managing telecom networks, and smart factories. Smart systems will be increasingly characterized by continuous and rapid change, and will employ continuous evolution and continuous learning from observations on their own behavior. Machine learning is a promising technique, however, its use in safety-critical domains poses some challenges. In fact, it is difficult to certify that machine learning software will now violate important properties and when dealing with safety-critical systems, the adaptation of systems must be certified as safe before it can be applied. In this paper we propose an approach that combines machine learning approaches with runtime monitoring that guarantees the preservation of important safety-critical requirements.

KEYWORDS

Autonomous systems, Safety-critical, Reinforcement learning, Machine learning, Runtime verification

ACM Reference format:
Anonymous Author(s). 2018. Keeping Smartness under Control. In Proceedings of 40th International Conference on Software Engineering, Gothenburg, Sweden, May 27–June 3, 2018 (ICSE 2018), 5 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Modern software systems need to increasingly operate in dynamic, uncontrollable, and partially known environments. These systems have to deal with various dimensions of uncertainty that cannot be completely predicted at design-time. Sources of uncertainty could be the environment around the system, the availability of the resources that the system can access at a given time or the difficulty of predicting the other systems behaviour [1–3].

Modern software systems are increasingly smart and autonomous. On one side they need to become self-adaptable systems, i.e. systems that are able to adapt their behavior at run-time without human intervention [4–6] in response to changes in the environment or in their internal state. Self-adaptive systems implement some sort of feedback loop that drives their adaptations [7]. This basic mechanism for adaptation has been applied for years in control engineering; it consists of four main activities: collect, analyze, decide, and act. A well-known reference model for describing the adaptation processes is the MAPE-K loop (consisting of the parts Model, Analyze, Plan, Execute, and the Knowledge Base) [8]. Furthermore, often systems collect data from the environment, learn from them, and, consequently, continuously improve.

On the other side, smart and autonomous systems will need to support continuous evolution of software, even when they are operating in the field, e.g., in automotive, when vehicles are already on the road, through Continuous integration and Continuous Deployment (CI&D) practices. CI&D practices promise to shorten integration, delivery, and feedback cycles [9]. These techniques have been applied by pure software companies, such as Facebook, and now many other domains, like automotive [10] and robotics[1], have a strong motivation to embrace continuous integration and delivery practices to yield improvements in flexibility and cycle time despite the challenges. Continuous evolution of a system should additionally exploit the knowledge collected by other systems. In this sense, assuming that mistakes are inevitable also for autonomous systems, not every system has to make the same mistake to learn from it. For instance in the automotive domain, once the software has been fixed, all other vehicles will be updated based on this knowledge and no other vehicle will do the same mistake again. As Tesla says: "as more real-world miles accumulate and the software logic accounts for increasingly rare events, the probability of injury will keep decreasing"[2].

Self-adaptation and online learning are promising concepts but are still far from practical application in safety-critical domains. Collecting data at runtime, self-adapt, and continuously evolve are a fascinating concepts. When dealing with safety-critical systems, the adaptation of systems must be certified as safe before it can be applied. However, with the intensive use of machine learning techniques it is hard to test the software and be sure that it will act always as intended.

In this paper we propose an approach that combines machine learning approaches with runtime monitoring that guarantees the preservation of important safety-critical requirements. On one side, systems will be able to adapt and evolve themselves at runtime through the use of machine learning techniques. On the other side, runtime monitoring will continuously check that the actions suggested by

ICSE 2018, Gothenburg, Sweden
2018 ACM. 978-x-xxxx-xxxx-x/YY/MM. . . $15.00
DOI: 10.1145/nnnnnnn.nnnnnnn

[1] e.g., http://wiki.ros.org/CIs and http://rosin-project.eu/

[2] https://www.tesla.com/blog/tragic-loss

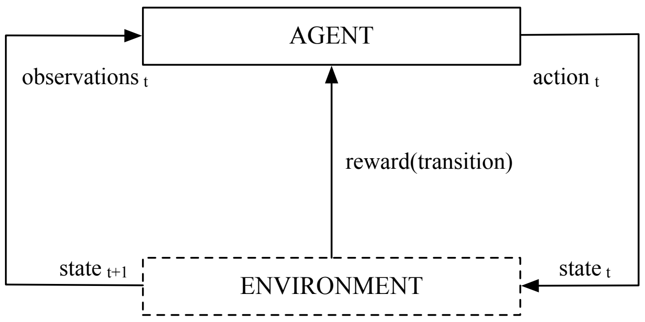

Figure 1: Reinforcement Learning framework

reinforcement learning will not violate the identified safety-critical requirements, which are specified in terms of invariant properties.

Moreover, the outcome of the runtime monitoring is also exploited to train the machine learning algorithms used to make decisions; in this way, they can continuously learn and suggest better actions in the future.

The paper is structured as follows: Section 2 introduces the machine learning techniques we use in our approach and Section 3 explains the challenges in using these techniques. Section 4 presents the approach we propose in this paper. Section 6 concludes with final remarks and research directions for the future.

2 REINFORCEMENT LEARNING

Typically to design the solution of a problem we start from examples of how the system should react based on its inputs following its requirements. With machine learning we start from the data, we analyze it and capture the properties that are necessary to achieve a certain goal. We first look at the data and define possible models to be used, we learn the parameters and structure of the models from data and finally, we use the models to make predictions and decisions.

One mechanism to perform decision-making using machine learning is Reinforcement Learning, which includes topics such as goal setting, planning, and perception. It involves an autonomous agent willing to navigate an uncertain environment with the goal of maximizing a numerical reward. These rewards are received by the agents from the environment, the agent keeps track of them and use them to generate the next actions to be performed [11].

Figure 1 shows a general framework for reinforcement learning. An agent interacts with its environment in consecutive time steps. At each time step `t` the agent receives observations from the current state of the environment. It has to choose a suitable action that will cause the environment to move to a new state. A reward associated with this transition is determined and sent back to the agent.

For example, given a set of actions (such as turn-left and turn-right), at each step the agent applies a certain policy to its current state and to the observations it gathers from the environment to generate the next suitable action and move to the next state. While in other techniques such as supervised learning the outcome is based on historical examples and there is no concept of current state, in reinforcement learning it is all about rewards and an action resolves in a change of the state.

One of the problems in reinforcement learning is to find a policy with the right tradeoff between exploration and exploitation [12]. We want our agent to explore the environment and seek the best path to reach the goal and this might involve taking sub-optimal actions on the way. At the same time, we want our agent to learn from the environment and hence perform actions that are already known to be good. Finding this balance between exploration and exploitation is on of the keys to a good policy.

3 CHALLENGES OF REINFORCEMENT LEARNING

Reinforcement Learning (RL) techniques require a set of training data which has to be independent of the validation data to avoid overfitting. One main problem with ML methods is that they are optimized for average cost function and they do not guarantee for corner cases. Challenges in this area are comprised of 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. This is one of the hot research areas at the moment and researchers are investigating different ways to visualize and understand the logic behind the learned neural networks in solving various tasks. Brute force testing is a widely used technique to validate the network resulting in an expensive and not always best validation method.

Furthermore, because the neural network learns the rules from a training set, if certain data is missing or wrongly correlated in the training data, this can cause the network to fail to 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 [13]. Hence, it is hard to detect and isolate bugs where the behavior is not expressed with traditional lines of codes but entrusted to a neural network. The network would need to be retrained with the risk to “unlearn" correct behaviors.

Moreover, an incorrect specification of the reward function can cause unexpected behaviours in the agent. One of the problems is reward hacking [14], meaning that the agent exploits the reward function and manages to get a high reward without achieving the designer’s intentions, but instead optimizing towards the rewards function that is indeed not exactly representing the designer’s intentions. For example, in a cleaning robot if the reward function gives 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 reward.

✎ todo describe some of the major challenges for guaranteeing safety in autonomous vehicles as pointed out also by Koopman [15] and Schmittner [16].

✎ todo Add examples, like: https://www.computerworld.com/article/3087328/emerging-technology/google-concerned-about-curious-but-destructive-cleaning-robots-that-hack-reward-systems.html http://www.wired.co.uk/article/google-ai-five-problems-robots, https://blog.openai.com/concrete-ai-safety-problems/

4 VALIDATION PROCESS FOR AUTONOMOUS SYSTEMS IS NOT CLEAR

A common way to assess safety in autonomous vehicles is through extensive testing, by test-driving the vehicles, and evaluating the vehicles performance. The more the vehicles drive in autonomous mode, the more experience they gather, and this will result in continuously improving system. Companies like Waymo[3] advertises that their fleet of autonomous vehicles has driven 2.5 million miles accumulating the equivalent of 400 years human driving experience. Although such numbers seam impressive, in order to demonstrate their reliability, autonomous vehicle might need to drive for hundreds of millions or in some cases billions of miles [17]. New methods to establish the safety of autonomous vehicles are needed. Big data analysis becomes very important in this regard, statistical signal processing and ML methods have to be developed to analyze the large amounts of data that is collected from the test vehicles or customer vehicles. Examples of such analysis includes: (i) the detection of the use cases for which the sensors or the complete system provide poor performance, (ii) anomaly detection in the sensor data [18], and (iii) the creation of realistic simulation frameworks through the use of sensor models where the logged sensor data are used to improve the quality of the simulations performed in a cluster of computers. To be able to do so, sensor comparison frameworks [19] are required to be able to compare the data obtained from two different sensors, where one of the sensors could have, for example, significantly higher accuracy and can be used as a reference sensor.

Furthermore, the use of probabilistic models (as in the object detection) and stochastic algorithms (as in planning) poses new challenges in the validation process. Having probabilistic system passing the test once does not guarantee that the same test will succeed every time. Testing becomes difficult for two reasons. The first reason is that due to the non-repeatability of such algorithms it can be difficult to exercise a particular corner case. The second reason is that it is difficult to evaluate whether a result is correct or not in the case of there are multiple correct behaviors for each test case.

[3]https://waymo.com

5 CONTROLLING REINFORCEMENT LEARNING DECISIONS AT RUNTIME

We present here a method that combines the flexibility and adaptability of machine learning algorithms with the more formal and traditional invariants preservation methods.

In safety critical application it is mandatory the "freedom from unacceptable risk of physical injury or of damage to the health of people, either directly, or indirectly as a result of damage to property or to the environment".

We propose a method for delegating part of the decision-making process of an Autonomous System to fully machine-learned algorithms while still preserving system's invariants.

Autonomous Systems require measurements and interaction with the environment. They perceive external information through sensors and effect the environment executing actions through actuators (in pure software system these are only inputs and outputs). The perception of the environment is a semantic representation of the raw data and it can be achieved with techniques such as sensor fusion, where information coming from multiple sensors are combined. The perception component performs an important process that enables the autonomous system to build a model of the external world and it is the source of observations for the decision-making component.

The execution component applies the actions to the external environment and it also communicates with the internal world model. The data transferred to the world model is used to keep track of the actions generated. This information can be used for simulation purposes but also to understand the effect that actions have produced to the environment and refining invariants generation process.

✎ todo Add simple figure describing the approach at high level

✎ todo Introduce and detail each part of the approach, identifying the challenges and how this can be realized

✎ todo discuss how the approach manages adaptation and evolution

5.1 Identifying invariants from high-level goals

Identifying from high-level goals operational requirements that should hold on components, parts of the system or operations is still an open problem. Most of the approaches that use specifications, such as formal methods, assume such operational requirements to be given. However, deriving "correct" operational requirements from high-level goals is challenging and is often delegated to error prone processes.

Letier and Lamwsveerde [20] propose an iterative approach that allows the derivation of operational requirements from high-level goals expressed in real-time linear temporal logic (RT-LTL). The approach is based on operationalisation patterns. Operationalization is a process that maps declarative property specifications to operational specifications satisfying them. The approach produces operational requirements in the form of pre-, post-, and trigger- conditions. The approach is guaranteed to be correct. Informally, here correct means that the conjunction of the operational requirements entails the RT-LTL specification. The approach is limited to a collection of goals and requirement templates provided by the authors. Moreover, the approach necessitates a fully refined goal model that requires specific expertise and is labour-intensive and error-prone.

The tool-supported framework proposed in [21, 22] combines model-checking and Inductive Logic Programming (ILP) to elaborate and refine operational requirements in the form of pre- and trigger- conditions that are correct and complete with respect to a set of system goals. System goals are in the form of LTL formulas. The approach works incrementally by refining an existing partial specification of operational requirements, which is verified with respect to the system goal. The verification is performed by using model checking, which returns a counter-example in the case of the considered property is not valid on the model. The counter-example is exploited to learn and refine the operational requirements. However, the approach does not support learning the operational requirements for an individual component of a system.

The work in [23] focuses on the service-oriented computing paradigm and starts from the observation that, when dealing with open-world software, it is unrealistic to assume the availability of the interface descriptions of third-party services. Then, this paper proposes an interesting approach to automatically generate behavioral interfaces of the partner services, by decomposing the requirements specification of a service composition.

In our approach we need to identify invariants from important safety-critical requirements.

5.2 Reinforcement learning for decision of autonomous systems

In the use of reinforcement learning algorithms, two aspects are extremely important: (i) training the algorithm with the right set of data, and (ii) specifying correctly the reward function. Referring to the first point in addition to select identify at the best we can proper training data, we support a continuous learning that involves data collected also from other systems. About the second point, we aim at providing methodologies to better engineering the reward function definition. We provide a methodology that permits to model the reward function is terms of state machines and to permit to close the gap between the designer informal goals and the reward signal. Moreover, our methodology supports the simulation and formal verification of the reward function through the use of the UPPAAL model checker[4]. Initial results in this direction might be found in [?][Patrizio: this should be anonymized].

[4]http://www.uppaal.org/

5.3 Monitoring as controller of reinforcement learning components

Runtime monitoring observes how the system is behaving and it detects if this behaviour is compliant with the system invariants.

The main idea of runtime verification based on monitors is to synthesize a monitor that can check whether the runtime behaviors satisfy or diverge from desired properties [24, 25]. Unfortunately, existing run-time monitoring approaches provide limited information to be exploited at run-time for early detecting and managing situations that most probably will lead to failures. There is the need of investigating new approaches to automatically generate predictive monitors able to predict violations that most probably will happen in the near future and to provide information that can be exploited to define strategies to prevent such failures. Our approach utilizes reinforcement learning and if the actions produced by this agent violates any safety-critical invariant the monitoring process prevents this action to be executed and tame the agent by sending feedback. This feedback has to processed and properly expressed in term of reward function. The reinforcement learning agent will get points for its correct actions and loose points for dangerous actions that also breaks some system's invariance. First works in the direction of predictive monitoring might be find in [26], where the authors propose three-valued semantics called 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. Further research is needed to investigate approaches to automatically generate predictive monitors able to predict violations that most probably will happen in the near future and to provide information that can be exploited to define strategies to prevent such failures or to prepare suitable reactions. Monitors should offer a multi-values and fine grained semantics enabling the definition of precise strategies to prevent possible failures. This permits the evaluation of specified properties taking into account the current status of the system and also possible evolutions of them in the near future.

In our approach monitoring has a double purpose:

(1) it monitors the actions taken by the decision-making component preventing those that violate system's invariants from being executed. The monitor should detect the hazardous actions before they are sent to execution. For instance, 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. [27].
(2) it trains the machine-learning algorithms by sending feedback for each produced action so that it can

learn. For example, an agent that models that behaviour of an autonomous vehicle would get a positive reward for staying in its lane, following the road, keeping a safe distance with the vehicle ahead etc. In the same way, it could loose points (hence getting a negative reward) in the case of wrong actions such as, for example, speeding or performing dangerous maneuvers.

5.4 Collecting feedback for enabling learning

Learning is based on data that are collected from both the external environment and the internal state of the system. Collected data are then analyzed in order to build a model of the environment that is programmatically accessible and continuously updated so that the autonomous system knows at any moment the context it is in.

In the case of autonomous vehicles, this component can contain a dynamic map of the world around the car that with new knowledge being incrementally added at any time. Currently, there are efforts in building a standardized representation of the world in the so-called “Local Dynamic Maps" [28]. It can also contain sophisticated knowledge representations such as KnowRob [29], an information ontology for autonomous robots.

Contextual requirements must be collected at run-time, as for instance done in Acon [30] that keeps an up to date knowledge about such context using machine learning algorithms.

6 CONCLUSIONS

REFERENCES

[1] N. Esfahani and S. Malek, Uncertainty in Self-Adaptive Software Systems. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 214–238. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-35813-5_9
[2] M. Autili, V. Cortellessa, D. Di Ruscio, P. Inverardi, P. Pelliccione, and M. Tivoli, “Eagle: Engineering software in the ubiquitous globe by leveraging uncertainty,” in Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, ser. ESEC/FSE ’11. New York, NY, USA: ACM, 2011, pp. 488–491. [Online]. Available: http://doi.acm.org/10.1145/2025113.2025199
[3] D. Garlan, “Software engineering in an uncertain world,” in Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research, ser. FoSER ’10. New York, NY, USA: ACM, 2010, pp. 125–128. [Online]. Available: http://doi.acm.org/10.1145/1882362.1882389
[4] 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.
[5] B. H. Cheng, H. Giese, P. Inverardi, J. Magee, R. de Lemos, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic et al., “Software engineering for self-adaptive systems: A research road map,” in Dagstuhl Seminar Proceedings. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2008.
[6] 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.
[7] Y. Brun, G. D. M. Serugendo, C. Gacek, H. Giese, H. M. Kienle, M. Litoiu, H. A. Müller, M. Pezzè, and M. Shaw, “Engineering self-adaptive systems through feedback loops.” Software engineering for self-adaptive systems, vol. 5525, pp. 48–70, 2009.
[8] J. O. Kephart and D. M. Chess, “The vision of autonomic computing,” Computer, vol. 36, no. 1, pp. 41–50, 2003.
[9] D. Sthl and J. Bosch, “Modeling continuous integration practice differences in industry software development,” J. Syst. Softw., vol. 87, pp. 48–59, Jan. 2014. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2013.08.032
[10] E. Knauss, P. Pelliccione, R. Heldal, M. gren, S. Hellman, and D. Maniette, “Continuous integration beyond the team: A tooling perspective on challenges in the automotive industry,” in Proceedings of ESEM ’16. ACM, 2016.
[11] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction. MIT press Cambridge, 1998, vol. 1, no. 1.
[12] M. Coggan, “Exploration and exploitation in reinforcement learning,” Research supervised by Prof Doina Precup, 2004.
[13] N. T. Nassim, “The black swan: the impact of the highly improbable,” Random House, 2007.
[14] T. Everitt, V. Krakovna, L. Orseau, M. Hutter, and S. Legg, “Reinforcement learning with a corrupted reward channel,” arXiv preprint arXiv:1705.08417, 2017.
[15] 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.
[16] C. Schmittner, Z. Ma, and T. Gruber, “Standardization challenges for safety and security of connected, automated and intelligent vehicles,” 2014 International Conference on Connected Vehicles and Expo (ICCVE), pp. 941–942, 2014.
[17] N. Kalra and S. M. Paddock, “Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?” Transportation Research Part A: Policy and Practice, vol. 94, pp. 182–193, Dec. 2016.
[18] A. Tashvir, J. Sjöberg, and N. Mohammadiha, “Sensor error prediction and anomaly detection using neural networks,” in The First Swedish Symposium on Deep Learning (SSDL), 2017.
[19] J. Florbäck, L. Tornberg, and N. Mohammadiha, “Offline object matching and evaluation process for verification of autonomous driving,” in Int. Conference on Intelligent Transportation Systems (ITSC), Nov. 2016, pp. 107–112.
[20] E. Letier and A. van Lamsweerde, “Deriving operational software specifications from system goals,” in Proceedings of SIGSOFT ’02/FSE-10. New York, NY, USA: ACM, 2002, pp. 119–128. [Online]. Available: http://doi.acm.org/10.1145/587051.587070
[21] D. Alrajeh, J. Kramer, A. Russo, and S. Uchitel, “Learning operational requirements from goal models,” in Proceedings of ICSE ’09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 265–275. [Online]. Available: http://dx.doi.org/10.1109/ICSE.2009.5070527
[22] D. Alrajeh, O. Ray, A. Russo, and S. Uchitel, “Using abduction and induction for operational requirements elaboration,” Journal of Applied Logic, vol. 7, no. 3, pp. 275 – 288, 2009, special Issue: Abduction and Induction in Artificial Intelligence. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570868308000633
[23] D. Bianculli, D. Giannakopoulou, and C. S. Păsăreanu, “Interface decomposition for service compositions,” in Proceedings of ICSE ’11. New York, NY, USA: ACM, 2011, pp. 501–510. [Online]. Available: http://doi.acm.org/10.1145/1985793.1985862
[24] N. Delgado, A. Q. Gates, and S. Roach, “A taxonomy and catalog of runtime software-fault monitoring tools,” IEEE Trans. Softw. Eng., vol. 30, no. 12, pp. 859–872, Dec. 2004. [Online]. Available: http://dx.doi.org/10.1109/TSE.2004.91
[25] M. Leucker and C. Schallhart, “A brief account of runtime verification,” The Jour. of Logic and Algebraic Progr., vol. 78, no. 5, pp. 293 – 303, 2009, the 1st Workshop on Formal Languages and Analysis of Contract-Oriented Software (FLACOS?07).
[26] A. Bauer, M. Leucker, and C. Schallhart, “Runtime verification for ltl and tltl,” ACM Trans. Softw. Eng. Meth., vol. 20, no. 4, 2011. [Online]. Available: http://doi.acm.org/10.1145/2000799.2000800
[27] 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.
[28] H. Shimada, A. Yamaguchi, H. Takada, K. Sato et al., “Implementation and evaluation of local dynamic map in safety driving systems,” Journal of Transportation Technologies, vol. 5, no. 02, p. 102, 2015.

[29] M. Tenorth and M. Beetz, "Representations for robot knowledge in the KnowRob framework," Artificial Intelligence, Jun. 2015.

[30] A. Knauss, D. Damian, X. Franch, A. Rook, H. A. Müller, and A. Thomo, "ACon: A learning-based approach to deal with uncertainty in contextual requirements at runtime," Information and Software Technology, vol. 70, pp. 85–99, 2016.