2023 Design, Automation & Test in Europe Conference (DATE 2023)
Special Initiative "Autonomous Systems Design"

Autonomy-driven Emerging Directions in Software-defined Vehicles

Unmesh Bordoloi*, Samarjit Chakraborty†, Markus Jochim‡, Prachi Joshi‡, Arvind Raghuraman*, and S. Ramesh‡
*Siemens EDA, USA, †University of North Carolina at Chapel Hill, USA, ‡General Motors, USA

***Abstract*—Over the past two decades, the volume of electronics and software in cars have grown tremendously. But this growth has also resulted in hardware and software architectures that are proving to be a bottleneck for further innovation and efficient design flows, especially when implementing compute-intensive functions necessary for modern autonomous features. For example, centralized architectures that are driven by the use of more powerful processors result in higher sensor-to-actuator delays. Similarly, timing uncertainties increase as signal-based in-vehicle communication is being replaced by more dynamic service-oriented communication architectures. Finally, the increasing volume of software running on powerful multicore ECUs is making timing analysis, including WCET estimation, to be very complex. As a result, timing estimates, when safe, are very pessimistic, which makes efficient implementations to be difficult. In this position paper, we outline some of these emerging challenges and discuss potential solutions.**

***Index Terms*—Software-defined vehicles, autonomous vehicles**

I. Introduction

There has been a tremendous growth in the volume of software in modern cars, and most emerging autonomous features are now software-based, resulting in the term "*software-defined vehicles*" (SDVs). This has necessitated techniques for scheduling [1]–[3] and management of such software, especially to ensure their real-time behavior. At the same time, the need for certifying the functional and timing correctness of such software has led to work on testing [4], [5] and formal verification of automotive control software [6]–[9]. For this, control-theoretic techniques and reachability analysis have been used for safety verification [10]–[14]. But both verification and efficient implementations of automotive control software has been complicated by evolving "zone-based" automotive architectures that result in large delays between sensing and actuation. Service-oriented in-vehicle communication, in contrast to classical signal based ones have also resulted in higher timing uncertainties. Therefore, ensuring timing safety results in overly pessimistic timing estimates and inefficient implementations. Alternatively, the controllers need to be redesigned to mitigate the impact of large delays and timing uncertainties. Several studies have addressed this problem [15], [16], including how to synthesize delay-tolerant controllers [17], [18] and how to co-synthesize controllers and their underlying task schedules [19]–[22]. This is also related to providing timing isolation to critical control software [23], [24] and the scheduling of mixed-criticality tasks [25].

In spite of such a large volume of work in this domain, there are still a number of open challenges. We discuss some of them in this paper along with potential directions for addressing them. We first discuss some of the timing analysis challenges associated with service-oriented in-vehicle communication in SDVs. This is followed by a description of the role of digital twins in early validation and design of SVDs. Finally, we discuss how to formally verify and synthesize efficient control software in SDV architectures in the presence of timing uncertainties.

II. Service-oriented Communication in SDVs

Traditionally, exchange of data between electronic control units (ECUs) in the automotive has been established using signal-based communication. This involves statically-configured systems during the system design phase, using specific protocols such as CAN and LIN. With the automotive industry moving towards SDVs, service-oriented communication (SOC) or service-oriented architecture (SOA) is being adopted into automotive electrical architectures [26]. SOC manages and distributes the communication between ECUs dynamically, using the Ethernet protocol, thus allowing more flexibility and scalability. However, it also requires more complex software. In this section we briefly discuss the need and also the impact of service oriented communication on SDVs.

Service-oriented communication is established between communicating entities using a "publish-subscribe" mechanism, where the communicating systems discover each other over the network and then proceed to communicate, thus providing dynamism and flexibility. For example, the various sensors in the vehicle (*e.g.*, wheel speed sensor) can "publish" data and a "client," *e.g.*, the powertrain control module or the advanced driver assistance system (ADAS) module, that is interested in consuming this data can subscribe to the wheel speed sensor service and start receiving the relevant data.

A. Advantages of Service-oriented Communication

The reason behind adopting SOC in the automotive domain is to increase the update and upgrade capabilities of software, even post sale of vehicles, which is the mission of SDVs. Further, it also provides scalability since signal-based communication can become increasingly complex as the number of ECUs in a vehicle increases. SOC also allows easy integration of new ECUs and updates to the communication protocol, making it more scalable. Additionally, signal-based communication requires a direct connection between the ECUs, which can make it inflexible in terms of adding new features or changing the communication protocol. SOC, on the other hand allows a more flexible communication structure, as the communicating entities are loosely coupled.

978-3-9819263-7-8/DATE23/© 2023 EDAA

B. Middleware

One of the main advancements for SDV is the move towards IP (Internet Protocol) based communication, and as such it enables the use of higher-layer protocols such as middleware like SOME/IP (Scalable service-Oriented MiddlewarE over IP) [27] and DDS (Data Distribution Service) [28]. Therefore, a comparison of the available protocols is important from a functional and performance perspective [29]. While SOME/IP was developed specifically for automotive needs, DDS was developed by OMG (Object Management Group) for real-time applications such as defense and aerospace. Both SOME/IP and DDS are broker-less publish-subscribe protocols with service discovery and serialization mechanisms, and are integrated with Adaptive AUTOSAR specifications (SOME/IP is available with AUTOSAR Classic too). However, they have some key differences: DDS provides quality of services (QoS), whereas SOME/IP does not have QoS in its specifications. However, SOME/IP is more readily available and is more integrated with AUTOSAR vendor solutions than DDS, owing to its relevance in automotive use cases.

C. Challenges in Service-oriented Communication

While SOC can have many benefits for SDV architectures, there are also a number of challenges that must be addressed in order to effectively implement SOC in an SDV context. One of the major challenges is ensuring that the services being exchanged between different software systems are well-defined and self-contained. This requires a significant amount of design and planning to ensure that the services are structured at the right level of granularity such that they are modular and they meet the needs of the overall system. From an automotive perspective, where real-time constraints are important due to safety-critical functions, another challenge is to ensure that real-time guarantees (hard or soft) are met. Timing analysis for signal-based communication over CAN/CAN-FD has been studied and developed for decades [30], [31]. However, there is little work on timing predictability for SOC [32]. Additionally, since service-oriented communication needs additional software layers (such as a middleware: SOME/IP or DDS) they also have an impact on the resource consumption of the hardware, which is typically constrained for memory and compute power. While no significant difference has been reported in terms of performance in using the SOME/IP versus DDS, it depends on the implementation to a large extent.

Thus, moving to SOC for enabling SDVs has several advantages. However, there are also challenges in designing SOC-based architectures, which may be resolved with real-time analysis and techniques discussed later in this paper.

III. "Shift-Left" Development of SDVs

The automotive industry is in a flux as it embraces a disruption from multiple dimensions that include battery powered vehicles and higher levels of autonomy. In all of these cases, and particularly for autonomous vehicles, one trend is clear: the car of the future will be defined by software. This market for SDVs is a hyper-competitive one with high pressures to mitigate the increasingly shorter software development cycles. This is aggressively driving the so called "*shift-left*" methodology that enables software development, characterization, and verification way before silicon hardware is available. However, such software development and validation can be meaningful only if the complete system-of systems (including the environment, software, compute and the actuation) can be virtually modeled throughout the design life-cycle with appropriate levels of abstraction and fidelity [33], [34]. In this section, we discuss how integrated *Digital Twin Platforms* can model simple component systems to "system-of-systems" and be applied to SDVs.

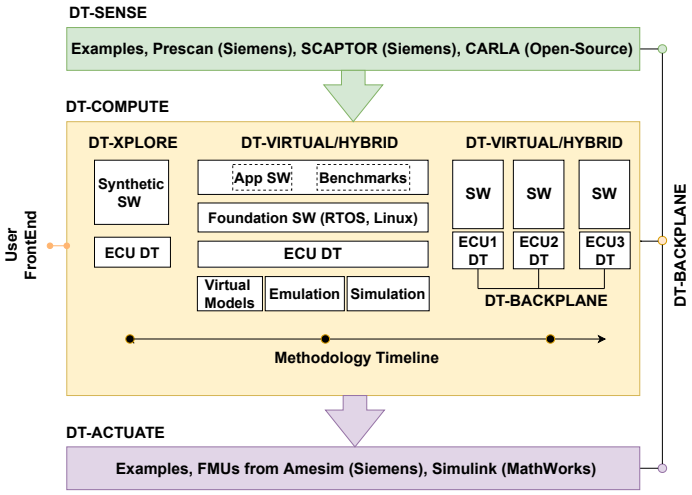

Fig. 1. Overview of Digital Twin platform for SDV, highlighting the Siemens PAVE360 solution as a case study.

A. Digital Twin Platforms for SDV

At the highest level, a Digital Twin (DT) platform for a SDV comprises of digital twins for sensing (DT-SENSE), compute (DT-COMPUTE) and actuation (DT-ACTUATE), representing the three key components of any vehicle (see Figure 1). Apart from these, an interconnect (DT-BACKPLANE) is needed to connect all the digital twins together in a synchronized fashion. Depending on the level of abstraction, each of these digital twins may be represented at different levels of fidelity, offering trade-offs between simulation speed and accuracy. Taking DT-COMPUTE as an example, we have the following options for abstraction at different points in the methodology timeline (also illustrated inside the DT-COMPUTE box in Figure 1).

1. DT-XPLORE: At the very initial stage of a new system design, typically, there is no production software or hardware available. However, system architects have initial hypotheses regarding few key properties of the system. These may include the number of software components, periodicity of the components, nature of the workload of the respective components (*e.g.*, compute bound vs. I/O bound) and so on. System architects also have knowledge about the expected hardware – regarding core types, number of cores, and their clock frequencies. Based on this limited input, DT-XPLORE auto-generates a digital twin that can (i) generate abstract software that mimics the workload, as well as (ii) a virtual hardware

platform that simulates the instruction sets. DT-XPLORE is designed in a way that allows architects to rapidly change the system properties (whether it is hardware or software) and quickly iterate on them.

2. DT-VIRTUAL: With progression of the design cycle, further decisions on hardware and silicon are made and further information on peripheral and CPU sub-systems are known. This allows engineers to create virtual platforms, typically based on SystemC/TLM2.0 [35], and models of the ECUs. In addition, the nature of software needed for the next-generation product is better known and foundation software components become available. Since complete ECU virtual models, and a software baseline that is in path to production is used, higher fidelity metrics are available for characterization of hardware and software compared to what is possible in DT-XPLORE.

3. DT-HYBRID: The next environment in this methodology timeline progression is DT-HYBRID. The core idea of the hybrid platform is to evaluate the system with both speed and fidelity. The hybrid platform allow users to run large amounts of software (SW) quickly to a point of interest on virtual models, and then run software they are most interested in accurately on hardware in RTL. When running accurately, hybrid platforms support RTL in simulation, emulation, or FPGA environments. This offers users the flexibility to strike the right balance between fidelity, speed, and compilation time in a seamless manner.

As outlined earlier, the emerging trend for SDV is a centralized compute architecture with zonal ECUs. Each ECU in such a distributed system may be represented by any of the above abstractions like DT-XPLORE, DT-VIRTUAL, or DT-HYBRID. But an interconnect DT-BACKPLANE is required to enable synchronous communication between them, as illustrated in Figure 1. Traditionally, each digital twin described above comes with its own IDE, with debug and visualization tools. For the system architect, interacting with heterogeneous tooling at different points of the methodology timeline quickly becomes onerous.

B. A Seamless Front-End for all Digital Twins

We envision a unified front-end that offers seamless tool experience across multiple digital twins for development and analysis. Such a front-end should deliver capabilities to develop, debug, and profile heterogeneous software environments across the breadth of DT-COMPUTE models, and the interactions between them. The same IDE should allow for profiling, visualization, and analysis of various software metrics. For example, activity on single/multi-core CPUs, processes/threads, memory, networking, file system, I/O, and other OS and application specific data. In addition, profiling of hardware performance counters can provide deeper insights into hardware activity.

C. Case Studies

The methodology and tooling discussed above is enabled by the Siemens PAVE360 solution [36], [37]. Figure 1 overlays it on the architecture discussed in the subsections III-A and III-B. Due to space restrictions, we do not discuss further details of the individual tools, but we provide below three examples, to illustrate how Digital Twin technologies are applied to solve industrial problems using the PAVE360 solution.

CPU	Freq (MHz)	Avg CPU Util (%)
Cortex A53x1	100	100
Cortex A53x1	1000	70
Cortex A53x2	100	43

TABLE I
EXAMPLE 1: CPU CONFIGURATIONS EXPLORED USING DT-XPLORE .

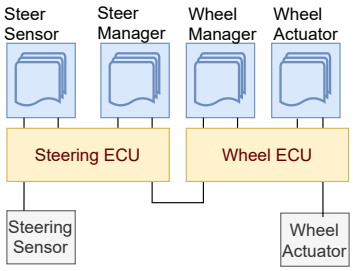

Fig. 2. Example highlighting DT-XPLORE for ECU consolidation.

Our **first example** (Figure 2) shows the application of the DT-XPLORE technology using PAVE360. As discussed earlier, the SDV trend towards zonal architectures is leading to ECU consolidation. In this example, system architects explore consolidating a Steering ECU and a Wheel ECU, along with accommodating future growth for software defined features. With limited data on software and hardware, DT-XPLORE technology allows users to define the hardware and software attributes for heir envisioned zonal ECU design using a standards based description in XML. Based on this specification from the user, the DT-XPLORE tool generates an abstract virtual model of hardware, and a synthetic software workload. The artifacts produced are representative of attributes defined by the user for the zonal ECU design. Post model generation, the tool executes the software workload on the hardware model, and generates relevant metrics of interest. The PAVE360 front-end IDE allows for visualization and analysis of such generated metrics. Results from execution of the said synthetic software workload for the zonal ECU on three different virtual hardware configurations are presented in Table I. The design objective of less than 50% average CPU utilization for the zonal ECU was achieved with a Cortex A53x2 core CPU clocked at 100 MHz.

Our **next example** (Figure 3) showcases the application of DT-VIRTUAL and DT-HYBRID to analyze containerized software applications running on top of the SOAFEE compliant software stack [38]. SOAFEE is an open software architecture for SDV that is being actively developed by an industry led consortium. Earlier in this paper, we discussed the significance of SOAs in SDVs. SOAFEE is pursuing its enablement by blending best practices from cloud-native with requirements of safety, security, and real-time requirements from the automotive domain. The automotive ecosystem is faced with the challenge of designing IPs, SoCs, and ECUs, to accommodate SOAFEE-like architectures. We integrated a reference implementation of SOAFEE based on Sokol Flex OS from Siemens into DT-

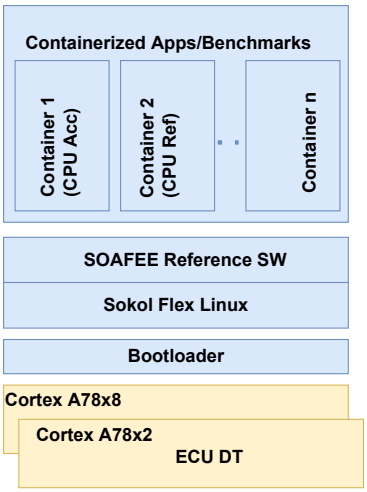

Fig. 3. Example to illustrate DT-VIRTUALand DT-HYBRID.

CPU	Latency (secs)	Avg CPU Util (%)
Cortex A78x2	5.3	100
Cortex A78x8	3.7	16

TABLE II
CPU WORKLOADS EXPLORED USING DT-VIRTUAL AND DT-HYBRID.

VIRTUAL and DT-HYBRID. We deployed containers that run the open-source armnn TFLite benchmark as workload. The workload runs with CPUAcc back-end that takes advantage of floating-point extensions, and feature a multi-threaded software implementation. Once again, the same PAVE360 front-end was used for visualization and analysis of metrics. The metrics from executing the containerized workload on two different hardware configurations are tabulated in Table II. We observe that due to thread-level parallelism in the workload there is improved performance in latency and average CPU utilization, when moving from Cortex A78x2 to a Cortex A78x8 hardware configuration.

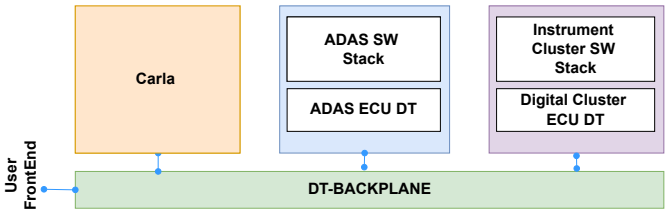

Fig. 4. Example to illustrate significance of DT-BACKPLANE.

The **final example**, illustrated in Figure 4, highlights the significance of the DT-BACKPLANE. The system being modeled here includes an ADAS ECU and an instrument cluster ECU, where both are ingesting external world data via camera and other vehicle virtual sensors. The DT-BACKPLANE synchronizes the virtual models of the two ECUs and models the communication between them. It also serves as the communication medium from the environment modeled by Carla [39], into the modeled ECU-DTs. As in our other examples, the PAVE360 front-end serves as the unified IDE for metric visualization and analysis. In an experimental run that involved analysis of software implementing PID controllers for control of steering and throttle control systems, a 300% jitter was observed in the Worst Case Execution Time (WCET) of the PID controller software routine. Further analysis using the PAVE360 front-end indicated other software applications running on the ADAS ECU that were bogging down CPU resources and inadvertently introducing the unacceptable jitter into the real-time PID control loop implemented by the ADAS software application. This condition was mitigated using appropriate thread-to-core affinity mechanisms to remove CPU contention. The details of this mitigation are being skipped for the lack of space.

These case studies show that Digital Twins are being successfully applied to "shift-left" the development of SDVs. As industrial adoption of this methodology grows, we expect to see applications of Digital Twin platforms to numerous other use cases throughout the design cycle in innovative ways.

IV. TOOLS FOR ASSURED (AND EFFICIENT) AUTONOMY

While verification and certification are well-known challenges in the design of autonomous vehicles, in this section we focus on those that arise specifically in the context of SDVs. The algorithmic core of most autonomous features consist of feedback control loops. There is a large body of literature on designing and verifying such controllers. But their *software implementation*, especially on modern automotive architectures, pose a number of unresolved challenges. The previous two sections discussed some aspects of such emerging automotive electrical and electronic (E/E) architectures, *viz.*, the move towards centralized compute architectures with zonal ECUs, and the use of service-oriented architectures. The former necessitates distributed implementations of automotive controllers, where sampled plant states from different sensors need to be communicated to a central compute unit and control signals computed by this unit are transmitted to multiple actuators. The use of service-oriented, in contrast to signal-based, communication adds considerable timing uncertainties in the communication of both sampled and actuation signals.

Further, fundamental timing analysis problems such as estimating the *worse case execution time* (WCET) of software tasks, especially for software and processor architectures found in modern vehicles, have still not been solved. In fact, solving the WCET analysis appears to involve challenges that will likely never be fully resolved [40], especially as modern processor architectures continue to become more complex and support features that optimize the average rather that the worse-case execution time.

The traditional workflow for implementing automotive controllers consist of first designing a control strategy, followed by implementing it as a software task, that is scheduled to meet the deadline determined during the controller design phase. This ensures a separation of concerns that enables control theorists and embedded systems engineers to communicate only via *deadlines* that need to be met. This flow is widely practiced in the automotive domain. However, meeting *all* deadlines, that is assumed in this design flow, is becoming exceedingly difficult for the reasons outlined above.

For WCET estimates and system-level timing analysis results for service-oriented architectures to be safe, a significant degree of overestimation gets introduced. Obtaining *safe* and *tight* WCET and timing estimates is a losing proposition, particularly for future automotive E/E architectures and software implementing machine learning based complex autonomous features. Therefore, meeting *all* task deadlines with overestimated timing values and providing verifiable guarantees will lead to overly pessimistic and infeasible implementations.

A. An Efficient and Verifiable Design Flow

In order to address the above challenge of obtaining an *efficient* and *verifiable* implementation of automotive controllers on modern zone-based architectures, we need to deviate from the traditional workflow outlined above. In particular, instead of focusing on meeting *all* deadlines, the focus should shift to satisfying "*system-level properties*" that are of relevance. Given the inherent robustness in most feedback controllers, such high-level properties may be satisfied even in the presence of certain deadline misses. In what follows, we provide an example of what such a system-level property might be, and how to synthesize an efficient and verifiable implementation satisfying this property. We finally outline the necessary tools to support this new design flow.

The system-level property we outline here is a *safety* property that may be defined as follows. Given a plant and a controller that has been designed for it, assuming "ideal" timing behavior, let τ_{nom} be a trajectory in the state space of the closed-loop system (plant + controller) when the control task meets *all* deadlines. This is referred to as the *ideal* or *nominal* behavior. Any other trajectory τ is referred to as "safe" if it is at most a specified d^{safe} distance away from τ_{nom}, under a suitably defined distance metric.

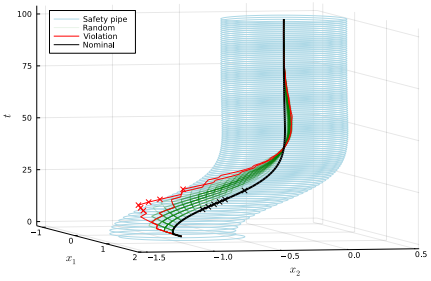

Fig. 5. System-level safety property [10].

Figure 5 [10] illustrates this safety property, where the solid black line shows the nominal behavior τ_{nom}, *i.e.*, the evolution of the system in the state space under ideal timing behavior. The light blue envelope or spiral around this nominal trajectory is a "safety margin" that encompasses all trajectories that are at most d^{safe} distance away from τ_{nom}. All the "green trajectories" (*i.e.*, the trajectories τ) are therefore safe and represent system evolutions under certain non-ideal timing behavior or deadline misses, whereas the "red" trajectories" are unsafe, *i.e.*, they represent deadline misses that are not acceptable.

The intuition here is that certain deadline miss patterns will result in a different but acceptable state space trajectory, as long as its deviation is not much from the ideal one. These allowed deadline misses – a deviation from the requirement that *all* deadlines must be met – allows a more flexible and efficient implementation. Given this definition of safety, there are two relevant questions: (i) For a given deadline miss pattern for a control task, is the safety property satisfied? and (ii) Given a set of control tasks, how to implement and schedule them to meet their specified safety properties?

The answer to (i) involves a reachability analysis, which is hard for hybrid systems. But we have developed a number of approximate reachability analysis techniques [10] using which we can compute safe over-approximations of the reachable set for a closed-loop system. We have shown that for a number of sample problems from the automotive domain, we can answer (i) effectively in a scalable manner. The answer to (ii), *i.e.*, the synthesis problem is more complex.

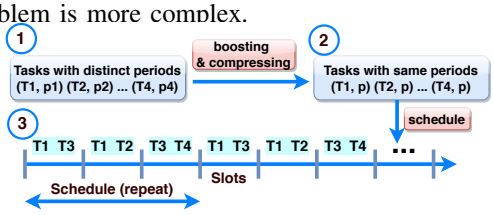

Fig. 6. Synthesizing a safe implementation.

Figure 6 provides an overview of such a synthesis. Given (1) a set of control tasks $T_1, T_2, \ldots$ with distinct sampling periods $P_1, P_2, \ldots$, the first step is to determine a *common* sampling period p by suitably increasing or reducing each period $P_1, P_2, \ldots$. This results in a new task set (2) that is then scheduled in a time-triggered manner on an ECU where time is partitioned into slots of the chosen period size p. Such a schedule (3) only allows a subset of tasks from the set $T_1, T_2, \ldots$ to be executed in each slot. The ones not scheduled *miss* their deadlines. In the example shown in Figure 6, the schedule for the task T_1 is $110110\ldots$, that of T_2 is $010101\ldots$, T_3 is $101101\ldots$, and finally that of T_4 is $001001\ldots$, where a 1 denotes the deadline being met and a 0 a deadline miss. Here, each slot is only large enough to execute at most two of the four tasks. Although there are several deadline misses, the schedule is derived in a manner that the safety property outlined above (see Figure 5) is satisfied for all the four control tasks. If *all* the deadlines were to be met, then the full task set (T_1 to T_4) would not have been implementable on a single resource and more ECUs would be necessary.

In order to synthesize the above schedule, for each control task T_i we first identify a set of valid deadline miss patterns using the solution to problem (i). We then combine such miss patterns of multiple tasks, using automata-theoretic techniques, to obtain a schedule for the full set of tasks. Such an "implementation" (or schedule) is both efficient, as well as guaranteed to satisfy the safety property outlined above. But there could be many other notions of such system-level properties and how the synthesis problem (ii) can be solved for such properties remains an open problem.

V. Concluding Remarks

In this position paper we have outlined some emerging challenges and potential solutions for implementing autonomous features in SDVs. In particular, we discussed how real-time guarantees are difficult to provide in service-oriented automotive in-vehicle communication architectures. Following this, we discussed the role of digital twin platforms to enable shorter innovation and software development cycles in the automotive domain. Finally, we outlined how timing uncertainties associated with emerging automotive E/E

architectures and service-oriented communication makes it difficult to obtain efficient and verifiable implementations of automotive controllers that form the core of many autonomous features. Here, we also discussed some potential verification and synthesis strategies that should be extended to real-life automotive architectures.

Acknowledgements: This work was partially supported by the NSF grant# 2038960.

REFERENCES

[1] S. Chakraborty, T. Erlebach, and L. Thiele, "On the complexity of scheduling conditional real-time code," in *7th International Workshop on Algorithms and Data Structures (WADS)*, ser. Lecture Notes in Computer Science, vol. 2125. Springer, 2001.

[2] L. Zhang, D. Roy, P. Mundhenk, and S. Chakraborty, "Schedule management framework for cloud-based future automotive software systems," in *22nd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2016.

[3] Z. Li, T. Chu, I. V. Kolmanovsky, X. Yin, and X. Yin, "Cloud resource allocation for cloud-based automotive applications," *CoRR*, vol. abs/1701.04537, 2017.

[4] G. Tibba, C. Malz, C. Stoermer, N. Nagarajan, L. Zhang, and S. Chakraborty, "Testing automotive embedded systems under X-in-the-loop setups," in *35th International Conference on Computer-Aided Design (ICCAD)*, 2016.

[5] P. Baumann, M. Krammer, M. Driussi, L. Mikelsons, J. Zehetner, W. Mair, and D. Schramm, "Using the distributed co-simulation protocol for a mixed real-virtual prototype," in *IEEE International Conference on Mechatronics (ICM)*, 2019.

[6] L. Ju, B. K. Huynh, A. Roychoudhury, and S. Chakraborty, "Performance debugging of Esterel specifications," in *6th International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2008.

[7] M. Broy, S. Chakraborty, D. Goswami, S. Ramesh, M. Satpathy, S. Resmerita, and W. Pree, "Cross-layer analysis, testing and verification of automotive control software," in *11th International Conference on Embedded Software (EMSOFT)*, 2011.

[8] Z. Wang, H. Liang, C. Huang, and Q. Zhu, "Cross-layer design of automotive systems," *IEEE Des. Test*, vol. 38, no. 5, pp. 8–16, 2021.

[9] L. Guo, Q. Zhu, P. Nuzzo, R. Passerone, A. L. Sangiovanni-Vincentelli, and E. A. Lee, "Metronomy: A function-architecture co-simulation framework for timing verification of cyber-physical systems," in *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2014.

[10] C. Hobbs, B. Ghosh, S. Xu, P. S. Duggirala, and S. Chakraborty, "Safety analysis of embedded controllers under implementation platform timing uncertainties," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 41, no. 11, pp. 4016–4027, 2022.

[11] A. Yeolekar, R. Metta, C. Hobbs, and S. Chakraborty, "Checking scheduling-induced violations of control safety properties," in *20th International Symposium on Automated Technology for Verification and Analysis (ATVA)*, ser. Lecture Notes in Computer Science, vol. 13505. Springer, 2022.

[12] M. Kauer, D. Soudbakhsh, D. Goswami, S. Chakraborty, and A. M. Annaswamy, "Fault-tolerant control synthesis and verification of distributed embedded systems," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2014.

[13] G. Xie, G. Zeng, Y. Liu, J. Zhou, R. Li, and K. Li, "Fast functional safety verification for distributed automotive applications during early design phase," *IEEE Trans. Ind. Electron.*, vol. 65, no. 5, pp. 4378–4391, 2018.

[14] J. Henriksson, M. Borg, and C. Englund, "Automotive safety and machine learning: Initial results from a study on how to adapt the ISO 26262 safety standard," in *1st IEEE/ACM International Workshop on Software Engineering for AI in Autonomous Systems (SEFAIAS@ICSE)*, 2018.

[15] D. Goswami, R. Schneider, and S. Chakraborty, "Relaxing signal delay constraints in distributed embedded controllers," *IEEE Trans. Control. Syst. Technol.*, vol. 22, no. 6, pp. 2337–2345, 2014.

[16] M. Maggio, A. Hamann, E. Mayer-John, and D. Ziegenbein, "Control-system stability under consecutive deadline misses constraints," in *32nd Euromicro Conference on Real-Time Systems (ECRTS)*, ser. LIPIcs, vol. 165. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, pp. 21:1–21:24.

[17] D. Goswami, R. Schneider, and S. Chakraborty, "Re-engineering cyber-physical control applications for hybrid communication protocols," in *Design, Automation and Test in Europe (DATE)*, 2011.

[18] S. De, S. Mohamed, D. Goswami, and H. Corporaal, "Approximation-aware design of an image-based control system," *IEEE Access*, vol. 8, pp. 174 568–174 586, 2020.

[19] R. Schneider, D. Goswami, S. Zafar, M. Lukasiewycz, and S. Chakraborty, "Constraint-driven synthesis and tool-support for flexray-based automotive control systems," in *9th International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2011.

[20] D. Roy, L. Zhang, W. Chang, D. Goswami, and S. Chakraborty, "Multi-objective co-optimization of FlexRay-based distributed control systems," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2016.

[21] L. Bhatia, I. Tomic, A. Fu, M. Breza, and J. A. McCann, "Control communication co-design for wide area cyber-physical systems," *ACM Trans. Cyber Phys. Syst.*, vol. 5, no. 2, pp. 18:1–18:27, 2021.

[22] R. Mahfouzi, A. Aminifar, S. Samii, A. Rezine, P. Eles, and Z. Peng, "Breaking silos to guarantee control stability with communication over Ethernet TSN," *IEEE Des. Test*, vol. 38, no. 5, pp. 48–56, 2021.

[23] A. Masrur, S. Drössler, T. Pfeuffer, and S. Chakraborty, "VM-based real-time services for automotive control applications," in *16th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2010.

[24] J. Freitag, S. Uhrig, and T. Ungerer, "Virtual timing isolation for mixed-criticality systems," in *30th Euromicro Conference on Real-Time Systems (ECRTS)*, 2018.

[25] R. Schneider, D. Goswami, A. Masrur, M. Becker, and S. Chakraborty, "Multi-layered scheduling of mixed-criticality cyber-physical systems," *J. Syst. Archit.*, vol. 59, no. 10-D, pp. 1215–1230, 2013.

[26] M. Rumez, D. Grimm, R. Kriesten, and E. Sax, "An overview of automotive service-oriented architectures and implications for security countermeasures," *IEEE access*, vol. 8, pp. 221 852–221 870, 2020.

[27] AUTOSAR, "SOME/IP protocol specification," in *AUTOSAR Release 1.4.0*. AUTOSAR, 2016, p. 696.

[28] G. Pardo-Castellote, "OMG data-distribution service: Architectural overview," in *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.* IEEE, 2003, pp. 200–206.

[29] P. Joshi, P. Venugopal, and M. Osella, "Do we need data distribution service (DDS) and service-oriented architecture for automotive applications?" in *2019 Ethernet IP @ Automotive Technology Day.* IEEE, 2019, p. 17.

[30] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien, "Controller area network (CAN) schedulability analysis: Refuted, revisited and revised," *Real-Time Systems*, vol. 35, pp. 239–272, 2007.

[31] P. Joshi, S. Ravi, Q. Liu, U. D. Bordoloi, S. Samii, S. K. Shukla, and H. Zeng, "Approaches for assigning offsets to signals for improving frame packing in CAN-FD," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 5, pp. 1109–1122, 2019.

[32] E. Fraccaroli, P. Joshi, S. Xu, K. Shazzad, M. Jochim, and S. Chakraborty, "Timing predictability for SOME/IP-based service-oriented automotive in-vehicle networks." DATE, 2023.

[33] Z. Wang, R. Gupta, K. Han, H. Wang, A. Ganlath, N. Ammar, and P. Tiwari, "Mobility digital twin: Concept, architecture, case study, and future challenges," *IEEE Internet of Things Journal*, vol. 9, no. 18, pp. 17 452–17 467, 2022.

[34] S. Almeaibed, S. Al-Rubaye, A. Tsourdos, and N. P. Avdelidis, "Digital twin analysis to promote safety and security in autonomous vehicles," *IEEE Communications Standards Magazine*, vol. 5, no. 1, pp. 40–46, 2021.

[35] "IEEE standard for standard SystemC language reference manual," *IEEE Std 1666-2011 (Revision of IEEE Std 1666-2005)*, pp. 1–638, 2012.

[36] L. Rizzatti and J.-M. Brunet, "Introducing PAVE360: System- of-systems development for autonomous vehicles," *New Electronics*, vol. 53, no. 9, 2022.

[37] R. Pugh, "A supply chain in flux," *New Electronics*, vol. 53, no. 14, 2022.

[38] M. Andreozzi and G. Shirasat, "High-performance real-time systems design from cloud to embedded edge," *ARM Whitepaper*, 2022.

[39] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[40] R. Wilhelm, "Real time spent on real time," *Commun. ACM*, vol. 63, no. 10, pp. 54–60, 2020.