Thesis for The Degree of Licentiate of Engineering

Engineering Software for Resilient Cyber-Physical Systems

Ricardo Diniz Caldas

Division of Software Engineering
Department of Computer Science & Engineering
Chalmers University of Technology | University of Gothenburg
Gothenburg, Sweden, 2021

Engineering Software for Resilient Cyber-Physical Systems

Ricardo Diniz Caldas

Department of Computer Science & Engineering
Division of Software Engineering
Chalmers University of Technology and Gothenburg University
Gothenburg, Sweden

This thesis has been prepared using LaTeX.
Printed by Chalmers Digitaltryck,
Gothenburg, Sweden 2021.

"Software is an invisible thread, and hardware is the loom on which we weave the fabric of computing. "

- Grady Booch

Abstract

Designing, implementing, and verifying resilient cyber-physical systems is challenging. Resilience is the ability to provide the required capability when facing adversity. Resilient cyber-physical systems should avoid, withstand, recover from, and evolve and adapt to cope with adversity stemming from computation, networking, or physical environment. From the engineering point of view, the usefulness of such systems is hindered by their lack of ability to adapt and overcome unknown stimuli, ever-changing and conflicting objectives, and deprecated internal components. Software as a tool for self-management is a key instrument for dealing with uncertainty. Yet, engineering software for resilient cyber-physical systems is hard since the effects of operating under the unknown might emerge during the execution, requesting decision-making at runtime rather than design time. Decision-making at runtime should guarantee the satisfaction of system goals, work efficiently to be effectively used in practice, and guarantee the expected quality. With this in mind, this thesis contributes towards the engineering of software for resilient cyber-physical systems by (i) combining control theory and artificial intelligence for efficient adaptation, (ii) using formal methods for ensuring correctness of control-theoretic software adaptation, and (iii) promoting a language for scenario-based testing autonomous systems. We found that the hybrid approach, combining control theory and artificial intelligence, improves the efficiency of the adaptation mechanism. The results shed light on the interplay between control theory and artificial intelligence as fundaments for engineering resilient cyber-physical systems. Yet, incorporating machine learning and control theory introduces non-deterministic autonomic behavior, posing a challenge for the assurance provision for such tools. On the one hand, we found that the use of formal methods helps to build confidence in software-based controllers. On the other hand, large and complex systems place barriers to the usage of formal methods. Thus, we explore the use of testing and specifically scenario-based testing for validating large and complex cyber-physical systems that are required to operate in complex and unpredictable environments, like autonomous vehicles. In a nutshell, this thesis argues in favor of introducing control theory and artificial intelligence in designing and implementing software-based controllers. Also, we exploit formal methods and testing as instruments for verifying and validating cyber-physical systems.

Keywords

resilience, smart cyber-physical systems, software, self-adaptation, control theory, machine learning, formal methods

Acknowledgment

I thank my supervisors, Patrizio Pelliccione and Thorsten Berger. Patrizio's guidance is constructive and empowering. Thorsten inspires leadership and effectiveness. I also thank all the EaseLab group members, especially Razan and Wardah. I am glad to be part of the Interaction Design and Software Engineering Division of the Computer Science and Engineering Department. I am also grateful to be part of the WASP community.

I thank Julianna for her patience, commitment, and always encouraging creativity. Anton, Kristina, and Zabou for the memorable moments and delightful company. Juan for the kindness, and Florence for showing the outside-the-box. Saulo for the company and enlightening discussions. Last but not least, my family and friends that are an ocean away from whom I gather strength in the moments of *saudade*.

This work is supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

List of Publications

Appended publications

This thesis is based on the following publications:

[A] R. Caldas, A. Rodrigues, E. B. Gil, G. N. Rodrigues, T. N. Vogel, P. Pelliccione "A Hybrid Approach Combining Control Theory and AI for Engineering Self-Adaptive Systems"
15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), IEEE/ACM, 2020.

[B] E. B. Gil, R. Caldas, A. Rodrigues, G. L. G. da Silva, G. N. Rodrigues, P. Pelliccione "Body Sensor Network: A Self-Adaptive System Exemplar in the Healthcare Domain"
16th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS). IEEE/ACM, 2021.

[C] R. Caldas, R. Ghzouli, A. V. Papadopoulos, P. Pelliccione, D. Weyns, T. Berger "Towards Mapping Control Theory and Software Engineering Properties using Specification Patterns"
2nd International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS). IEEE, 2021.

[D] R. Queiroz, D. Sharma, R. Caldas, K. Czarnecki, S. Garcia, T. Berger, P. Pelliccione "A Driver-Vehicle Model for ADS Scenario-based Testing"
In submission.

Other publications

The following publications were published during my PhD studies, or are currently in submission/under revision. However, they are not appended to this thesis, due to contents overlapping that of appended publications or contents not related to the thesis.

[a] G. S. Rodrigues, R. Caldas, G. Araujo, V. de Moraes, G. Rodrigues, P. Pelliccione "An Architecture for Mission Coordination of Heterogeneous Robots"
Submitted and under revision to ACM Journal of Systems and Software.

[b] M. Askarpour, C. Tsigkanos, C. Menghi, R. Calinescu, P. Pelliccione, S. García, R. Caldas, T. J. von Oertzen, M. Wimmer, L. Berardinelli, M. Rossi, M. M. Bersani, G. S. Rodrigues "RoboMAX: Robotic Mission Adaptation eXemplars"
16th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), IEEE/ACM, 2021.

Research Contribution

In this section, I describe my contributions to the appended papers. The contributions are classified using the Contributor Roles Taxonomy (CRediT)[1].

In Paper A, I led the conceptualization and implementation of the software controller and the formal analysis and validation (artifact reuse). Also, I contributed to the study design, visualization, and writing.

In Paper B, I led the conceptualization and implementation of the artifact. Also, I contributed to the review and editing of the final draft and visualization. I played a supporting role in the formal analysis and validation (reproducibility) of the artifact.

In Paper C, I led the study design and implementation, responsible for modeling and checking the property against the system model. I also led the management (research planning), resource allocation, and final draft writing. I have contributed to the analysis, validation, and visualization.

In Paper D, I led the implementation of the behavior-tree-based DSL. I contributed to the formal analysis by implementing the NHTSA scenarios and collecting data to account for the expressiveness and reuse of the language. I played a supporting role in the paper writing.

[1] `https://casrai.org/credit/`

Contents

Chapter 1

Introduction

Cyber-Physical Systems (CPSs) unite the digital and physical worlds, increasingly supporting individuals and groups in their social and professional endeavors. In contrast to traditional embedded systems, CPSs are often designed as networks of interactive and dynamic elements [1], including healthcare systems, mobility systems, process control systems, and collective robotics. Such applications directly reflect strategic economic and social development areas, i.e., transport, energy, well-being industry, and infrastructure. An example of CPS is the central patient monitoring system from Philips[1] in which sensors continuously monitor patients' vital signals keeping nurses and doctors informed even in the distance. More precisely, cyber-physical systems are *"engineered systems that are built from, and depend upon, the seamless integration of computational algorithms and physical components"* [2].

What are the technological challenges for engineering cyber-physical systems? The European roadmap and strategy for cyber-physical systems (CyPhERS) lists five challenges [3].

- Interoperability [4]: Integrating components from different suppliers is especially challenging in the presence of legacy parts and when aiming at the continuous update of the cyber-physical system [5, 6].
- Autonomy: The large scale and complexity of CPSs amplify the challenge of increasing the level of automatic behaviour in the individual components participating in the system and their collaboration [7].
- Privacy: When there is the need of collecting and processing data along the cross-organization chains of services, data become available to participating parties of these services, and this exacerbates issues on the security and privacy of sensitive information [8, 9].
- Resilience[2]: Faults, and in general changes, threaten the integrity of the services provided by cyber-physical systems and might compromise their persistence [10]. Dealing with faults emerging in dynamic environments is challenging, especially in large and complex CPSs [11].

[1] `https://www.usa.philips.com/healthcare/solutions/patient-monitoring/central-monitoring-systems`

[2] Schatz [3] originally proposes dependability. Instead, we discuss *resilience* following Laprie's rationale [10].

- Uncertainty: Dealing with imprecise or incomplete information and seamlessly operate in the face of the unknown [12] threaten confidence in the provided service [13].

Although all these challenges are important and interconnected, we focus on discussing resilience, autonomy, and uncertainty as cross-cutting concerns in the engineering of *resilient* CPSs. Engineering resilient CPSs involves rethinking the design with a focus on dynamic environments. CPSs are subject to interactions with (human) users and operators [14, 15], changing needs [16], and heterogeneous components joining and leaving the system [6]. The software controlling the CPS operation must automatically manage its dynamicity and uncertainty, using, for instance, *self-adaptation* to continuously adequate the system behavior to the specification goals [17–19]. Accordingly, the CPS control software must also *provide assurances* guaranteeing compliance with the system requirements [20, 21].

The overall objective of this research is to support self-adaptation, verification, and validation for resilient CPSs. We elaborate the research objective in three research questions.

RQ1: How to design efficient adaptation for cyber-physical systems operating in the face of uncertainty?

Two established approaches guide the design of software for promoting resilient operation through self-adaptation: (i) architecture-based adaptation, which relies on Monitor-Analysis-Planning-Executing (MAPE) components that reason over knowledge to make adaptation decisions [22–25], and (ii) control-based adaptation, which relies on principles of control theory (CT) to realize adaptation of a target system [26–28]. Control-based methods are promising since they enable correct-by-design solutions. However, the high level of uncertainty in CPS impairs the application of traditionally designed adaptation using control-theoretic methods, asking for runtime evolution of the control software. To answer RQ1, we delve into improving control-based designed self-adaptation with learning to cope with uncertainty.

RQ2: How to formally verify whether the adaptation complies with control-theoretic specifications?

Introducing control-theoretic components in the adaptation loop hinders the traditional software-based verification of adaptation against CPS requirements. Verification techniques can be used to assess the control behavior, e.g. SMT for checking ordinary differential equations [29], or hybrid model checking [30]. As far as we are concerned, such techniques cannot be used for the formal verification of software adaptation loops, which typically aim at verifying software engineering properties like safety, liveness, reachability, and others [31]. To answer RQ2, we find it essential to map control-theoretic specifications and software properties to verify adaptation loops enhanced with control-based components.

RQ3: How to validate large and complex cyber-physical systems operating in realistic scenarios?

Cyber-physical systems are often large and complex systems that must account for partially-controllable and partially-known environment and human interactions. Validating their behavior in the face of adversities is a challenging task. On the one side, automated checking of formal properties is effective but does not scale. On the other side, testing scales but it requires models for the specification of complex interactions with the environment and external agents (either humans or other systems). To answer RQ3, we develop a model for scenario-based testing autonomous driving vehicles in realistic scenarios.

Our research is organized and reported in four papers, indexed A, B, C, and D, answering questions RQ1, RQ2, and RQ3.

1.1 Background and Related Work

Resilience is achieved by reacting to changes, faults, and/or challenges to normal operation to provide and maintain an acceptable level of service. Resilient behavior can be achieved through control theoretical design. Control theory presents a set of rigorous methods for addressing runtime changes, and it has been applied to adapt software applications. This section describes the fundaments of control theory and the state-of-the-art in using control theory to adapt software. Moreover, resilient behavior requires justification of correctness. This section also hovers over state-of-the-art methods assuring correctness in cyber-physical systems.

1.1.1 Fundaments of Control Theory

1.1.1.1 Feedback control loop

Feedback control loop is a form of control, which aims at realizing a feedback loop [32] originating from control engineering [33]. As shown in Fig. 1.1, the *plant* is the subsystem to be controlled by a control signal from the *controller*. A reference, i.e., a setpoint, is defined for an attribute observed on the plant. The controller's goal is to keep the system attribute sufficiently close to this reference despite the disturbance affecting the plant. The error, i.e., the difference between the system attribute and the reference, is fed back to the controller that computes the control signal.

A control system that uses a feedback loop to adjust its behavior is a closed-loop system [26]. Such a system can have different responses to change, that is, how the attribute of interest develops over time after a perturbation or reconfiguration.

1.1.1.2 Control properties

Control specifications determine the desired behavior of the control system in terms of control-theoretic properties. Such properties are used as design objectives and for analysis of the control system. As shown in Fig. 1.2 the

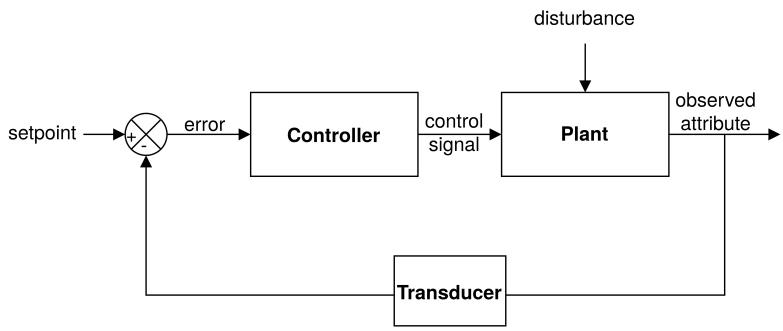

Figure 1.1: Block diagram representing a closed feedback control loop.

typical properties are stability, accuracy, settling time, and overshoot properties also referred to as SASO properties in classical control. A typical feedback loop controller is the Proportional-Integral-Derivative (PID) controller [34]. The PID controller is also known as a three-term controller, alluding to the three parameters (K_p, K_i, K_d) that can be configured. Eq. 1.1 illustrates the controller's transfer function[3].

$$G(s) = K_p + K_i \frac{1}{s} + K_d s \tag{1.1}$$

In the context of this thesis, a control specification is an instance of control transfer functions either in algebraic formulation, such as Eq. 1.1, or block diagram.

Stability. There are a few definitions for stability in the control literature [36]. The bounded-input, bounded-output (BIBO) stability defines a system is stable for all bounded inputs when the output signal is also bounded [26]. This theory accounts for inputs such as steps but not for input ramps or cases of exponential signals. Another form of stability is Lyapunov stability, which is defined over equilibrium points. In this theory, if the initial value of the output signal is close enough to the equilibrium point, then the evolution over time of the output signal will not diverge from the equilibrium point [36].

Settling Time. The settling time is the amount of time required for the output signal to stay within an acceptance margin of its final value, for all future times [33].

Accuracy (or steady-state error). Once stabilized, if the output of a system at a steady state does not exactly agree with the input, the system is said to have steady-state error. This error is indicative of the accuracy of the system [37].

Overshoot/Undershoot. The overshoot is the percentage of the final value by which the output signal rises above the final value. [33].

[3]The controller's transfer function models the controller's output for each possible input. In Laplace domain (s-domain) [35].

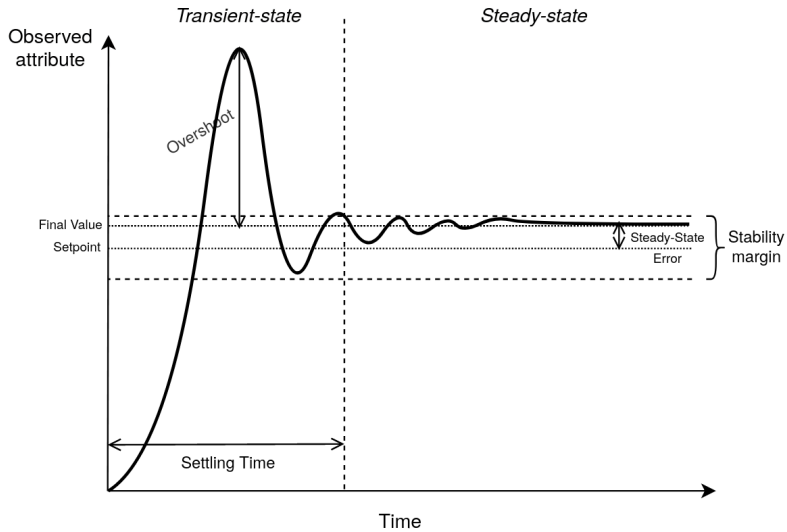

Figure 1.2: Illustrative response of a system to a sudden step variation on the reference.

1.1.2 Control-based Self-Adaptation

How do engineers use control theory to design resilient cyber-physical systems? Engineering controllers for resilient operation is not a one-way process [38]. Changes in the environment or in the system itself require the evolution of the controller model. Runtime changes challenge the controller evolution. To face this challenge, software engineers are working on new techniques to handle change at runtime without incurring penalties and downtime, culminating in what is commonly referred to as software self-adaptation [28].

1.1.2.1 Taxonomy and History of self-adaptation

As shown in Fig. 1.3 a taxonomy on self-adaptation provides an overview self-adaptation as an engineering discipline [39, 40].

The evolution of self-adaptive systems can be explained from a historical perspective through both seminal roadmaps on software engineering for self-adaptive systems [41, 42] and summarized in a set of seven complementary waves [43], encompassing the topics of (i) automation of management tasks; (ii) architecture-based adaptation; (iii) use of models at runtime; (iv) adaptation based on goals; (v) provision of guarantees under uncertainties; (vi) control-based approaches; and (vii) use of learning techniques in adaptation. In this thesis we touch several waves, (iv) adaptation based on goals, (v) provision of guarantees under uncertainty, (vi) control-based, and (vii) use of learning. We mainly focus in providing a background and discussion on wave (vi) control-based approaches since it would be needed to understand our contribution in the field.

Figure 1.3: A taxonomy of self-adaptation

1.1.2.2 Challenges and engineering

In a systematic literature review, Stepan et al. [28] characterize the state-of-the-art and practice in applying control theory for designing software self-adaptation. We summarize their findings in four noteworthy insights that shed light on current challenges in advancing control-based self-adaptation.

- Most studies motivate control theory in self-adaptation design aiming to obtain formal guarantees of adaptation correctness.
- The majority of the software models used are linear and time-invariant, although software models are considered highly non-linear.
- Current research does not exploit the full potential of control theoretical guarantees, focusing mostly on software qualities.
- It is not known whether and how software qualities relate to control theoretic guarantees.

Engineering software self-adaptation in the light of control theory involves an iterative process where control experts work separately from software engineers [44]. Such a process usually comprises six steps: (1) identify the adaptation goals, (2) identify the knobs, (3) devise the system model, (4) design the controller, (5) implement and integrate the controller, and (6) test and validate the system [27, 45]. These steps are detailed in the following:

(1) Goals Identification. The simplest type of objective is tracking a reference value. In this case, the controller keeps the system attribute quantity as close as possible to the given reference value, maintaining a constant rate of frames per second [46]. The second type of goals resides in a specific threshold.

For example, a meeting scheduler must keep a low weekly cost with at most 85% of success rate [47]. The third type of goal concerns the optimization of a system attribute. For example, we may want to minimize the response time in cloud applications [48, 49].

(2) Identify the Knobs. Knobs are configuration options that can be tweaked to reach the specified goals. There are two main types of knobs and parameters. For example, they might affect the changing of the parameter that allows only mandatory content appearing in the web application [50]. As another example, a knob or parameter might deal with swapping agents with different workload policies in travel (flight) reservation system [51]. Souza et al. proposed a methodology for identifying possible knobs and their impact in the system [52].

(3) Modeling. In control theory, engineers devise analytic models, i.e., mathematical relationships like logic formulas or dynamic models. A dynamical model can be either in state-space form [53] or expressed as an input-output relationship [33], also referred as transfer function. For instance, engineers might be interested in modeling disturbances of the system or somehow dealing with those disturbances' existence. In a previous work [54], we provide a framework for (goal-)modeling the system by explicitly acknowledging uncertainty (i.e., disturbance). This permits to have an automatic model transformation to input-output relationship ready to be used in control-theoretic adaptation.

(4) Controller Design. There are three main techniques for the controller design, i.e., the control synthesis[4]:

- Synthetic design consists of taking pre-designed control blocks and combining them. A typical tool used for synthetic design is Simulink [55].
- Control selection with parameter optimization is similar to the first one. The choice of the control parameters follow methods for finding (nearly-)optimal parameters depending on the model, for example, relay feedback [56].
- Analytical design consists of solving an analytical problem such as calculating the PID parameters from the model [34].

(5) Implement and Integrate. The implementation and integration of controllers into a larger system is considered a challenging task [57]. In principle, engineers follow a model-driven approach [58, 59], relying on a variety of tools and modeling languages (e.g., MATLAB/Simulink, Modelica, SCADE), where the control model is transformed into hardware- and environment-specific models and eventually into source code. The generated code needs to be further optimized and customized before it can be integrated into a larger system, for instance, to deal with integrator windup or actuator saturation [34, 60, 61]. This requires extensive manual modification of the code [62]an error-prone activity that may introduce bugs that are hard to spot without proper verification techniques.

(6) Test and validate. This can be divided in checking the controller itself and checking the control system, i.e., the controller together with the system under control. Static analysis can be used for conformance of the controller

[4]This terminology is used in control engineering, contrasting other definitions in computer science.

code to its intended behavior and to numerical errors using modern SMT tools applied to ordinary differential equations (ODEs) [29] or hybrid model checking [30]. Current hybrid model checkers are usually limited to linear differential equations [63,64]. Testing the controller separately from the control system is insufficient since the larger system also influences the controller. One common way to validate the control system is to show statistical evidence, for example, using experimentation [50].

It is also desirable to check the properties present in the design of control-theoretical adaptation, the control-theoretic (CT) properties. Such properties are usually expressed in ODEs and cannot be directly checked against the control system. As a first step into checking such properties in code, Villegas et al. [65] analyze the CT properties from the software engineering perspective proposing a mapping between CT properties and software quality attributes, i.e., Performance, Dependability, and Security. Then, Camara et al. [66] modeled the CT properties using Linear Temporal Logic (LTL).

1.1.3 Verification and Validation of CPSs

Roughly, only 5% of 58 studies on safety for mobile robotic systems present technological advancement with enough maturity to be adopted by the industry [67]. Another study found out that roboticists do *testing* but hardly ever use *formal methods* for assuring the quality of their implementation [68]. Also, roboticists list robustness, *validation*, and *simulation* among the major challenges of working in the field [68]. In this section, we hover upon techniques, i.e., formal methods, run-time verification, and testing, to support the adoption of cyber-physical systems in the real world by providing resilience from the software perspective. Although we do not explicitly use a few techniques in this thesis, for instance, static analysis, run-time verification, field testing, and mutation testing, we describe them for the sake of completeness, highlighting fields that we consider for future work.

1.1.3.1 Formal methods

Static analysis efficiently computes approximate but sound guarantees about the behavior of a program without executing it [69]. For example, abstract interpretation can be used in control software for anticipating run-time failures to compile-time [70,71]. Static analysis can also be used for type inference in untyped or weakly typed languages or type checking in languages with non-static type systems [72]. For example, by inferring dimensional inconsistencies in robotics applications [73–75], we can approximate the computation semantics to the physical (real-)world's semantics [76,77]. However, for mission-critical CPS applications existing industry static analysis tools either do not scale well or present many false positives [78]. Moreover, as opposed to model checking, it does not generate counterexamples [79] when the verification is not successful.

Model checking is the automatic verification of a system against its formal specification [80–83]. A set of model checkers are available for use varying modelling languages, e.g. C++ code in DIVINE [84], or timed automata in Uppaal [85], or property specification languages, like probabilistic computation tree logic (PCTL) in PRISM [86]. A benefit of most model checkers is the

automatic counterexample generation for tracing back the path that violates the specification. However, model checking is prone to state-explosion [87], and complexity issues in the property specification [88].

On the one side, raising the model abstraction may avoid the state explosion. In counterpart, representativeness is lost. This can be a problem, especially when the environment and complex interactions are important. On the other side, too difficult properties specifications create barriers to the successful application of model checking. Property specification patterns [89, 90] rely on proven solutions to common domain problems to assist in formalizing such properties to tackle the property specification problem.

A specification pattern [89] is defined as a tuple <*Scope, Pattern*>. A Scope determines the *extent of a program execution over which the pattern holds*[5]. Examples of scopes are Globally (entire execution trace) and After X (execution trace after a state/event X). A Pattern describes a *generalized recurring system attribute* [90]. Examples of patterns are Universality (a property that always holds) and Existence (a property that eventually holds). A property can be specified by one specification pattern or composition of multiple patterns using a nesting operator. For instance, the Universality and the Existence patterns might be composed under the Globally scope to obtain the Globally, Universality Existence (i.e., $\Box\Diamond P$).

Formal methods alone are not enough to tackle the challenges in CPS verification and validation [91, 92].

1.1.3.2 Run-time verification

Typically in run-time verification, a property is translated into a monitor. Such monitor checks whether a current execution or traces of a finite set of executions hold against the property [93]. Challenges in run-time verification of cyber-physical systems stem from specifying behavior and generating monitors from a specification [94, 95].

One of the main specification languages for the formal specification of CPSs is Signal Temporal Logic (STL) [96, 97]. STL admits ordering of events and the notion of a distance metric. A taxonomy of various types of signal-based properties was used to specify properties of a satellite control system for attitude determination [98]. A pattern-based domain-specific language aids the specification and trace-checking using STL properties [99].

Synthesizing monitors for CPSs face complexity, changing and uncertain operating environment, and the monitors need to avoid much overhead. For example, PREDIMO [100] generates predictive monitors from scenario-based specifications for preventing near future failures in the smart homes domain. Another example is the generation of monitors departing from automatically inferred invariants from physical constraints in robotic systems [101].

1.1.3.3 Testing

Testing software activities can be categorized in *in-house*[6] testing, and field testing [102]. In-house testing is generally conducted independently from the

[5] `https://matthewbdwyer.github.io/psp/patterns/scopes.html`
[6] also named *in-vitro*.

production context, except for the failures reported from the field, which may trigger in-house testing activities. Field testing consists of generating test cases that can be executed directly in the field on the same instance of the software used in production, i.e., *online testing*. Field testing can also happen on a separated instance still running in production, i.e., *offline testing*. For example, a study builds on ChaosMonkey (i.e., field testing technique from Netflix [103]) for developing a fault injection mechanism for testing the robustness of radio base stations similarly in the running system [104]. The last category of field testing consists of in-house testing but on data collected from the field, i.e., *ex-vivo testing*. However, testing CPSs often require dedicated hardware, software emulators, or simulators [105, 106] which poses a challenge to understanding the quality of the test suite.

Mutation tests automatically mutate the program syntax producing semantic program variants, i.e., generating artificial defects for qualifying the test suite. The raised challenges for testing CPSs substantially limit the applicability of mutation testing in such domain. To this end, a series of domain-specific optimizations enhanced the mutation analysis pipeline for testing space software [107]. In the autonomous driving vehicles domain, mutation analysis has been used to assess a set of quality metrics, and oracles for validating the quality of tests for autonomous driving vehicles (ADVs) [108]. Moreover, an initial study proposes the development of a domain-specific language enabling the definition of oracles independent of underlying ADVs simulators [109].

Domain-specific languages (DSLs) are a key-enabler for scenario specification in ADVs simulation and testing, e.g., Geoscenario [110], SUMO [111], and Carla [112]. DSLs for specifying the environment is a promising path for testing CPSs [113]. Furthermore, the gap between simulation and the real world raises concerns about the usefulness of such scenario-based approaches. To this end, a study relies on formally specifying scenario descriptions with a photorealistic full-stack simulator for ADVs [114]. Another study integrates sensing data from simulation and the real world for promoting mixed-reality testing [115]. A third study integrates a robotic digital twin for prediction of the robot's movement in parallel with the actual operation of the robot for online testing safety constraints [116].

1.2 Methodology

We have followed *design science* to conduct the research presented in this thesis. Fundamentally, design science is a problem-solving paradigm [117], seeking to deliver innovative and useful ideas, practices, technical capabilities, and products. The paradigm looks at research as means to delivering relevant artifacts justified by knowledge applied with rigor [118–120]. Design science leverages *solution-seeking research* [121] in a suitable research framework to tackle our research goal.

The goal of our research is to support self-adaptation, verification, and validation for resilient CPSs. To this end, we collect relevant problems elicited in the scientific communities and industrial initiatives, and rigorously apply fundamental and applied knowledge from control theory and software engineering to fulfill the research objectives. Our papers A-D contribute artifacts in accordance

to the problem-solution framework from Engström et al. [122]. We classify our papers and their contributions using the clustering from their paper, based on individual knowledge-constructing activities: descriptive study, solution validation study, solution design study, and problem-solving study.

Papers A and B in combination are classified as a *solution validation* study. The problem tackled by the two papers aim at distilling the design of adaptation for CPSs operating in scenarios with emergent uncertainties. The solution design promotes increasing the efficiency of the search for adequate controller parameters by applying an evolutionary algorithm that learns from the operational context. Concretely, we instantiate an NSGA-II optimization pipeline for processing runtime input and output data from the software controller and its effects in the controlled system. Our solution leverages (i) *autonomic* operation by supporting the controller parameter's reconfiguration with less human intervention, (ii) *resilient* operation by reducing the number of runtime faults, and (iii) *uncertainty* management by enabling reconfiguration for dynamic scenarios operation. We implement our solution in the healthcare domain and demonstrate efficient adaptation meanwhile preserving the controller specifications in uncertain scenarios. The solution is published and validated as a scientific artifact in an open repository[7].

Paper C is classified as a problem-solution pair study. Departing from the gap between control-theoretic specifications and control software verification, the solution design builds on a mapping between the properties from control and software disciplines. We concretely follow the complete engineering process, modeling the control specification in software-based notation, and validate one controller-theoretic property against code. Our solution leverages (i) *autonomic* behavior by supporting the verification of software controllers, and (ii) *resilient* operation by enabling the collection of assurance cases on software controllers in the light of control theoretic specifications. We implement the whole engineering process for controlled experimentation, using external tooling for validation, and publish the experimental results along instructions for replicating the experiments in a dedicated online repository.

Paper D is also classified as a problem-solution pair study. Departing from the problem of testing realistic crash scenarios in large and complex systems, the solution design unites domain specific languages and mathematical models for trajectory planning. Concretely, we propose a three-layered driving model separating driving behavior, maneuver models and scenario execution, and evaluate it in terms of scenario design effectiveness using real-world pre-crash scenarios. Our solution leverages (i) *autonomy* by improving testing of the system under test interaction with multiple agents in complex scenarios, (ii) *resilient* operation by improving the means to generate assurance cases against critical scenarios, and (iii) *uncertainty* management by enabling the simulation dynamic interactions. The solution builds on previous works and state-of-the-art DSL design and mathematical models for trajectory simulation. We implement the model and validate the language expressiveness, and reuse as well as simulating scenarios with a real-world autonomy stack. The evaluation relies established pre-crash scenarios from the regulatory institution for vehicle safety in US, National Highway Traffic Safety Administration (NHTSA). Also, we compare the driver model's motion accuracy to naturalistic urban traffic.

[7] `https://www.hpi.uni-potsdam.de/giese/public/selfadapt/exemplars/`

Lastly, we designed scalability experiments with 20 vehicles to understand whether our model supports high-density traffic simulation.

1.3 Summary of Papers

In this section, we introduce a summary of the papers on which this thesis is based and relate them to the research questions formulated in the introduction.

1.3.1 Paper A

There is an increasing trend in using control theory to guide the design of software controllers for dynamic adaptation with guarantees of trustworthiness and robustness. Software controllers designed through control theory must quantitatively guarantee its convergence in time to reach the adaptation goal, even in the face of errors and noise. When operating under high levels of uncertainty, the control model must evolve to cope with the system model drift from reality. Therefore, in the control-theoretical design of software controllers, it is paramount to identify appropriate adaptation strategies by efficiently exploring the solution space. Artificial Intelligence (AI) techniques have been used to avoid the exhaustive analysis of large adaptation space. Although such significant contributions in the literature have promoted a solid mathematical foundation of control theory for building software controllers, there is still the need for understanding:

i. How can AI aid the parameter optimization of controllers?

ii. How to apply AI-based techniques in the search for adaptation solutions in control-theoretical strategies?

The study reported in paper A is motivated by these questions. Our long-term goal with this study is to design and implement a control-theoretic adaptation enabling efficient exploration of the solution space and using AI, which minimizes errors due to the system model drift during runtime.

We designed and implemented a two-layered software controller to achieve our research goals, showcasing the optimization of PID parameters aided by an evolutionary algorithm (NSGA-II). To validate our solution's efficiency, we collect data from the optimization pipeline to find an adaptation solution. Then, we designed experiments employing a self-adaptive system in the healthcare domain, namely the Body Sensor Network (BSN). Scenarios with uncertainty stress the software controller to assess whether the AI-aided parameter optimization found an effective solution. We show that our hybrid approach finds nearly optimal solution within manageable time.

Paper A contributes to the design and implementation of adaptation for efficiently finding adaptation strategies for resilient CPSs (RQ1).

1.3.2 Paper B

The development of healthcare CPSs is challenging due to their safety-critical nature. Providing assurance evidence for the compliance of quality attributes

such as safety, reliability, and energy cost is vital in this domain. Therefore, equipping these systems with self-configuration, self-healing, and self-management competencies has become paramount for healthcare and assisted living applications. Many works have applied self-* to enhance such quality attributes, which mainly use experimentation as a matter of validating their approach [123]. Yet, evaluating the contributions of self-adaptive systems may raise particular challenges due to the specifics of these systems with uncertainty during operation. Although there are proposals available in the healthcare domain and the engineering of safety and mission-critical adaptive systems, to the best of our knowledge, there is no particular exemplar of a CPS based on control-theoretical principles. The study reported in paper B is motivated by the lack of such an exemplar.

Our long-term goal with this study is to support the evaluation of control-theoretic software self-adaptation of CPSs with safety-critical constraints.

To address such a goal, we have made an exemplar of the Body Sensor Network (BSN) publicly available, supporting a set of scenarios and adaptation policies to cope with distinct classes of uncertainty. The exemplar comes equipped with a control-theoretic software controller that can be tuned or exchanged by other controllers. The exemplar enables the composition of other adaptation scenarios with a component that injects faults at runtime, simulating disturbances, alongside a comprehensible set of configurations of components representing the distinctive parts of CPSs.

Paper B contributes an exemplar for reproducible experimentation for validating the design and implementation of adaptation in CPSs prone to uncertainty (RQ1).

1.3.3 Paper C

A traditional approach to realize self-adaptation in software engineering is using feedback loops. Additionally, providing guarantees for a self-adaptive system is done by testing it against its requirements, or formally, by specifying formal properties that are verified against system models, e.g., model checking. By such means, engineers can collect assurance cases certifying that the software system works under a set of assumptions. On the other hand, control-theoretic adaptation design promotes guarantees by construction, leveraging the method as the driver for assurance provision. Yet, teams of engineers from different fields cooperate in the design of CPSs. For instance, control experts design a controller using domain-specific tools (e.g., MATLAB or Simulink). The control model is transformed into low-level code, which goes through customizations and is finally integrated into a bigger system. In this engineering process, the guarantees leveraged by the control experts in the design need to be reassured after customization. Yet, it is unclear how and to what extent traditional approaches to providing assurances in software engineering consider properties from control theory. The study reported in paper C is motivated by the lack of understanding of the interplay between the two disciplines' properties.

The long-term goal of this study is to open a pathway between control theory and software engineering enabling an interchange of notations, tools, techniques for the benefit of diverse teams of engineers designing CPSs.

In a first step to reach our research objectives, we aim to explore the

fundamental properties of control theory and software engineering. Paper C follows a bottom-up approach to map control theory and software engineering properties using property specification patterns [89, 90]. Our approach consists of a Simulink specification of a control design inspired in Ferrari Scuderia (F1) [124], with safety guarantees. We use a reliable transformation from the control model into C code and check whether the code exhibits the safety behavior encoded using specification properties. We map the stability property (from CT) to liveness property (from SE).

Paper C contributes a vision on how to map control theory and software engineering properties to benefit the engineering of resilient CPS (RQ2).

1.3.4 Paper D

Autonomous Driving Systems (ADS) can get complex due to the interface between vehicle-vehicle or vehicle-x, i.e., x stand for any other cyber-physical element that might interact with the vehicle. Yet, ADS must be vigorously tested before deployment in the traffic. When it comes to testing in realistic scenarios, two approaches are most common, (i) using domain-specific languages (DSLs) to expressing behavior and interaction, and (ii) using catalogs of pre-defined trajectories that emulate the adversary vehicles. Current DSLs for specifying vehicle behavior and interaction are limited to expressing what interacting vehicles must do and when they must do it, but not how to behave. Trajectory emulation of adversary vehicles is more realistic but is limited to static scenario testing. It is imperative to account for dynamicity in the driving environment. The study reported in paper D is motivated by the limitations of current DSLs and trajectory models for specifying realistic driving scenarios.

The long-term goal of this study is to enable scenario-based testing of realistic and complex autonomous driving behavior, fostering an expressive framework for validating complex cyber-physical systems.

In paper D, we propose Geoscenario SDV (Simulated Driving Vehicle) to unite DSLs and trajectory models in a model for expressing and testing autonomous driving vehicles to address our research objectives. The model disposes of three-layers, behavior, maneuver, and execution. The behavior layer coordinates the vehicle's behavior at the level of maneuvers, i.e., cut-in, overtake. The maneuver layer is responsible for the actual motion and contains a set of maneuvers synthesizing trajectory models based on the road, surrounding actors, and expected driver's behavior. The execution layer is responsible for the model simulation.

Uniting DSLs technology and trajectory models, Geoscenario SDV promotes expressiveness of complex driving scenarios and precision in motion synthesis. We validate the expressiveness of our model by generating an official set of pre-crash scenario topology from NHTSA. We also compare the accuracy of our generated trajectories, in maneuver level, with data collected from real vehicles in the traffic. Furthermore, we deployed one of the NHTSA scenarios in a real ADS testing environment where an autonomy stack is used as the subject system. Finally, we evaluate the scalability of our model by instantiating increasingly denser traffic scenarios.

Paper D contributes a model for complex and realistic scenario-based testing leveraging dynamic scenarios and uncertainty to validate the behavior

of complex cyber-physical systems (RQ3).

1.4 Contributions and Discussion

These research contributions are composed of engineering activities and artifacts, see Figure 1.4. The engineering activities are the specification, adaptation, verification, and validation of cyber-physical systems. The produced artifacts are goals and property specifications, an adaptive system, and assurance cases. The remainder of the section details the respective activities and artifacts, binding the research questions to the papers and their contributions.

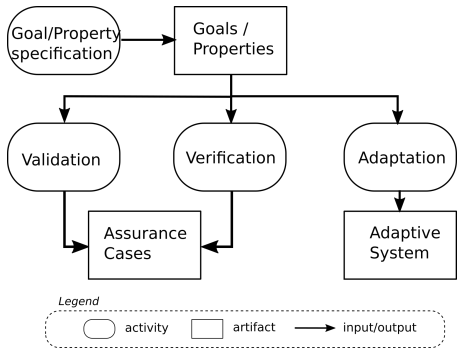

Figure 1.4: Overview of the contributions. Remark: this figure was assembled by uniting the contributions to this thesis from a holistic perspective of the work. The validation of how the elements play together will be done in future work.

RQ1: How to design efficient adaptation for CPS operating in the face of uncertainty?

Efficiently operating in the face of uncertainty demands seamless adaptation to changes in the environment or system itself. An efficient adaptation encompasses the correctness and timeliness of the adaptation choice, and both may violate the system's goals. Using rigorous analysis methods provide guarantees of correctness but stumbles upon large adaptation spaces. Control theory can be used to design adaptation with rigorousness, and machine learning methods can be used to overcome time constraints.

We propose the intertwining between control-theoretical design and machine learning for efficient adaptation of CPS operating in the face of uncertainty. As described in paper A, the approach takes as input a control model (i.e., PID controller), a prototype of a cyber-physical system alongside a model of its quality attributes, and the desired control properties. The three artifacts are composed and simulated, generating a set of execution traces matching possible control parameters and resulting software quality. We feed a learning pipeline with the execution traces and constraints from the control model. The pipeline's output is a configuration of control parameters that guide the system into adapting to faults emerging during runtime. Finally, we integrate the control model, parameters, and the target system into a control system, namely the resilient cyber-physical system. Such adaptation can be re-run for unpredicted

faults emerging during runtime. Results show that the learning pipeline finds a nearly optimal solution (i.e., stable, minimum accuracy error, and overshoot) in around 10.37 seconds for 50 control parameter configurations. The means for simulation and exploration of other uncertainty scenarios, learning pipelines, control models is described in paper B.

We answer RQ1 with the statement that CPSs can be profitably engineered and designed by intertwining control theory and artificial intelligence. Efficient adaptation can be obtained by exploiting control-theoretic design for promoting correctness and artificial intelligence as curve fitting method for reducing the adaptation space.

RQ2: How to formally verify whether the adaptation complies with control-theoretic specifications?

Introducing control-theoretic components and learning components in the adaptation loop hinders the verifiability of the control system. These components are black-box, and verifying their behavior depends on the integration with the controlled system. After integrating the control model into the larger system, we might think to use methods such as model checking to verify the control system against a specification encoding desired properties. However, the properties checked at the software level, e.g., safety, reliability, and liveness differ from properties used in the specification for control-theoretic design, such as stability, settling time, and overshoot. In this context, it is unclear whether and how properties specified in control-theoretic notations are checked in software. Understanding which properties are enough to cover control-theoretic specifications demands a (bi-directional) mapping from CT and SE properties.

In this direction, we contribute a bottom-up approach to mapping the control specifications described in CT notation into a notation commonly utilized in software verification, i.e., LTL. Our approach exploits property specification patterns to model CT properties in temporal logic. Once CT properties are specified in temporal logic, it is possible to compare SE and CT properties and create a mapping among them. Paper C details this process and the specification and checking of stability, and indicates a mapping between stability and liveness properties.

We answer RQ2 by stating that formally verifying software controllers against control-theoretic specifications depends on using a software notation for expressing such properties. Consequently, it requires a mapping between properties that are currently verified by existing tooling and the control-theoretical properties. In summary, our work contributes towards creating a mapping between SE and CT properties, thus enabling the formal verification of adaption against control-theoretic specifications.

RQ3: How to validate large and complex cyber-physical systems operating in realistic scenarios?

Cyber-physical systems operating in the real world must handle complex iterations with humans and the environment. For instance, automated driving vehicles often face unpredictable scenarios that must conform to traffic regulations and industry standards. Testing autonomous driving systems requires simulating a wide range of realistic scenarios to ensure safety. Such scenarios must reflect how these dynamic interactions between humans and the subject system unfold in real traffic. Current models for vehicle simulation are either macroscopic [125] or microscopic [126] models. In either way, these models

lack enough detail to simulate complex interactions or controllability of the scenario execution, questioning whether and how to design models for effectively capturing realistic scenarios.

We contribute a Domain-Specific Language (DSL) that unites high-level and low-level modeling, constrained by the vehicle dynamics and generates a behavioral model of simulated driving vehicles to be used in realistic scenario-based testing. The DSL leverages expressiveness and reuse, relying on behavior trees for high-level modeling. Also, it encompasses accuracy concerning realistic human driving behavior in the lower-level maneuver modeling. Results show that the DSL can be used to model realistic scenarios from the NHTSA regulator, accounting for the multiple light-vehicle crashes in traffic. Furthermore, the simulated-driver vehicles seamlessly integrate with state-of-the-art simulators such as Carla [112] for scenario-based testing.

We answer RQ3 with the statement that the validation of large and complex cyber-physical systems operating in realistic scenarios requires to combine testing with simulation, which is able to offer an abstraction of the real world in which CPSs are required to operate. In the context of autonomous vehicles, we united high-level control with low-level mathematical trajectory models in a domain specific language, and we used the DSL to generate test cases for autonomous vehicles operating in realistic scenarios.

1.5 Conclusion and Future Work

This research aims to support self-adaptation, verification, and validation for resilient CPSs. We used design science to fulfill our research objective. Firstly, we obtained efficient adaptation exploiting control-theoretic design for promoting correctness and artificial intelligence as a curve fitting method for reducing the adaptation space. Secondly, we contribute towards creating a mapping between software engineering and control-theoretic properties, thus enabling the formal verification of adaption against control-theoretic specifications. Finally, we unite high-level control with low-level mathematical trajectory in a domain-specific language for generating realistic scenarios. We have published all data used in the papers along with scripts for data analysis and visualization, implementation, and instructions on how to reproduce the experiments (when applicable).

We intend to support the adoption of self-adaptation in real-world CPSs by collecting assurance cases in their control software layer. As we learned during this thesis, efficient self-adaptation will contain artificial intelligence components (ML, NN, RL). Thus, looking into verification of AI-enabled self-adaptive components is a natural next step for providing the required guarantees [127]. We also learned that control theory experts typically work separately from software engineers to design and implement control software. Reflecting on the gap between the properties ensured in both domains. We believe that mapping these properties will build a fundamental basis for the collaboration between practitioners and researchers from these domains, for example, by supporting the checking of control software, which also invariantly benefits the verification of self-adaptive systems. Typical control-theoretic properties are defined in the observed variable of the output signal. We intend

on using the templates for signal-based temporal logic elaborated and trace-checked in recent studies [98, 99]. Finally, we learned that the distance between testing with simulation and the real world can be significant, being one of the worries of roboticists on a daily basis [68]. Difficulting the implementation of meaningful tests for checking CPS. We plan to investigate world-in-the-loop simulation [115], which can provide CPSs with a mixed-reality that integrates sensing data from simulation and the real world.