Towards Better Adaptive Systems by Combining MAPE, Control Theory, and Machine Learning

Danny Weyns
KU Leuven, Belgium
Linnaeus University, Sweden
danny.weyns@kuleuven.be

Bradley Schmerl
Carnegie Mellon University
Pittsburgh, USA
schmerl@cs.cmu.edu

Masako Kishida
National Institute of Informatics
Tokyo, Japan
kishida@nii.ac.jp

Alberto Leva
Politecnico di Milano
Milan, Italy
alberto.leva@polimi.it

Marin Litoiu
York University
York, Canada
mlitoiu@yorku.ca

Necmiye Ozay
University of Michigan
Ann Arbor, MI, USA
necmiye@umich.edu

Colin Paterson
University of York
York, United Kingdom
colin.paterson@york.ac.uk

Kenji Tei
Waseda University
Tokyo, Japan
ktei@aoni.waseda.jp

arXiv:2103.10847v1 [cs.SE] 19 Mar 2021

***Abstract*—Two established approaches to engineer adaptive systems are architecture-based adaptation that uses a Monitor-Analysis-Planning-Executing (MAPE) loop that reasons over architectural models (aka Knowledge) to make adaptation decisions, and control-based adaptation that relies on principles of control theory (CT) to realize adaptation. Recently, we also observe a rapidly growing interest in applying machine learning (ML) to support different adaptation mechanisms. While MAPE and CT have particular characteristics and strengths to be applied independently, in this paper, we are concerned with the question of how these approaches are related with one another and whether combining them and supporting them with ML can produce better adaptive systems. We motivate the combined use of different adaptation approaches using a scenario of a cloud-based enterprise system and illustrate the analysis when combining the different approaches. To conclude, we offer a set of open questions for further research in this interesting area.**

***Index Terms*—Self-adaptive systems, MAPE, control theory, machine learning, Cloud enterprise system.**

I. Introduction

Designing self-adaptive systems with some level of autonomy has been studied for over two decades [1]–[6]. A number of different approaches have been used to engineer such systems. Two established approaches are (i) architecture-based adaptation that relies on Monitor-Analysis-Planning-Executing (MAPE) components that reason over architectural models (aka Knowledge) to make adaptation decisions [7]–[11], and (ii) control-based adaptation that relies on principles of control theory (CT) to realize adaptation of a target system [12]–[14]. Recently, we also observe a rapid growing interest in applying machine learning (ML) to support different adaptation mechanisms [15], for instance to use classifiers to reduce large adaptation spaces [16], learn a model of the system with MAPE [17], or learn adaptation rules with CT [18].

This paper is concerned with the following question:

How can MAPE and CT be combined with and supported by ML techniques to produce better adaptive systems?

In answer to this question, this paper contributes insights into: (1) the characteristics and strengths of both CT and MAPE, and how ML can support adaptation techniques, (2) how CT and MAPE can be combined with and supported by ML, and (3) a number of open topics for further research in this area. This paper is an outcome of a working group of the 3^{rd} Shonan Meeting on Controlled Adaptation of Self-adaptive Systems.[1]

The remainder of this paper is structured as follows. In Section II we summarise a selection of related work. Section III introduces the main concepts and outlines the scope of this work. Section IV highlights strengths of CT and MAPE and looks into support of ML to adaptation techniques. Section V zooms in on combining CT and MAPE and supporting them with ML in an example case. Finally, Section VI wraps up and outlines a set of open questions for further research.

II. Related Work

We summarise a selection of related work, starting with approaches that combine MAPE with CT. Then we look at work that combines MAPE or CT with ML. We conclude by motivating the research question posed in this paper.

Combining MAPE with CT. DYNAMICO is a conceptual model that considers three levels of dynamics in adaptive systems that map to three interacting feedback loops [19]. A first feedback loop monitors requirements (adaptation goals) to ensure their fulfilment. A second feedback loop manages context information preserving context information relevant to adaptation. Finally, a third feedback loop controls the target system according to control objectives, while taking into account the context. The feedback loops can be realised using different adaptation mechanisms, e.g., MAPE for the first and second and CT for the third feedback loop. The idea of combining MAPE-based discrete decision-makers/planners with continuous low-level controllers has also been proposed in the context of symbolic control, with applications in robotics [20], self-driving cars and driver-assist systems [21], [22]. In these approaches, the higher levels guarantee satisfaction of user

[1]The 3^{rd} Shonan Meeting on Controlled Adaptation of Self-adaptive Systems (CASaS). Report: https://shonan.nii.ac.jp/docs/No.153.pdf

goals (often specified in temporal logics) under discrete external factors, while lower levels ensure robust tracking of the goals set by the higher levels under disturbances.

Supporting MAPE with ML. As the complexity of self-adaptive software has steadily grown over time, the opportunities to exploit ML to support MAPE have also increased. Just as ML has found substantial success in recent years in the perception pipeline of automotive vehicles, such techniques have been deployed to monitor computing systems at runtime to derive meaningful measures from data gathered in complex environments [23] and to detect faults [24]. During the analysis phase, ML has been used when the space of possible adaptations grows too large to be handled with traditional techniques [25] or where patterns may be extracted from large data sets (e.g., network traffic [26]). Finally, we see ML used in the planning stage where selecting a policy for adaptation becomes difficult due to the size of the adaptation space. Here ML can pre-filter adaptation options and present a set of Pareto optimal solutions from which the MAPE solution can choose an appropriate action [27], or evolutionary search can be used to change control parameters to more optimal settings [28].

Supporting CT with ML. ML has been used in control theoretical solutions for a long time. One established area is the use of ML for model and system identification, see for instance [29]–[32]. Recent examples that use ML techniques to derive a linear model of a system are [33], [34], while [35] uses recurrent neural networks to capture the behaviour of non-linear systems. More generally, neural networks have been used for a long time to solve highly non-linear control problems [36], [37]. Recently, reinforcement learning has gained increasing attention to deal with control problems where the state space is large and unknown a priori [38], or the number of controller parameters that need to be tuned is large [39].

Motivation for Research Question. While some efforts show benefits of combining MAPE with CT on the one hand and the potential of exploiting ML to support the adaptation mechanisms on the other hand, further research is required on how these approaches can be combined to build better adaptive systems. In particular, further research is required to understand: (1) the characteristics and strengths of CT and MAPE and the use of ML for adaptation, (2) how CT and MAPE can be combined with and supported by ML, (3) how we can consolidate this knowledge into reusable assets.

III. Concepts and Scope

We briefly describe the main concepts and the scope of the work presented in this paper. We start with the two adaptation techniques we focus on: CT and MAPE, and conclude with ML that we aim to use in support for adaptation.

A. Control Theory

In this paper, for CT we essentially mean classical control structures – single-loop or at most feed forward compensation, cascade controls – made of individually simple linear time-invariant blocks such as Proportional-Integral-Derivative controllers (PID), running at a fixed sampling rate or triggered by events. These structures have demonstrated to be powerful at keeping some variable(s) at prescribed set points or within prescribed ranges in the face of disturbances, provided that the relationship between the control signal and the controlled variable is not too complex and does not vary too much over time [40]. In large applications, the "classical CT layer" is often extended with a higher "advanced layer" realised as Model Predictive Control (MPC). This upper layer feeds commands to the lower one, adapting its behaviour when the conditions of the system (plant) require such an action.

This two-layer control scheme fits domains like process control very well, where the plant is ruled by clearly defined physical laws, but in general does not seem to fit equally well adaptive software. Among the reasons are that similar laws are not so easy to define for software systems, that objectives can be very heterogeneous, and time scales for decision-making may not fit well [12], [14]. Moreover, adaptive software systems often require complex types of adaptations, e.g., changing the structure of a managed system.

Key insights: CT's strength lies in keeping variables at prescribed set points or within prescribed ranges, regardless of disturbances. Adding an MPC control layer atop a classical control layer fits well for process control, but does not seem to map naturally to software systems. We conjecture that in adaptive software the two control schemes can be re-used, where the upper layer may be realised using MAPE.

B. MAPE

MAPE concerns techniques usually covered by the software engineering for self-adaptive systems community. MAPE is an acronym introduced by Kephart and Chess [41] referring to the essential activities (functions) that should be covered by any systems with adaptive capabilities: Monitoring, Analysis, Planning, and Execution. The aim of MAPE is usually to adapt software – its configuration, structure, behaviour, etc. – rather than physical phenomena. While all forms of control are likely to have one or more of these activities (or are known in other guises as Sense-Plan-Act, for example), we use MAPE as a label in this paper to capture approaches to adapt software and the techniques that have been developed to handle this.

The MAPE activities are centered around Knowledge models that typically include various forms of runtime models [42], such as software architecture models of the managed system and environment, goal models, Markov networks that allow predicting qualities of different system configurations, etc. These models usually focus on the properties that one wishes to maintain the software [43]–[45]. Monitoring is used to update these models with data about the current state of the software, the environment in which it is running, and the goals of the software. Analysis typically involves taking the current state and determining if the properties align with user or business goals. Often, analysis is focused on a multi-dimensional space that trades off different quality attributes of the software – for example, balancing performance, reliability, security, or cost. Tools that can be used here include runtime simulations,

model checking, architecture analysis, etc. Analysis can be expressed as thresholds or constraints that the software should achieve or maintain. E.g., we might desire a response time less than two seconds; we may aim to minimise operational costs. Planning then decides the set of steps required to change the software to satisfy the properties. In simple condition-based MAPE systems, planning would just be choosing which action to take based on some condition. Other approaches might choose some organised sequence or tree of actions that maximise some utility function. Sometimes more advanced techniques are required, possibly based on AI, to generate plans. Finally, execution takes these actions and manages their enactment on the actual software system. This may involve non-trivial synchronisation with the system that is adapted.

A pioneering work that applies different levels of control is IBM's Autonomic Computing Reference Architecture [46], see also [47]. Yet this work considers hierarchies of MAPE loops rather than integrations of MAPE and CT-based solutions.

A key aspect for the industrial adoption of MAPE is frameworks that offer interfaces for monitoring and safe adaptations of the underlying system. A typical example is OSGi [48] that offers a dynamic component model for Java.

Key insights: MAPE addresses adaptation of software rather than physical properties or resources; MAPE has a global perspective on the system (or subsystems being managed); MAPE deals with trading off requirements/quality attributes to determine what needs to be changed about the software.

C. Machine Learning

ML techniques can be considered in four dimensions [49], [50]. First: unsupervised vs supervised vs interactive. An unsupervised learner aims at finding patterns in data sets without labels. Cluster analysis is an example [51]. A supervised learner learns a function that maps input to output based on example pairs. An interactive learner collects the input-output pairs by interaction with the environment. A classic example is reinforcement learning [52]. Second: active vs passive. An active learner queries some information source in the environment (e.g., a user or teacher [53]) to obtain the outputs at new data points that are then used to affect the environment. A passive learner only perceives the information from the environment without affecting it. Third: adversarial vs non-adversarial. Adversarial learning attempts to fool models through malicious input. An example is using obfuscated spam messages in email filtering [54]. Non-adversarial learning has no concept of malicious input. Fourth: online vs batch [55]. Online learning updates the learning model using sequential data, while batch learning learns the model using the entire training data set at once.

We focus here on ML techniques that support building models and/or strategies for CT and MAPE. In applications with high dimensional data that operate in uncertain environments, it is often difficult to manually build models with appropriate precision. ML, especially deep learning, has shown to be a powerful tool to build effective models with small size and high precision by predicting future behaviours based on previously observed data in uncertain environments [56], [57].

IV. Characteristics of Approaches

We outline characteristics of CT and MAPE emphasising strengths, and look at the support ML can offer to adaptation.

A. Control Theory

Discrete- and continuous-time models: CT works with both discrete-time and continuous-time models [58], like those used in "fluid" modelling for queuing systems [59]. Continuous-time models are often written based on conservation laws. Their parameters have a natural physical meaning (e.g., the maximum speed of a CPU), and the effect of modifying them can be foreseen based on physical intuition.

Preserving system properties: both for the discrete- and the continuous-time cases, CT offers formal means to assess whether the system will not drift away from the desired operating point (stability) and what are the tolerable modifications to the system that preserve a given property, such as settling time (robustness) [60]. As long as linear theory applies [61], which is often obtained by convenient local-in-the-operating-point approximations, such checks can be done offline and do not require measured data or synthetic stimuli.

Atomic vs. sequence of control actions: controllers naturally lend themselves to "atomic" control actions, like setting a value for some input to the controlled system. They are not equally apt to more articulated control actions, for example involving a sequence of operations. If some characteristic parameter (e.g., maximum CPU speed) of the system changes, CT can accommodate for such variations as long as a means to detect them online is available (adaptive control [62]). If the same detection implies complex operations on large sets of data, possibly analysing the history of the system, alternative techniques (ML) may be more appropriate.

B. MAPE

Complex combination of sensor data: raw data may be of low-level and need to be combined in complex ways to generate meaningful knowledge to reason about adaptations; different dimensions may apply to combine data, for which MAPE offers support, e.g., combining measurements over time, or integrating data from different data sources.

Complex relation between observable data and properties of interest: deriving properties that are needed to reason about adaptation from observable parameters may be complex and require advanced models and analysis techniques that naturally fit MAPE, e.g., interference in a wireless network may be represented as parameters of a probabilistic model; packet loss can then be predicted using online model checking [45].

Complex quality goals need to be combined: stakeholders often require an adaptive system to provide different quality goals; these goals may be different in nature, e.g., maximise profit, minimise delay, ensure a minimum level of performance, ensure a constant throughput; this leads to conflicts

and requires potentially complex trade-off analysis. Such types of analysis are at the heart of MAPE [63].

Variability: the variation of a software system under control can be parametric or structural (e.g., adding/removing components, changing their connections, and changing deployment of components); MAPE approaches can handle the structural variability in analysis and planning.

Switching types of adaptations/modes: MAPE offers means for adaptations ranging from system parameter tuning to architectural reconfigurations; the latter requires discrete changes of the system, such as activation/deactivation of components, and switching the operation mode of the system.

Long-time scale: achieving the objectives of adaptive systems may require reasoning and planning over long time spans during which the conditions of the system or the environment may change significantly requiring complex replanning.

Known complex actuation: actuation on a software system may require performing a sequence of low-level parametric or structural changes; its execution is not instantaneous generally, but its consequence is predictable. MAPE works with such complex actuation types.

C. Machine Learning

Model building: ML provides methods to build a model for dynamical systems using data, even where first-principle modelling is not possible. This includes, but is not limited to classical system identification techniques used in CT. Recently popular is to use deep learning to complement a rough first-principle model, in order to add non-linear effects and external disturbances, which are difficult to model.

Complexity reduction: ML can be used to reduce the complexity or dimension of a model supporting the design of a CT controller efficiently and effectively. Commonly used ML approaches for model reduction include principal component analysis, singular value decomposition, and auto-encoder.

Estimating properties: For MAPE, ML can provide an a-priori estimation of performance where the environment contains uncertainties or where the environment is not directly observable. ML can also be used for clustering the data, which may provide increased understanding of the patterns in the data. Indeed ML can be used to generate models for the analysis phase of the MAPE loop to provide complex non-linear inference where deriving models traditionally are difficult or impossible. Dimensionality reduction may also help understanding or allow computationally intractable problems to be tackled using traditional software techniques.

Optimise policies: In many domains deriving plans for MAPE is non-trivial and ML techniques may help. Where the problem can be specified as an abstract state representation, reinforcement learning may be employed to optimise policies, i.e., a plan of action in each state, which can then be encoded as a plan in the MAPE loop.

Designing control input: (Deep) Reinforcement learning may also be used to design a sequence of control inputs with or without using a model. However, unlike MPC, it is not always straightforward to incorporate hard-constraints.

V. Combining Adaptation Techniques

With the characteristics and strengths of CT and MAPE in hand, we combine the two and support them with ML. To that end, we use a Cloud-based enterprise system. We also studied a second case of a self-driving car, but due to space limitations, we refer to [64]. Based on our experiences, we propose a first pattern for combining CT with MAPE supported by ML. This pattern offers an initial reusable asset in this area.

A. Cloud-based Enterprise System

Consider a cloud application consisting of logical partitions, e.g., a web application deployed across web, application, and data tiers, or an IoT deployed across fog, edge, and central cloud partitions. Partitions are made of Virtual Machines (VM) with containers (C) that host the application, e.g., microservices that communicate over links within and across VMs.

A common high-level goal of such applications is to serve users with high quality services, while minimising the cost. Goals can be expressed in terms of *Service Level Agreements* (*SLA*), *budget constraints*, and *user satisfaction*. Goals can be translated into technological metrics, such as *response time* and *cost*, which can be considered controlled outputs. The outputs can be affected by control inputs, such as the number of VMs (*#vm*), the number of containers (*#containers*), and the number of threads (*#threads*), buffer or connection pool sizes. Applications are subject to uncertainties (perturbations). *Load*, which is the rate of service requests that arrive at the application, can be highly unpredictable, non-linear, and multi-dimensional. Another uncertainty is *cloud interference*, which is the effect other applications running on the cloud have on the application. Cloud interference that can affect the CPU, IO, memory, bandwidth, etc., can be highly non-linear and is usually not directly measurable.

B. Designing the Controller.

To achieve the goals of a cloud-based enterprise system in the presence of uncertainties, we combine the strengths of MAPE and CT and support these with ML in a four-layer architecture as shown in Figure 1.

Goal layer: The goal layer takes the application owner goals, e.g., an SLA, budget constraints, and user satisfaction, as input and converts them into technological goals for MAPE, such as end-to-end response times and penalties that affect cost.

MAPE Layer: The MAPE layer monitors the progress of the goals and makes reconfiguration decisions, such as changing the number of VMs/containers on each partition. MAPE will also determine the set points for the lower-level CT controllers. To that end, the MAPE layer monitors the response time of each partition, the end-to-end response time, and the current cost (given by the cost of resources in the cloud eventually the cost model of the cloud provider). To determine the control decisions (number of VMs/containers and set points for the CT layer), the MAPE layer can rely on techniques such as

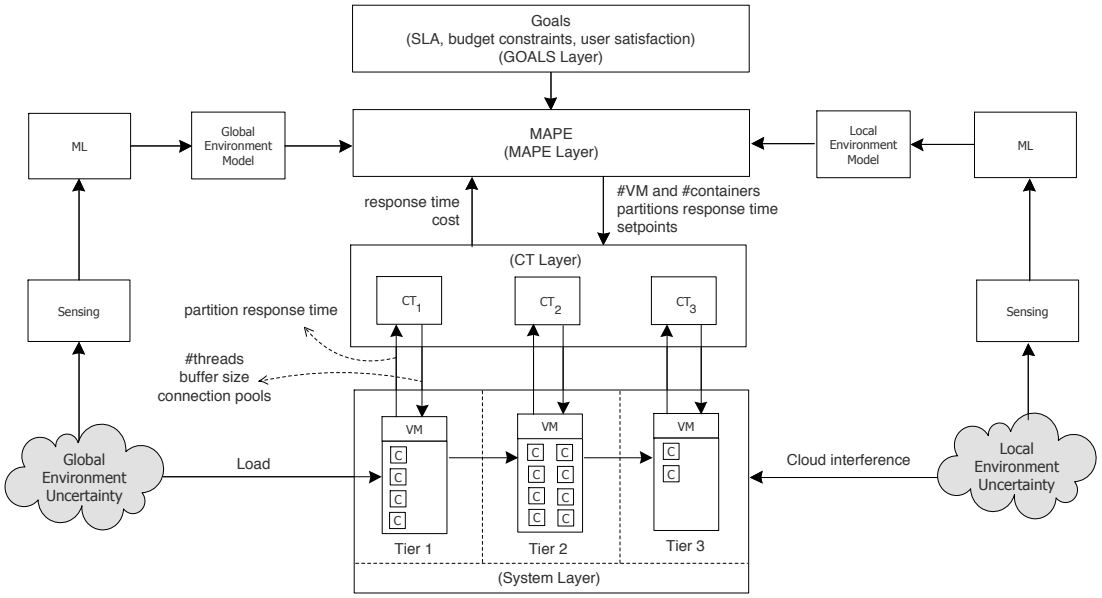


Fig. 1. Adaptive Cloud-based enterprise system.

look-ahead optimisation, search-based algorithms, simulation, or queuing models. The decisions of the MAPE layer are produced in a time scale of order of minutes.

CT layer: The CT layer controls the number of threads used by the microservices among other local parameters, such as buffer sizes, and connection pools. The decisions of the CT layer are produced in a time scale in the order of milliseconds to seconds. By using control theory-based adaptation, this layer is able to deal with fast transients and attenuate high frequency uncertainties. CT is appropriate here since the control inputs and their effects on the outputs are close to a continuous time domain, and assurance for stability and robustness is crucial.

System layer: The system layer that is the subject of adaptation comprises the application logic of a cloud system that is set up as a three-tier architecture as explained above.

The role of ML: ML in our application provides models for: (a) the *load* over time (of the day) as service requests can vary in frequency, data size, and required data outputs, and (b) the *cloud interference* along with high dimensional OS/VM/container metrics over long periods of time. By predicting the request load and expected extra load on the application, the MAPE layer can make better adaptation decisions.

C. Analysis of Combined Architecture

As an initial evaluation of combining MAPE and CT, supported by ML, we model and simulate a simplified version of the three-tier cloud application in Modelica, see Figure 2. The application is modelled in continuous time as three queue-plus-server blocks, each one receiving as input the output of its predecessor. The processing speed is obtained by allotting more or less "Computational Units" (CUs for short, e.g., #threads, #connections) that can range from 0 to some maximum. By changing that maximum, for example by adding or removing threads, we can re-configuring the system.

The CT layer comprises Proportional-Integral (PI) controller that allots CUs to each tier to maintain the desired response time in the face of dynamics of the load and network disturbances (uncertainties *unc*) that affect the throughput. In addition, the PI controllers compute the number of CUs that they would allow to comply with the required response time if an infinite number of CUs would be available as follows:

$$CU_{need} = \frac{CU_{desired} - CU_{maxavail}}{CU_{maxavail}} \tag{1}$$

The CU needs are fed to the upper MAPE/ML layer. Knowing these indices and the required response time for the overall system, this layer can (i) decide the response times for the individual tiers (set points for the PIs) and (ii) reconfigure the system by modifying the maximum CU availability.

Figure 3 shows a simulation run with a randomly varying load (input rate r_{in} at the top, plotted normalised to its nominal value) and efficiencies (2nd plot). We see that the CT layer can keep the system on track when this is feasible. We also see that the CU need indices do signal the need for a reconfiguration, and more importantly, that their shape distinguishes short-time infeasibilities from sustained ones. Dealing with the complex information of these indices and adapting the system accordingly, is a task for which MAPE/ML is particularly well suited. This is illustrated by comparing the CU allocation (3rd and 5th plot) and the obtained response times (4th and 6th plot) respectively without and with activating the functionality of the MAPE/ML layer. The results show that the adaptations applied by the MAPE/ML layer enable the underlying CT layer to better manage temporary infeasibilities.

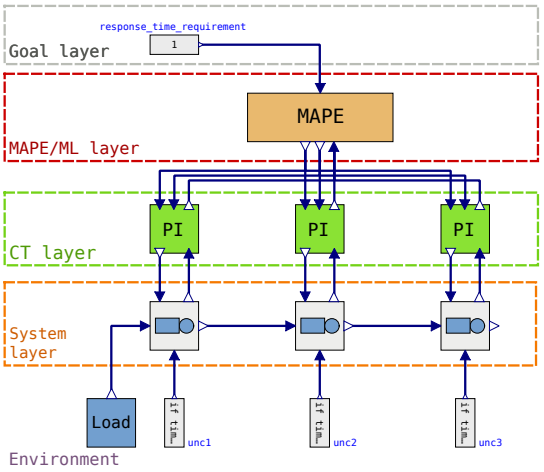


Fig. 2. Modelica diagram for a simplified version of the cloud example.

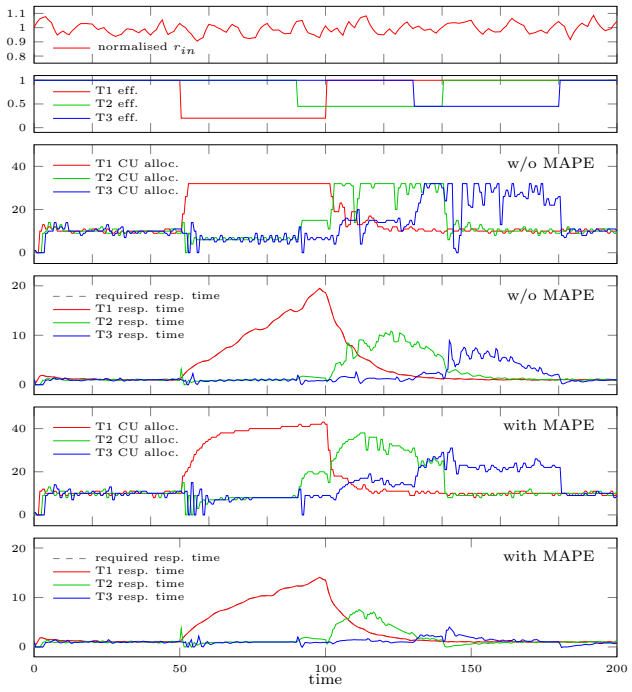


Fig. 3. Simulation results: from the top: in-out rate, efficiencies, required and obtained response times, CU allocation, and resource need indices.

D. Pattern: 1-MAPE-n-CT with ML for Uncertainty Modelling

Based on our experiences, we identified a first pattern for combining MAPE and CT as shown in Figure 4.

The pattern structures the self-adaptive system in four layers where a single MAPE loop is combined with multiple controllers to adapt a distributed system. The rationale for applying the pattern is: (i) CT handles small perturbations, (ii) when CT cannot cope, MAPE is called to adapt the system and change the CT settings to achieve the high-level goals. ML is used to generate up-to-date models of the environment that produces uncertainties the system needs to withstand.

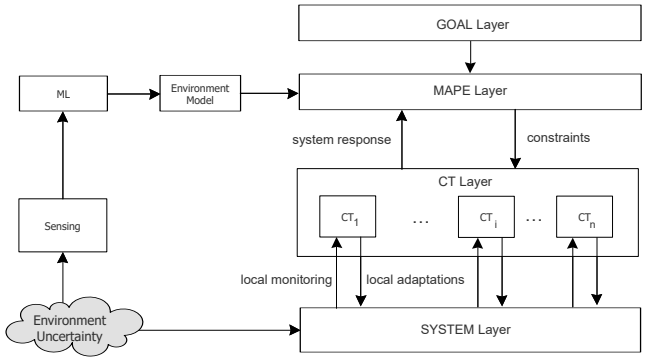


Fig. 4. 1-MAPE-n-CT with ML for uncertainty modelling pattern.

VI. Conclusions and Further Research

This paper presented work in progress on how to combine MAPE and CT and support them with ML to produce better adaptive systems. We characterised the different adaptation approaches: MAPE's strength is its global perspective on the system and its ability to deal with complex trade-offs between requirements. CT's strength lies in keeping variables at prescribed set points or within prescribed ranges, regardless of disturbances. ML, on the other hand, can support MAPE and CT in different ways; one key way is to build runtime models and adaptation strategies from complex and high dimensional data obtained from uncertain environments.

Second, we illustrated how CT and MAPE can be combined and supported by ML using a use case from the cloud domain. In particular, the case shows how self-adaptive software can be organised in four layers. At the top, a goal layer translates user requirements to operational goals for the adaptation logic. Next, a MAPE layer monitors the progress of the goals, the resources available to the system, and system-wide uncertainties in the environment to determine bounds for the reconfiguration decisions. Then, a CT layer tracks the system behaviour and local uncertainties in the environment, to make reconfiguration decisions of the system within the bounds defined by MAPE. Finally, the system layer comprises the managed system that is subject to adaptation. ML techniques can support the MAPE and CT layers in different functions, for instance, building global and local models of the environment.

While these initial insights are promising, further research is needed to better understand how to combine adaptation approaches. Crucial questions to be answered will be: What are the typical use cases for combining MAPE and CT? How to allocate adaptation responsibilities when MAPE and CT are combined? Can we identify patterns to combine MAPE and CT, and what are their tradeoffs? Can we identify typical coordination mechanisms for MAPE and CT to interact? What are the interesting use cases for ML to be applied in adaptive systems? How can we provide guarantees for the adaptation goals in hybrid architectures that combine CT and MAPE? Answering these questions will require a substantial joint effort among researchers with backgrounds in architecture-based adaptation, control theory, and machine learning.

REFERENCES

[1] P. Oreizy *et al.*, "An architecture-based approach to self-adaptive software," *IEEE Intelligent Systems and their Applications*, vol. 14, no. 3, pp. 54–62, 1999.

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

[3] B. H. C. Cheng *et al.*, *Software Engineering for Self-Adaptive Systems: A Research Roadmap*. Springer, 2009, pp. 1–26.

[4] R. de Lemos *et al.*, *Software Engineering for Self-Adaptive Systems: A Second Research Roadmap*. Springer, 2013.

[5] ——, "Software engineering for self-adaptive systems: Research challenges in the provision of assurances," in *Software Engineering for Self-Adaptive Systems III. Assurances*. Cham: Springer International Publishing, 2017, pp. 3–30.

[6] D. Weyns, *Introduction to Self-Adaptive Systems: A Contemporary Software Engineering Perspective*. Wiley, IEEE Press, 2020.

[7] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[8] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-based self-adaptation with reusable infrastructure," *Computer*, vol. 37, no. 10, p. 46–54, 2004.

[9] J. Kramer and J. Magee, "Self-managed systems: An architectural challenge," in *Future of Software Engineering*, 2007.

[10] D. Weyns *et al.*, "FORMS: Unifying reference model for formal specification of distributed self-adaptive systems," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 7, no. 1, 2012.

[11] D. Weyns, "Software engineering of self-adaptive systems," *Handbook of Software Engineering*, 2019.

[12] J. Hellerstein *et al.*, *Feedback Control of Computing Systems*. USA: John Wiley Sons, Inc., 2004.

[13] A. Filieri *et al.*, "Control strategies for self-adaptive software systems," *ACM Trans. on Autonomous and Adaptive Systems*, vol. 11, no. 4, 2017.

[14] S. Shevtsov *et al.*, "Control-theoretical software adaptation: A systematic literature review," *IEEE TSE*, vol. 44, no. 8, pp. 784–810, Aug. 2018.

[15] O. Gheibi *et al.*, "Applying machine learning in self-adaptive systems: A systematic literature review," in *arXiv, 2103.04112*, 2021.

[16] F. Quin *et al.*, "Efficient analysis of large adaptation spaces in self-adaptive systems using machine learning," in *SEAMS*. IEEE, 2019.

[17] N. Bencomo *et al.*, "RaM: Causally-Connected and Requirements-Aware Runtime Models using Bayesian Learning," in *Models*, 2019.

[18] P. Jamshidi *et al.*, "Fuzzy self-learning controllers for elasticity management in dynamic cloud architectures," in *QoSA*, April 2016.

[19] G. Tamura *et al.*, "Improving context-awareness in self-adaptation using the DYNAMICO reference model," in *SEAMS*, 2013.

[20] C. Belta *et al.*, "Symbolic planning and control of robot motion," *IEEE Robotics & Automation Magazine*, vol. 14, no. 1, 2007.

[21] T. Wongpiromsarn *et al.*, "Receding horizon temporal logic planning," *IEEE Transactions on Automatic Control*, vol. 57, no. 11, 2012.

[22] P. Nilsson *et al.*, "Correct-by-construction adaptive cruise control: Two approaches," *Trans. on Control Systems Technology*, vol. 24, no. 4, 2015.

[23] A. Metzger *et al.*, "Feature-model-guided online learning for self-adaptive systems," *arXiv preprint arXiv:1907.09158*, 2019.

[24] F. Affonso *et al.*, "A framework based on learning techniques for decision-making in self-adaptive software." in *SEKE*, vol. 15, 2015.

[25] J. Van Der Donckt *et al.*, "Applying deep learning to reduce large adaptation spaces of self-adaptive systems with multiple types of goals," in *International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 2020.

[26] L. Fernández Maimó *et al.*, "A self-adaptive deep learning-based system for anomaly detection in 5g networks," *IEEE Access*, vol. 6, 2018.

[27] P. Jamshidi *et al.*, "Machine learning meets quantitative planning: Enabling self-adaptation in autonomous robots," in *SEAMS*, 2019.

[28] R. Diniz Caldas *et al.*, "A hybrid approach combining control theory and AI for engineering self-adaptive systems," in *SEAMS*, 2020.

[29] K. Åström and P. Eykhoff, "System identification—a survey," *Automatica*, vol. 7, no. 2, pp. 123 – 162, 1971. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0005109871900598

[30] A. Chiuso and G. Pillonetto, "System identification: A machine learning perspective," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 2, no. 1, pp. 281–304, 2019.

[31] J. Sjöberg, H. Hjalmarsson, and L. Ljung, "Neural networks in system identification," *IFAC Proceedings*, vol. 27, no. 8, 1994.

[32] L. Ljung, *System Identification*. American Cancer Society, 2017, pp. 1–19. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/047134608X.W1046.pub2

[33] A. Filieri, H. Hoffmann, and M. Maggio, "Automated design of self-adaptive software with control-theoretical formal guarantees," in *36th International Conference on Software Engineering*, 2014.

[34] S. Shevtsov, D. Weyns, and M. Maggio, "Simca*: A control-theoretic approach to handle uncertainty in self-adaptive systems with guarantees," *ACM Trans. on Autonomous and Adaptive Systems*, vol. 13, no. 4, 2019.

[35] Z. Wu *et al.*, "Machine learning-based predictive control of nonlinear processes. Part I: Theory," *AIChE Journal*, vol. 65, no. 11, 2019.

[36] D. Nguyen and B. Widrow, "Neural networks for self-learning control systems," *IEEE Control systems magazine*, vol. 10, no. 3, 1990.

[37] T. Chow *et al.*, "A recurrent neural-network-based real-time learning control strategy applying to nonlinear systems with unknown dynamics," *IEEE Transactions on Industrial Electronics*, vol. 45, no. 1, 1998.

[38] R. Kamalapurkar *et al.*, *Model-Based Reinforcement Learning for Approximate Optimal Regulation*. Elsevier, 2016.

[39] Y. Wen *et al.*, "Online reinforcement learning control for the personalization of a robotic knee prosthesis," *Transactions on Cybernetics*, 2019.

[40] A. Leva *et al.*, *Control-based operating system design*. IET, 2013.

[41] J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan. 2003.

[42] G. Blair, N. Bencomo, and R. B. France, "Models@ run.time," *Computer*, vol. 42, no. 10, p. 22–27, Oct. 2009. [Online]. Available: https://doi.org/10.1109/MC.2009.326

[43] J. Cámara *et al.*, *Analyzing Self-Adaptation via Model Checking of Stochastic Games*. Springer, 2017, no. LNCS 9640.

[44] D. Weyns and M. Iftikhar, "Model-based simulation at runtime for self-adaptive systems," in *Models at Runtime, IEEE International Conference on Autonomic Computing*, 2016, pp. 364–373.

[45] R. Calinescu *et al.*, "Engineering trustworthy self-adaptive software with dynamic assurance cases," vol. 44, no. 11, 2018, pp. 1039–1069.

[46] "An Architectural Blueprint for Autonomic Computing," IBM, Tech. Rep., Jun. 2005. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.150.1011&rep=rep1&type=pdf

[47] Villegas *et al.*, "Architecting software systems for runtime self-adaptation: Concepts, models, and challenges," in *Managing Trade-Offs in Adaptable Software Architectures*. Morgan Kaufmann, 2017.

[48] A. de Castro Alves, *OSGi in Depth*. Manning Publ. Co., USA, 2011.

[49] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*. Cambridge University Press, 2014.

[50] C. Bishop, *Pattern recognition and machine learning*. Springer, 2006.

[51] A. Albalate and W. Minker, *Semi-Supervised and Unsupervised Machine Learning: Novel Strategies*. Wiley, 2013.

[52] R. S. Sutton and A. G. Barto, *Reinforcement Learning, second edition: An Introduction*. Westchester, 2018.

[53] B. Settles, "Active learning literature survey," *TR1648*, 2009.

[54] E. Blanzieri and A. Bryl, "A survey of learning-based techniques of email spam filtering," *Artificial Intelligence Review*, vol. 29, p. 63–92, 2008.

[55] N. Littlestone, "From on-line to batch learning," in *2nd Annual Workshop on Computational Learning Theory*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1989, p. 269–284.

[56] Y. Bengio, *Learning deep architectures for AI*. Now Publishers, 2009.

[57] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[58] G. F. Franklin, J. D. Powell, M. L. Workman *et al.*, *Digital control of dynamic systems*. Addison-wesley Reading, MA, 1998, vol. 3.

[59] R. Kumar, Y. Liu, and K. Ross, "Stochastic fluid theory for p2p streaming systems," in *IEEE INFOCOM 2007-26th IEEE International Conference on Computer Communications*. IEEE, 2007, pp. 919–927.

[60] D. Hinrichsen and A. J. Pritchard, *Mathematical systems theory I: modelling, state space analysis, stability and robustness*. Springer Science & Business Media, 2011, vol. 48.

[61] J. P. Hespanha, *Linear systems theory*. Princeton university press, 2018.

[62] K. J. Åström and B. Wittenmark, *Adaptive control*. Courier Corporation, 2013.

[63] S.-W. Cheng, D. Garlan, and B. Schmerl, "Architecture-based self-adaptation in the presence of multiple objectives," in *Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2006.

[64] Weyns *et al.*, "Self-driving car case," 2021. [Online]. Available: https://people.cs.kuleuven.be/danny.weyns/material/smart-vehicle.pdf