Work-In-Progress: a DSL for the safe deployment of Runtime Monitors in Cyber-Physical Systems

Giann Spilere Nandi, David Pereira, José Proença, Eduardo Tovar
CISTER - Research Centre in Real-Time & Embedded Computing Systems, Portugal
Email: {giann,drp,pro,emt}@isep.ipp.pt

***Abstract*—Guaranteeing that safety-critical Cyber-Physical Systems (CPS) do not fail upon deployment is becoming an even more complicated task with the increased use of complex software solutions. To aid in this matter, formal methods (rigorous mathematical and logical techniques) can be used to obtain proofs about the correctness of CPS. In such a context, Runtime Verification has emerged as a promising solution that combines the formal specification of properties to be validated and monitors that perform these validations during runtime. Although helpful, runtime verification solutions introduce an inevitable overhead in the system, which can disrupt its correct functioning if not safely employed. We propose the creation of a Domain Specific Language (DSL) that, given a generic CPS, 1) verifies if its real-time scheduling is guaranteed, even in the presence of coupled monitors, and 2) implements several verification conditions for the correct-by-construction generation of monitoring architectures. To achieve it, we plan to perform statical verifications, derived from the available literature on schedulability analysis, and powered by a set of semi-automatic formal verification tools.**

***Index Terms*—runtime verification, cyber-physical systems, DSL, safety, mode change**

I. Introduction

Cyber-Physical Systems are physical and engineered embedded systems whose operations are monitored, coordinated, controlled, and integrated by one or more computing and communication devices [24]. Typically, these systems are composed of a network of devices that sense its environment and perform physical activities controlled and monitored by local or distributed software [20]. Herein we focus on safety-critical CPS, a sub-group that complies with high safety standards to avoid failures that could cause loss of lives, property damage, or catastrophic environment accidents.

The automotive domain is a notable example of a safety-critical CPS that also serves as a good use-case for this work, due to its current trend of adding value to its software functions rather than focusing purely on the car's mechanics [21]. Examples of it are services like advanced driver-assistance systems (ADAS), entertainment, and autonomous driving. Being able to guarantee that such complex software architectures do not fail upon deployment can quickly become a considerable challenge as these systems commonly face emergent partially known, or even unknown, events. Providing such guarantees is especially tricky in the context of CPS, as there is a need for analyzing and verifying both functional and non-functional properties of these systems [1].

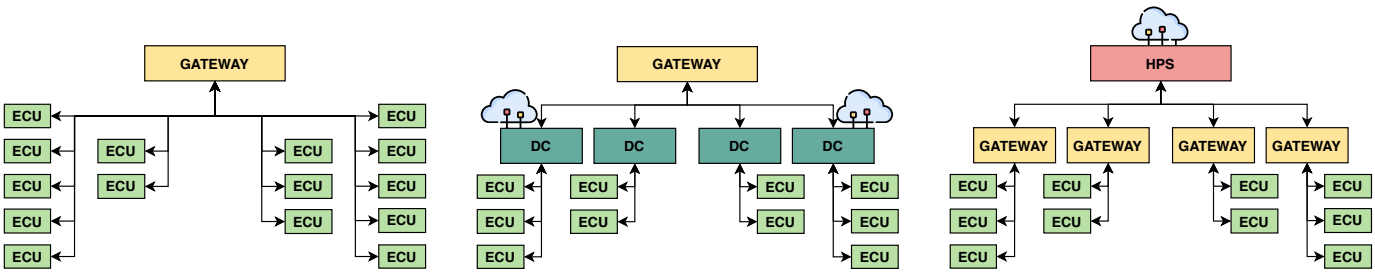
Although it remains challenging to prove the correctness of the numerous aspects of CPS, various tools that apply formal methods (rigorous logical and mathematical techniques) are currently available to specify, develop, and verify a wide range of properties. Such techniques can offer proofs or evidences about specific aspects of the system and reveal inconsistencies, ambiguities, and incompletenesses that are generally overlooked by traditional testing procedures.

Runtime Verification (RV) is currently one of the most promising formal approaches for verifying CPS' correctness [3], [27]. RV techniques use monitors [22] that are generated and orchestrated within a software architecture. Monitors are coupled to a target system to observe its execution and identify, during runtime, aspects that could not be checked during the design-phase, or errors that were not proven to be absent via static verification methods [6].

When making use of monitors, one has to ensure that they do not affect the functional and the safety non-functional requirements of the system (e.g., task scheduling [18]). To address this, we propose the creation of a DSL that can express both functional and non-functional properties of systems while also supporting the concept of mode changes. The envisioned DSL will consider, at the core of its design, the correct-by-construction generation of monitors given a formal specification and the compliance of the monitoring solutions with the safety requirements of the target system.

The envisaged DSL describes criteria over the properties that the monitors should analyse, and over the timing restrictions for deploying tasks and monitors. These criteria will be analyzed by schedulability analysis algorithms present in the literature and external, (semi-)automatic formal verification tools such as *model checkers* (e.g., mCRL2, UPPAAL), *SMT solvers*, *proof-assistants* (e.g., Coq), and *security-focused* formal verification tools (e.g., ProVerif). We are currently working on the DSL syntax and experimenting with some of

This work was partially supported by National Funds through FCT/MCTES (Portuguese Foundation for Science and Technology), within the CISTER Research Unit (UID/CEC/04234); also by the Norte Portugal Regional Operational Programme (NORTE 2020) under the Portugal 2020 Partnership Agreement, through the European Regional Development Fund (ERDF) and also by national funds through the FCT, within project NORTE-01-0145-FEDER-028550 (REASSURE); also by COMPETE 2020 under the PT2020 Partnership Agreement, through ERDF, and by national funds through the FCT, within project POCI-01-0145-FEDER-029946 (DaVinci); also by FCT within project ECSEL/0016/2019 and from the ECSEL Joint Undertaking (JU) under grant agreement No 876852. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Austria, Czech Republic, Germany, Ireland, Italy, Portugal, Spain, Sweden, Turkey.



(a) Distributed E/E Architecture (b) Domain Centralized E/E Architecture (c) Vehicle Centralized E/E Architecture

Fig. 1: Expected evolution of automotive E/E architectures.

these tools and techniques for validation purposes.

We motivate our work by modern electric/electronic architectures present in the automotive domain. However, our approach is generic, as it provides support to safely deploy runtime verification solutions also applicable in other domains with similar timing restrictions such as critical systems in the domains of healthcare, industrial robotics, and railway.

The rest of this paper presents our motivation together with a brief use-case contextualization in Section II, the presentation of our DSL and its upcoming implementation challenges in Section III, and the paper conclusion in Section IV.

II. Contextualization and Motivation

Enhancements to CPS through the use of software solutions have drastically increased during the last decade. A clear example of this is the automotive industry, which is currently transitioning away from the traditional signal-based electric and electronic (E/E) architecture, purely based on distributed electronic control units (ECUs), to a more contemporary approach using a mix of distributed ECUs and centralized high-performance units [9]. Such a change is partially motivated by the inability of the traditional approach to support the high computation and communication demands of more advanced software-based car functions [10], the desire to reduce material production costs (traditional wiring represents both the third most expensive and heaviest hardware component of a car [19]), and the necessity of interacting with services and receiving Over The Air updates [4].

Bucaioni et al. [5] classify automotive E/E architectures into three groups:

Distributed E/E architectures ($<$ 2019, Fig. 1(a)), date back to the '90s and were composed of 30 to 100+ distributed function-specific ECUs that communicate using a single gateway. Such an approach creates a communication bottleneck on the gateway and drastically increases the complexity of network management.

Domain Centralised E/E architectures (2019 $\rightarrow$ 2023, Fig. 1(b)) reduce the number of distributed ECUs and increase the computational power of some units to cope with the increased software complexity. They attempt to separate the vehicle's ECUs into specialized domains that perform domain-specific functions, e.g., powertrain, infotainment, and chassis. Each domain contains a high performance computing unit named Domain Controller (DC) capable of managing its associated ECUs and communicating with other DCs [12]. This approach concentrates computation and communication within each domain while also allowing for inter-domain data exchange and leveraging cloud-based services.

Vehicle-Centralized E/E architectures ($>$ 2023, Fig. 1(c)) are seen as the future of automotive E/E architectures, further further centralizing low latency in-car computation while also integrating the idea of continuously interacting with cloud-based services (Fig. 1(c)). Vehicle centralized architectures use multi/many-core high-performance servers (HPS) to perform massive data processing while sending and receiving information through various zone controllers (gateways), which support both legacy and newer technologies.

Newer E/E automotive architectures still face several challenges for their real-world implementation and adoption [17], [19]. Although not explicitly tailored for the automotive domain, our work contributes to overcoming two crucial real-world obstacles the newer generation of cars face: under-utilization of resources and safety guarantees. We approach the former by supporting the use of mode changes [25] and the latter by proposing a correct-by-construction generation of RV solutions with safety guarantees regarding real-time properties.

While the concept of mode changes allow systems to optimize the use of its resources consonant to its current state, RV solutions can help check, during runtime, if a system behaves according to its specification. RV is of great importance in the context of CPS as it helps to verify aspects that are not verifiable with statically defined techniques either by the nature of the problem or due to the issue of state explosion. With these concepts in mind, we aim at creating a DSL that supports both RV and mode changes while abstracting the formal aspects of the correct and safe deployment of these solutions in any CPS architecture that complies with a set of constraints that are further detailed in the next section.

III. DSL Requirements and Challenges

Our work consists of exploring the idea of a DSL capable of deploying correct-by-construction RV architectures in CPS while also abstracting the burden of performing several formal verifications semi-manually. On top of centralizing domain

knowledge, DSLs are also known for improving productivity and enhancing validation and verification procedures during system design [26]. These two aspects are of crucial importance in our case as we aim at minimizing the number of errors when deploying RV solutions while also reducing the time to deploy CPS. This is achieved by delegating the formal verification of the system to a compiler that includes a set of semi-automatic formal verification tools. Summarising, our idealized DSL should be able to:

- express both functional and non-functional CPS properties to be verified by the monitors;
- verify, statically, the schedulability of a target system and its associated set of monitors by considering a set of supported scheduling algorithms, hardware configurations, operating modes, and the possible mode transitions that the system could go through. In other words, we verify if the coupling of monitors does not impact the temporal safety of the target system execution;
- generate correct-by-construction monitoring architectures that exploit 3rd party formal verification tools.

A. Specifying the architectures with mode changes

On top of motivating and contextualizing our proposed work, the E/E architectures of Fig. 1(b) and Fig. 1(c) are also representative examples of the systems that we intend to support. However, the applicability scope of this work generalises these particular architectures.

The structuring elements of our envisioned DSL (Listing 1) are nodes and operating modes. The nodes represent the top-level software components (in the example considered, DCs and ECUs), and the behavior of is specified by one or more operation modes that manage the parametrization of the computing elements of the language. Each computing elements is classified as being either a system **task** or **monitor**.

Nodes can thus operate in different modes (in the example, m_1 and m_2), one at a time. While a **monitor** is described as tuple with its worst-case execution time (**WCET**), minimum inter-arrival time (**T**), and relative deadline (**D**), a **task** inherits all of these parameters and also includes its fixed priority.

Each **node** has an assigned **mode** at runtime, defining which tasks can be executing and its respective schedulability parameters. A transition between two modes is triggered by a mode-change request, which consists of graciously terminating tasks from the previous mode while starting tasks from the new mode without compromising deadlines, resources, nor breaking dependencies.

At its core, our DSL aims to concisely describe the assignment of nodes, tasks, modes, schedulability parameters, and intra-dependencies among tasks. Further extensions to this DSL include the support for hierarchical nodes, where tasks can be assigned to different nodes at runtime, and the support to a fine control over who can request mode changes (e.g., DC issuing requests to its associated ECUs Fig. 1(b)). Many variations of these topics have been investigated in the literature (e.g. [14], [23]).

Listing 1: DSL syntax – initial design efforts

```
node DC_1(scheduler = rm, cores = 2, alloc_policy = global){
  mode m_1 {
    task tsk_1{T = 10 ms, D = 10 ms, WCET = 5 ms};
    task tsk_2{T = 5 ms, D = 3 ms, WCET = 2 ms};
    monitor mon_1{T = 5 ms, D = 3 ms, WCET = 2 ms};
  }
  mode m_2 {
    task tsk_1{T = 30 ms, D = 20 ms, WCET = 15 ms};
    task tsk_2{T = 5 ms, D = 3 ms, WCET = 2 ms};
    monitor mon_1{T = 3 ms, D = 2 ms, WCET = 1 ms};
  }
}
node ECU_1 (scheduler = rm, cores = 1){
  mode m_1 {
    task tsk_3{...}; task tsk_4{...}; task tsk_5{...};
  }
  mode m_2_ {
    task tsk_5{...}; task tsk_6{...}; monitor mon_2 {...};
  }
}
```

B. Scheduling Guarantees

A key concern of the proposed underlying tools are safety requirements. Our analysis tools will address these in two ways. Firstly, they will verify all nodes' schedulability considering multiple modes of operation and runtime transitions between modes. Secondly, they will calculate the overhead caused by monitoring tasks in all nodes, helping to configure the scheduler, and also targeting multiple modes of operation and transitions among them.

To keep the supported system architecture as generic as possible regarding hardware and scheduling policies, while also being in line with industrial practices, we aim at supporting fixed-priority scheduling for both single-core and multiprocessor units (i.e., multi- and many-core). Supporting many hardware architectures implies performing schedulability analysis of several scheduling and allocation policies on single-core and multiprocessors (e.g., global scheduling, partitioned scheduling, and semi-partitioned scheduling) [2], [11], [15], [16]. While the referred works present solutions for the schedulability analysis of specific system architectures and scheduling policies, we aim at centralizing their results in the form of a DSL compiler.

Listing 1, although simplistic, showcases some of the challenges in dealing with mode changes, not including functional aspects of the tasks and monitors. For **node** `DC_1`, the DSL compiler would calculate the schedulability analysis of **mode** `m_1`, **mode** `m_2`, and possibly (application dependent) the transitions between `m_1` to `m_2` and `m_2` to `m_1`. In this example, the number of tasks is kept the same, although some parameters are changed consonant to the executing **mode**. The case portraited by `ECU_1` is a bit different, as although parameters are kept the same, tasks that are relevant to **mode** `m_1` are not relevant to `m_2`. Nodes' parameters, such as the number of **cores** and employed **scheduler** (`rm` = rate monotonic), are fixed to each **node** and will define which schedulability analysis will be used to verify its feasibility.

C. Runtime Verification

Apart from having the overhead introduced by monitors checked concerning safety guarantees, we aim to develop a

DSL capable of expressing which functional (e.g., calculations results [13], ordering of events, access restrictions [7]) and non-functional (e.g., temporal properties [8]) aspects of the system are to be verified against a given formal specification.

Regardless of the end-objective of what needs to be monitored, user-specified monitors would have its correctness checked by a combination of external, (semi-)automatic formal verification tools like model checkers (e.g., mCRL2, UPPAAL), SMT solvers, proof-assistants (e.g., Coq), and security-focused (e.g., ProVerif) formal verification tools. By doing so, we guarantee that such monitors are correct with respect to its specification, allowing a correct-by-construction monitoring architecture on the target system.

D. Current State of Progress

Our current efforts are distributed into two distinct domains: DSL syntax and formal verification tools. The DSL syntax tries to balance the trade-off between completeness and generality, and the verification tools try to analyze the right abstractions to provide useful feedback. These two aspects are essential to maintaining the domain knowledge present while still covering a broad range of system architectures.

We have been implementing and validating schedulability analyses present in the literature (for instance [2] and [15]) with the objective of porting them to a compiler that will complement the proposed DSL. We have also experimented with tools like mCRL2, Coq, Verdi, and ProVerif to understand its usefulness to our work and plan to keep on experimenting with other tools to expand our set of validation tools.

IV. Conclusion

This work presented the initial concepts and ideas behind a DSL proposal for the correct deployment of RV solutions in the scope of CPS. With the intuit to contextualize and motivate, we built a use case around the automotive domain to serve as an example of the type of systems compatible with our ideas. At the core of our work proposal is the problematic of deploying monitoring solutions in such a way that: 1) safety aspects are considered during design time; 2) coupled monitors do not affect the functional and the safety non-functional aspects of the target system; 3) the code generated for the monitoring architectures and monitors is correct-by-construction. To address these concerns, we leverage the available literature on schedulability analysis to perform static scheduling verifications and use third-party formal tools to guarantee the correctness of user-specified monitors.

References

[1] Mehrnoosh Askarpour, Carlo Ghezzi, Dino Mandrioli, Matteo Rossi, and Christos Tsigkanos. Formal methods in designing critical cyber-physical systems. In *From Software Engineering to Formal Methods and Tools, and Back*, pages 110–130. Springer International Publishing, 2019.

[2] Hyeongboo Baek, Kang G. Shin, and Jinkyu Lee. Response-time analysis for multi-mode tasks in real-time multiprocessor systems. *IEEE Access*, 8:86111–86129, 2020.

[3] Ezio Bartocci, Yliès Falcone, Adrian Francalanza, and Giles Reger. Introduction to runtime verification. In *Lectures on Runtime Verification*, pages 1–33. Springer International Publishing, 2018.

[4] Stefan Brunner, Jurgen Roder, Markus Kucera, and Thomas Waas. Automotive e/e-architecture enhancements by usage of ethernet TSN. In *2017 13th WISES*. IEEE, June 2017.

[5] Alessio Bucaioni and Patrizio Pelliccione. Technical architectures for automotive systems. In *2020 IEEE International Conference on Software Architecture (ICSA)*. IEEE, March 2020.

[6] Ian Cassar, Adrian Francalanza, Luca Aceto, and Anna Ingólfsdóttir. A survey of runtime monitoring instrumentation techniques. *Electronic Proceedings in Theoretical Computer Science*, 254:15–28, August 2017.

[7] Vadim Cebotari and Stefan Kugele. On the nature of automotive service architectures. In *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*. IEEE, March 2019.

[8] André de Matos Pedro, Jorge Sousa Pinto, David Pereira, and Luís Miguel Pinho. Runtime verification of autopilot systems using a fragment of MTL- $${\int }$$ $\int$. *International Journal on Software Tools for Technology Transfer*, 20(4):379–395, August 2017.

[9] Darren Buttle Detlef Zerfowski. Paradigm shift in the market for automotive software, nov 2019.

[10] Mahdi Dibaei, Xi Zheng, Kun Jiang, Sasa Maric, Robert Abbas, Shigang Liu, Yuexin Zhang, Yao Deng, Sheng Wen, Jun Zhang, Yang Xiang, and Shui Yu. An overview of attacks and defences on intelligent connected vehicles, 2019.

[11] Zhishan Guo, Kecheng Yang, Sudharsan Vaidhun, Samsil Arefin, Sajal K. Das, and Haoyi Xiong. Uniprocessor mixed-criticality scheduling with graceful degradation by completion rate. In *2018 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, December 2018.

[12] Waldemar Haas and P. Langjahr. Cross-domain vehicle control units in modern e/e architectures. In *Proceedings*, pages 1619–1627. Springer Fachmedien Wiesbaden, 2016.

[13] Donal Heffernan, Ciaran MacNamee, and Padraig Fogarty. Runtime verification monitoring for automotive embedded systems using the ISO 26262 functional safety standard as a guide for the definition of the monitored properties. *IET Software*, 8(5):193–203, October 2014.

[14] T.A. Henzinger, B. Horowitz, and C.M. Kirsch. Giotto: a time-triggered language for embedded programming. *Proceedings of the IEEE*, 91(1):84–99, January 2003.

[15] Wen-Hung Huang and Jian-Jia Chen. Techniques for schedulability analysis in mode change systems under fixed-priority scheduling. In *2015 IEEE 21st International Conference on Embedded and Real-Time Computing Systems and Applications*. IEEE, August 2015.

[16] Wen-Hung Huang and Jian-Jia Chen. Utilization bounds on allocating rate-monotonic scheduled multi-mode tasks on multiprocessor systems. In *Proceedings of the 53rd Annual Design Automation Conference on - DAC '16*. ACM Press, 2016.

[17] Rolf Johansson, Rikard Andersson, and Markus Dernevik. Enabling tomorrow's road vehicles by service-oriented platform patterns. 2018.

[18] Muhammad Taimoor Khan, Dimitrios Serpanos, and Howard Shrobe. A rigorous and efficient run-time security monitor for real-time critical embedded system applications. In *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*. IEEE, December 2016.

[19] Jochen Klaus-Wagenbrenner. Zonal ee architecture: Towards a fully automotive ethernet–based vehicle infrastructure, sep 2019.

[20] Edward Lee. Cyber-physical systems - are computing foundations adequate? 01 2006.

[21] Edward Marmounie Marc Bellanger. Service oriented architecture: impacts and challenges ofan architecture paradigm change. *10th European Congress on Embedded Real Time Software and Systems*, jan 2020.

[22] Ramy Medhat, Borzoo Bonakdarpour, Deepak Kumar, and Sebastian Fischmeister. Runtime monitoring of cyber-physical systems under timing and memory constraints. *ACM Trans. Embed. Comput. Syst.*, 14(4):79:1–79:29, October 2015.

[23] Claire Pagetti, Julien Forget, Frédéric Boniol, Mikel Cordovilla, and David Lesens. Multi-task implementation of multi-periodic synchronous programs. *Discrete Event Dynamic Systems*, 21(3):307–338, May 2011.

[24] R. Rajkumar, I. Lee, L. Sha, and J. Stankovic. Cyber-physical systems: The next computing revolution. In *Design Automation Conference*, pages 731–736, June 2010.

[25] J. Real and A. Crespo. Mode change protocols for real-time systems: A survey and a new proposal. *Real-Time Systems*, 26, March 2004.

[26] M. Voelter. *DSL Engineering: Designing, Implementing and Using Domain-specific Languages*. CreateSpace Independent Publishing Platform, 2013.

[27] C. Watterson and D. Heffernan. Runtime verification and monitoring of embedded systems. *IET Software*, 1(5):172–179, October 2007.