INTERNATIONAL MULTI DISCIPLINARY JOURNAL FOR RESEARCH & DEVELOPMENT
eISSN 2394-6334 Impact factor: 7,854 Volume 11, issue 10 (2024)

Architectural Frameworks for Deterministic Cyber-Physical Systems: Integrating Component-Based Software Engineering, Multi-Core Resource Management, and Time-Sensitive Networking

Trion Laufer
Department of Embedded Systems and Computer Science, University of Munich, Germany

Abstract: The rapid evolution of modern industrial and automotive systems has necessitated a paradigm shift from monolithic software designs to sophisticated, distributed component-based architectures. As these systems transition toward high-performance multi-core platforms and heterogeneous networking environments, ensuring temporal predictability and functional safety becomes increasingly complex. This research article provides a comprehensive investigation into the integration of component-based software models, such as the Rubus Component Model and COMDES-II, with advanced multi-core resource management techniques and deterministic communication standards. We explore the critical role of performance isolation in Multiprocessor Systems-on-Chip (MPSoC) through mechanisms like memory bandwidth reservation (MemGuard), cache partitioning (Coloris), and virtualization via real-time separation kernels. Furthermore, the article analyzes the shift from traditional Controller Area Networks (CAN) to Time-Sensitive Networking (TSN) and switched Ethernet, evaluating the timing analysis and modeling requirements for distributed vehicle functions. By synthesizing theoretical advancements in system-level performance analysis (SymTA/S) and fault-tolerant architectures, this study establishes a holistic framework for the design and optimization of next-generation cyber-physical systems. The findings emphasize the necessity of cross-layer predictability, from the software component level through the hypervisor and memory controller, to the network interface, ensuring that the rigorous demands of real-time control are met in the presence of task jitter and resource contention.

Keywords: Component-Based Software Engineering, Real-Time Systems, Multi-Core Resource Isolation, Time-Sensitive Networking, Rubus Component Model, Cyber-Physical Systems.

Introduction

The contemporary landscape of embedded systems is defined by an unprecedented increase in computational demand and structural complexity. Historically, embedded control systems utilized single-core microcontrollers and simple, cyclic executive schedulers to manage time-critical tasks. However, the advent of Industry 4.0 and the push toward autonomous driving have forced a transition toward Multiprocessor Systems-on-Chip (MPSoC) and distributed architectures. In these modern environments, software is no longer a localized script but a complex assembly of reusable components distributed across multiple Electronic Control Units (ECUs) and interconnected via high-speed networks. This shift introduces a profound challenge: how to maintain the deterministic behavior required for safety-critical control while leveraging the high-throughput capabilities of modern hardware.

The foundational response to this challenge has been the development of Component-Based Software Engineering (CBSE) specifically tailored for resource-constrained real-time systems. The Rubus Component Model (RCM), for instance, was designed to bridge the gap between high-level architectural modeling and low-level timing analysis, providing a structured approach to building predictable embedded software in the vehicle industry (Hänninen et al., 2008; Mubeen et al., 2017). By encapsulating functionality within components with well-defined interfaces and execution semantics, RCM allows developers to reason about system-level timing and resource consumption early in the design phase. Similarly, frameworks like COMDES-II offer generative development paths for distributed real-time control, emphasizing the need for a seamless transition from abstract models to executable code (Ke et al., 2007).

Despite the strengths of component-based models, the move to multi-core platforms introduces new sources of non-

determinism, primarily stemming from shared resource contention. When multiple processing cores attempt to access a single main memory (DRAM) or a shared cache simultaneously, the resulting interference can lead to significant execution time variability, often referred to as task jitter. Traditional real-time analysis techniques, which assume constant or bounded access times to memory, are frequently invalidated in these environments. Consequently, there is a critical need for performance isolation mechanisms that can enforce strict resource budgets at the hardware level. Systems like MemGuard and BWLOCK have emerged as vital tools for memory bandwidth reservation and dynamic access control, ensuring that high-criticality tasks are not starved by memory-intensive, low-priority applications (Yun et al., 2013; Yun et al., 2017).

Furthermore, the communication infrastructure must evolve to match the performance of the processing units. While the Controller Area Network (CAN) has been the workhorse of the automotive industry for decades, its limited bandwidth and non-deterministic queuing behavior at high loads make it unsuitable for data-heavy applications like sensor fusion and real-time video processing. The transition toward switched Ethernet and, more specifically, IEEE Time-Sensitive Networking (TSN) marks a significant milestone in network evolution. TSN provides the mechanisms for scheduled traffic, frame replication, and precise time synchronization, allowing for the deterministic delivery of critical data over a standard Ethernet physical layer (Pop et al., 2016). However, modeling and analyzing vehicle functions distributed over such switched networks require new methodologies that account for the unique characteristics of hardware switches and asynchronous traffic (Ashjaei et al., 2017).

In this article, we argue that true predictability in next-generation cyber-physical systems can only be achieved through a holistic, cross-layer approach. This involves integrating component-level timing specifications with separation kernels that provide temporal and spatial isolation, backed by hardware-level memory and cache management, and ultimately coordinated through a time-sensitive network. We explore the theoretical foundations of these technologies, evaluate their practical implementations in platforms like PikeOS and Xen, and discuss the implications of fault-tolerant architectures for safety-critical zonal controllers. By identifying the literature gaps in inter-core communication frameworks and dynamic cache partitioning, this study provides a roadmap for researchers and practitioners aiming to build the reliable, high-performance systems of the future.

Methodology

The methodology employed in this research follows a multi-faceted analytical approach, combining a review of established component models with a deep dive into resource isolation techniques and network timing analysis. The primary objective is to synthesize a unified framework that addresses the various levels of non-determinism present in distributed real-time systems.

First, we analyze the architectural principles of the Rubus Component Model (RCM). The methodology involves evaluating how RCM handles the separation of functional and non-functional requirements. Specifically, we examine the "Pipe-and-Filter" style used in RCM for data flow and the separate trigger mechanisms for control flow. This analysis extends to the RCM timing model, where we investigate how response-time analysis (RTA) is integrated into the development toolchain to provide early-stage feedback on task schedulability (Hänninen et al., 2008). We also compare RCM with other models like COMDES-II and specialized CORBA specifications to understand the trade-offs between flexibility and predictability in distributed environments (OMG, 2011; Ke et al., 2007).

The second stage of the methodology focuses on multi-core performance isolation. We conduct a detailed theoretical review of the "memory interference delay analysis" for COTS (Commercial Off-The-Shelf) multi-core systems. This involves modeling the DRAM controller behavior and the impact of bank-level parallelism on execution time (Yun et al., 2015). We specifically evaluate the MemGuard system, analyzing its "budget-replenishment" mechanism and its ability to maintain high system utilization while guaranteeing a minimum bandwidth to critical tasks (Yun et al., 2013). Parallel to memory management, we analyze cache partitioning methodologies, focusing on "page coloring" as implemented in the Coloris system. This allows for the dynamic reallocation of cache ways to specific tasks, mitigating the effects of inter-core cache thrashing (Ye et al., 2014).

Third, the methodology addresses the virtualization layer. We evaluate the performance measurements of hypervisors on embedded ARM processors, comparing Type-1 and Type-2 hypervisors in terms of interrupt latency and context-switch overhead (Toumassian et al., 2016). The analysis focuses on separation kernels like PikeOS and virtualized kernels like RT-Xen, which are designed to support mixed-criticality systems. We examine the formal API specifications of these kernels and their role in creating "partitioned" environments where tasks of different safety levels can coexist without interference (Verbeek et al., 2015; West et al., 2016).

Fourth, we investigate network-level timing analysis. The methodology involves modeling vehicle functions distributed over switched Ethernet, utilizing tools like the MPS-CAN analyzer to compare legacy CAN performance with modern Ethernet throughput (Mubeen et al., 2014; Ashjaei et al., 2017). We apply the principles of IEEE Time-Sensitive Networking (TSN), specifically analyzing the design optimization of cyber-physical systems through the use of TSN's scheduled traffic and per-stream filtering and policing (Pop et al., 2016). This includes exploring ontology-based "Plug-and-Play" approaches for TSN, which aim to simplify the configuration and integration of new sensors and actuators into an existing network (Farzaneh and Knoll, 2016).

Finally, the methodology integrates fault-tolerance considerations. We examine the dual-core lockstep architecture of the NXP S32G processor, analyzing its efficacy as a zonal controller in automotive systems. The methodology evaluates how lockstep execution provides a hardware-level safety net, detecting transient hardware faults through real-time comparison of core outputs (Abdul Salam Abdul Karim, 2023). This integrated methodology allows us to draw comprehensive conclusions about the interplay between software components, hardware resource management, and deterministic communication.

The evolution of component models for real-time systems is rooted in the need to manage complexity while ensuring that non-functional properties-such as timing, memory usage, and reliability-are treated as first-class citizens. The Rubus Component Model (RCM) represents a significant departure from general-purpose software engineering practices by explicitly distinguishing between the execution environment and the application logic. In RCM, a component is more than just a code unit; it is a structural entity that defines how it interacts with the system's temporal properties. This is achieved through the use of software circuits, where components are connected by "data wires" and "trigger wires."

The significance of this separation cannot be overstated. By decoupling the data flow from the control flow, RCM allows for a highly deterministic execution order. A component in RCM does not execute when data is available; rather, it executes when its trigger port receives a signal from the scheduler. This "triggered execution" model ensures that the scheduler has absolute control over the task sequence, which is essential for calculating the Worst-Case Execution Time (WCET) and the overall system response time. Mubeen et al. (2017) emphasize that this approach is particularly valuable in the vehicle industry, where predictability is a prerequisite for safety certification. The Rubus approach provides a seamless transition from the abstract component design to the physical implementation on a target ECU, ensuring that the timing constraints defined at the model level are enforceable in the final binary.

In contrast, other models like COMDES-II focus on the "generative" aspect of real-time development. COMDES-II utilizes an actor-based model where distributed real-time control systems are built using predefined architectural patterns. The framework emphasizes the use of "state-machine" components to manage complex control logic in a distributed fashion (Ke et al., 2007). While RCM excels in local predictability and ECU-level timing analysis, COMDES-II provides a robust framework for managing the interactions between distributed nodes, ensuring that the control-intensive nature of embedded systems is preserved across network boundaries.

Another vital consideration in the component-based paradigm is the role of middleware. The Common Object Request Broker Architecture (CORBA), particularly through its specialized real-time and embedded specifications, attempted to bring the benefits of object-oriented distribution to the real-time world (OMG, 2011). However, the overhead associated with traditional CORBA implementations often proved prohibitive for resource-constrained systems. This led to the development of Lightweight CORBA and other stripped-down versions that prioritize minimal memory footprints and deterministic request processing. The integration of these middleware standards with component models like RCM or Sentilles' model for distributed embedded systems (Sentilles et al., 2008) provides a standardized way for components to communicate across heterogeneous hardware, which is a key requirement for modern "connected" vehicles.

The theoretical gap that remains is the alignment of these component models with the underlying multi-core hardware. Historically, component models were developed with a single-core mindset. When an RCM task is moved to a multi-core MPSoC, the simple response-time analysis used in the past is no longer sufficient. The shared resources, such as the DRAM controller and the interconnect, introduce "interference delays" that are not accounted for in the component's local timing budget. Therefore, a modern theoretical framework must extend the component model's metadata to include hardware-specific resource demands, such as memory bandwidth and cache requirements, allowing the scheduler to perform resource-aware task placement.

Multi-Core Resource Isolation and Interference Analysis

As we move toward high-performance platforms like the Xilinx Zynq UltraScale+ or the Versal ACAP, the complexity of resource management increases exponentially. These platforms feature multiple ARM Cortex-A and Cortex-R cores,

along with programmable logic and high-speed memory interfaces (Xilinx, 2022; Xilinx, 2023). In such environments, the primary enemy of predictability is "inter-core interference." This occurs when a task on one core consumes an excessive amount of a shared resource-such as the memory bus-causing tasks on other cores to stall.

Yun et al. (2015) provide a foundational analysis of parallelism-aware memory interference. They argue that traditional models, which treat memory access as a sequential process, fail to account for the internal structure of modern DRAM. DRAM is divided into multiple banks that can be accessed in parallel. If two cores access different banks, they can theoretically proceed without interference. However, if they access the same bank or require the same shared command bus, the contention becomes significant. To address this, researchers have proposed DRAM controllers like Medusa, which are designed specifically for multicore-based embedded systems to provide high performance while maintaining predictability (Valsan and Yun, 2015).

To enforce isolation at the system level, the MemGuard framework introduces a memory bandwidth reservation system. MemGuard operates by monitoring the performance counters of each core to track the number of memory requests. Once a core reaches its allocated bandwidth budget for a given time period (or "period"), the framework throttles the core by using an interrupt-based mechanism to pause its execution until the next budget replenishment cycle (Yun et al., 2013). This ensures that a "rogue" or overly-intensive task cannot monopolize the memory interface, providing a guaranteed minimum bandwidth for safety-critical control tasks. Recent advancements, such as BWLOCK, have refined this approach by providing dynamic access control for soft real-time applications, allowing for more flexible resource utilization when the system is not under peak load (Yun et al., 2017).

Cache management is another critical pillar of performance isolation. In multi-core systems, the Last Level Cache (LLC) is typically shared among all cores. A task with a large memory footprint can "thrash" the cache, evicting the data of other tasks and causing their execution times to spike. The Coloris system addresses this through dynamic cache partitioning using "page coloring." Page coloring is a technique where the operating system exploits the mapping between virtual and physical addresses to ensure that data from specific tasks is mapped only to specific sets (or "colors") in the cache (Ye et al., 2014). By partitioning the cache into distinct regions, Coloris prevents tasks on different cores from evicting each other's data, effectively creating a "private" cache for each core within the shared physical hardware.

The challenge of inter-core communication further complicates the multi-core landscape. For distributed component models to function effectively on an MPSoC, there must be a way for components on different cores to exchange data without introducing non-deterministic delays. Tabish et al. (2021) propose an analyzable inter-core communication framework that leverages shared memory and structured synchronization primitives to ensure that data exchange does not become a bottleneck. By modeling the communication delay as part of the overall response-time analysis, this framework allows for the design of high-performance multicore systems where inter-core interactions are as predictable as intra-core task scheduling.

Results

The integration of multiple functions of varying criticality (e.g., a safety-critical braking system and a non-critical infotainment system) on a single hardware platform requires a robust virtualization layer. Virtualization allows for the creation of multiple isolated partitions, each running its own Operating System (OS) or runtime environment. In the context of real-time systems, this is typically handled by a "separation kernel" or a real-time hypervisor.

PikeOS is a prime example of a separation kernel used in the aerospace and automotive industries. It provides strict spatial and temporal isolation between partitions, ensuring that a failure in one partition cannot propagate to others. Verbeek et al. (2015) highlight the importance of formal API specifications for such kernels, as they allow for the mathematical verification of the isolation properties. By providing a verified foundation, separation kernels allow for the certification of mixed-criticality systems according to standards like ISO 26262 (automotive) or DO-178C (avionics).

In the open-source domain, the Xen hypervisor has been adapted for real-time use through projects like RT-Xen. Traditional hypervisors prioritize throughput and "fairness," which can lead to excessive latency for real-time tasks. RT-Xen addresses this by introducing real-time schedulers, such as the RTDS (Real-Time Deferrable Server) scheduler, which provides fixed-priority and earliest-deadline-first (EDF) scheduling for virtual machines (Xi et al., 2011; Wiki.Xenproject, 2019). This allows developers to assign specific execution budgets and periods to each VM, ensuring that time-critical partitions receive the processing time they need, exactly when they need it.

The performance overhead of virtualization, however, remains a concern. Toumassian et al. (2016) conducted extensive

measurements on embedded ARM processors to quantify the costs associated with hypervisor-managed context switches and interrupt handling. Their results show that while Type-1 hypervisors (which run directly on the hardware) generally offer lower overhead than Type-2 hypervisors (which run on top of a host OS), the jitter introduced by hypervisor-level scheduling must still be carefully managed. Simulation becomes an essential tool in this regard; Tran et al. (2021) argue that simulation is indispensable for embedded control systems where task jitter can lead to instability in the physical process. By simulating the hypervisor behavior and the task-level execution together, developers can identify potential timing violations before deploying to physical hardware.

The future of virtualization in the Internet of Things (IoT) and automotive sectors is moving toward "virtualized separation kernels" that combine the isolation of a traditional separation kernel with the flexibility of a hypervisor. West et al. (2016) describe such a system that supports mixed-criticality by using a combination of hardware-assisted virtualization and software-defined resource partitioning. This approach allows for the efficient use of MPSoC resources while maintaining the rigorous safety guarantees required for cyber-physical systems. WindRiver (2016) also emphasizes that virtualization is the key to managing the complexity of IoT-enabled embedded systems, allowing for the consolidation of legacy software and new, connected applications on a single platform.

Deterministic Communication: From CAN to TSN

While host-level resource management is crucial, the communication network is the lifeblood of any distributed cyber-physical system. For decades, the Controller Area Network (CAN) has provided a reliable, low-cost solution for automotive networking. However, its event-triggered nature and limited 1 Mbps bandwidth are increasingly inadequate. The MPS-CAN analyzer represents an integrated approach to response-time analysis for CAN, allowing engineers to verify that all messages will meet their deadlines under worst-case scenarios (Mubeen et al., 2014). Yet, as the number of nodes and the volume of data grow, CAN inevitably becomes a bottleneck.

The transition to switched Ethernet offers a massive increase in bandwidth (to 1 Gbps and beyond), but standard Ethernet lacks the determinism required for real-time control. In a standard switch, packets are buffered and forwarded based on "best-effort" logic, which can lead to unpredictable delays and packet loss due to congestion. Modeling vehicle functions over such networks requires a deep understanding of switch internals and the impact of cross-traffic on high-priority streams (Ashjaei et al., 2017).

IEEE Time-Sensitive Networking (TSN) addresses these limitations by introducing a suite of standards that bring determinism to Ethernet. The core of TSN is the IEEE 802.1Qbv scheduled traffic standard, which uses a "time-aware shaper" to partition network bandwidth into distinct time slots. This allows critical traffic to be transmitted in dedicated windows, free from interference from non-critical data. Pop et al. (2016) discuss the design optimization of cyber-physical systems using TSN, noting that the ability to schedule traffic at the microsecond level allows for the tight synchronization of distributed sensors and actuators.

Another critical component of TSN is the IEEE 802.1AS-2020 standard for timing and synchronization. This protocol ensures that all nodes in the network share a common global timebase with sub-microsecond accuracy. This global clock is essential for the coordinated execution of distributed tasks, as it allows for the precise alignment of sensor sampling and actuator commands across different ECUs. Farzaneh and Knoll (2016) propose an ontology-based "Plug-and-Play" approach for in-vehicle TSN, which leverages standardized metadata to automatically configure the network when a new device is connected. This significantly reduces the manual effort required to manage complex TSN configurations and ensures that the network remains deterministic even as the system configuration changes.

System-level performance analysis tools, such as SymTA/S (Symbolic Timing Analysis for Systems), are vital for verifying these complex distributed environments. SymTA/S uses a compositional approach, where the timing properties of individual tasks and network messages are combined to determine the end-to-end response times (Henia et al., 2005). By modeling the interactions between the software components on the ECUs and the messages on the TSN network, SymTA/S can identify potential bottlenecks and verify that the entire system meets its temporal requirements. This holistic analysis is the final step in the design process, ensuring that the integration of component-based software, multi-core hardware, and time-sensitive networking results in a truly predictable cyber-physical system.

Discussion

The convergence of the technologies discussed in this article-component-based software models, multi-core resource isolation, real-time virtualization, and time-sensitive networking-forms the backbone of the next generation of deterministic cyber-physical systems. However, the synthesis of these fields reveals several critical tensions that must

be addressed by future research.

One primary tension is between "static predictability" and "dynamic flexibility." Traditional real-time systems rely on static configurations, where every task's schedule and every message's transmission time are fixed at design time. While this ensures absolute predictability, it is highly inefficient and difficult to scale. Modern automotive systems, however, require the flexibility to handle dynamic software updates and varying operational modes. The move toward "soft real-time" frameworks and dynamic resource management (like BWLOCK) is an attempt to introduce flexibility without sacrificing safety. The challenge lies in creating "formally verifiable dynamic systems" where the reconfiguration logic itself is proven to be safe and predictable.

Another critical area is the "semantic gap" between software models and hardware execution. The Rubus Component Model provides an excellent way to specify timing constraints at the application level, but it has limited visibility into the DRAM controller or the cache state. Conversely, hardware-level isolation mechanisms like MemGuard are effective at throttling cores but are "task-agnostic"-they do not know which software component is being throttled or how it affects the overall system state. Future systems must bridge this gap by creating a bidirectional interface between the software component model and the hardware resource manager. The component model should provide "resource-usage hints" to the hardware, while the hardware should provide "interference telemetry" back to the software scheduler, allowing for more intelligent, context-aware resource allocation.

The role of fault-tolerance and safety-critical hardware also warrants deeper discussion. The dual-core lockstep (DCLS) architecture, such as that in the NXP S32G, provides a high degree of confidence in the correctness of computation (Abdul Salam Abdul Karim, 2023). However, DCLS is essentially a "spatial redundancy" technique that does not necessarily improve temporal predictability. In fact, the synchronization required to keep two cores in lockstep can introduce its own overhead. A truly robust architecture must combine spatial redundancy (for functional safety) with temporal redundancy and network-level reliability (like TSN's frame replication). The design of "zonal controllers" must therefore be seen as a multi-objective optimization problem, balancing safety, predictability, performance, and cost.

Furthermore, the "separation kernel" approach is evolving into a more complex "distributed orchestration" layer. In a vehicle with a central high-performance computer and multiple zonal controllers, the hypervisor on the central node must coordinate with the kernels on the zonal nodes via the TSN backbone. This creates a "macro-system" where the entire vehicle acts as a single, virtualized computer. The standardization of the interfaces between these distributed kernels is essential for interoperability and for the long-term maintainability of the software-defined vehicle.

Finally, the impact of task jitter on control performance remains a significant hurdle. Even with perfect network synchronization and core isolation, small variations in execution time-caused by interrupt handling or pipeline stalls-can accumulate and affect the stability of high-speed control loops. The use of "jitter-aware" control theory, combined with advanced simulation tools, is necessary to design control algorithms that are resilient to the temporal imperfections of the underlying platform. The work of Tran et al. (2021) on simulation for embedded control systems provides a crucial link between the computer science of real-time systems and the control engineering required to interact with the physical world.

Conclusion

This research has explored the complex interplay between software architecture, hardware resource management, and communication protocols in the design of next-generation cyber-physical systems. We have demonstrated that the transition to distributed, multi-core platforms necessitates a cross-layer approach to predictability. Component-based software models like the Rubus Component Model provide the necessary structure for application-level timing, while hardware-level isolation mechanisms like MemGuard and Coloris are essential for mitigating the non-deterministic effects of inter-core interference.

The integration of real-time hypervisors and separation kernels allows for the safe consolidation of mixed-criticality functions, provided that the performance overhead and jitter are carefully managed. Simultaneously, the evolution of automotive networking from CAN to Time-Sensitive Networking provides the deterministic communication backbone required for data-intensive, distributed applications. The synthesis of these technologies into a unified framework-supported by fault-tolerant hardware and rigorous system-level analysis-is the key to building the autonomous and industrial systems of the future.

Ultimately, the goal is to move toward a "predictability-by-design" paradigm, where the temporal behavior of a complex

cyber-physical system is as well-understood and verifiable as its functional logic. This requires continued innovation in inter-core communication, hardware-software co-design for resource isolation, and the standardization of distributed real-time orchestration. By bridging the gaps identified in this study, the research community can ensure that the next generation of embedded systems is not only high-performance and flexible but also fundamentally reliable and safe.

References

1. Abdul Salam Abdul Karim. (2023). Fault-Tolerant Dual-Core Lockstep Architecture for Automotive Zonal Controllers Using NXP S32G Processors. International Journal of Intelligent Systems and Applications in Engineering, 11(11s), 877–885. Retrieved from https://ijisae.org/index.php/IJISAE/article/view/7749

2. Ashjaei, M., Mubeen, S., Lundbäck, J., Gålnander, M., Lundbäck, K., Nolte, T. Modeling and timing analysis of vehicle functions distributed over switched Ethernet. In: 43rd Annual Conference of the IEEE Industrial Electronics Society, 2017.

3. Catalog of specialized CORBA specifications. OMG group, 2011.

4. Farzaneh, M., Knoll, A. An ontology-based Plug-and-Play approach for in-vehicle Time-Sensitive Networking (TSN). 7th IEEE Annual Information Technology, Electronics and Mobile Communication Conference, IEMCON, pp. 1-8, 2016.

5. Hänninen, K., et al. The rubus component model for resource constrained real-time systems. In: IEEE Symposium on Industrial Embedded Systems, 2008.

6. Henia, R., Hamann, A., Jersak, M., Racu, R., Richter, K., Ernst, R. System level performance analysis - the SymTA/S approach. Comput. Digit. Tech., 152 (2), pp. 148-166, 2005.

7. Ke, X., Sierszecki, K., Angelov, C. COMDES-II: A component-based framework for generative development of distributed real-time control systems. In: 13th International Conference on Embedded and Real-Time Computing Systems and Applications, 2007.

8. Mubeen, S., Lawson, H., Lundbäck, J., Gålnander, M., Lundbäck, K. L. Provisioning of predictable embedded software in the vehicle industry: The rubus approach. In: IEEE/ACM 4th International Workshop on Software Engineering Research and Industrial Practice, SER&IP, pp. 3–9, 2017.

9. Mubeen, S., Mäki-Turja, J., Sjödin, M. MPS-CAN analyzer: Integrated implementation of response-time analyses for Controller Area Network. J. Syst. Archit., 60 (10), pp. 828-841, 2014.

10. Pop, P., Raagaard, M. L., Craciunas, S. S., Steiner, W. Design optimisation of cyber-physical distributed systems using IEEE time-sensitive networks. IET Cyber-Phys. Syst.: Theory Appl., 1 (1), pp. 86-94, 2016.

11. Sentilles, S., Vulgarakis, A., Bures, T., Carlson, J., Crnkovic, I. A component model for control-intensive distributed embedded systems. In: International Conference on Component Based Software Engineering, CBSE, pp. 310–317, 2008.

12. Tabish, R., Wen, J., Pellizzoni, R., et al. An analyzable inter-core communication framework for high-performance multicore embedded systems. Journal of Systems Architecture, p 10217, 2021.

13. Toumassian, S., Werner, R., Sikora, A. Performance measurements for hypervisors on embedded arm processors. 2016.

14. Tran, L., Radcliffe, P. J., Wang, L. Simulation is essential for embedded control systems with task jitter. Des. Autom. Embedded Syst., 25, pp. 177-191, 2021.

15. Valsan, P. K., Yun, H. Medusa: A predictable and high-performance dram controller for multicore based embedded systems. In: 2015 IEEE 3rd International Conference on Cyber-Physical Systems, Networks, and Applications, pp 86–94, 2015.

16. Venkata, S. K., Ahn, I., Jeon, D., et al. Sd-vbs: The san diego vision benchmark suite. In: IISWC. IEEE

Computer Society, pp 55–64, 2009.

17. Verbeek, F., Havle, O., Schmaltz, J., et al. Formal api specification of the pikeos separation kernel. NASA Formal Methods Symposium, Springer, pp. 375-389, 2015.

18. West, R., Li, Y., Missimer, E., Danish, M. A virtualized separation kernel for mixed-criticality systems. ACM Trans. Comput. Syst. (TOCS), 34 (3), pp. 1-41, 2016.

19. Wiki.Xenproject. Xen wiki - rtds-based-scheduler. 2019.

20. WindRiver Systems Inc. Virtualization and the Internet of Things. WindRiver White Paper, p. 4, 2016.

21. Wolf, W., Jerraya, A. A., Martin, G. Multiprocessor system-on-chip (MPSoC) technology. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., 27 (10), pp. 1701-1713, 2008.

22. Xi, S., Wilson, J., Lu, C., Gill, C. Rt-xen: Towards real-time hypervisor scheduling in xen. International Conference on Embedded Software, IEEE, pp. 39-48, 2011.

23. Xilinx. ZCU 102 MPSoC TRM. 2022.

24. Xilinx. Xilinx Versal. 2023.

25. Xu, M. Rt-xen: Real-time virtualization based on xen. 2013.

26. Ye, Y., West, R., Cheng, Z., Li, Y. Coloris: a dynamic cache partitioning system using page coloring. International Conference on Parallel Architecture and Compilation Techniques, IEEE, pp. 381-392, 2014.

27. Yun, H., Ali, W., Gondi, S., et al. BWLOCK: a dynamic memory access control framework for soft real-time applications on multicore platforms. IEEE Trans Comput 66(7):1247–1252, 2017.

28. Yun, H., Pellizzoni, R., Valsan, P. K. Parallelism-aware memory interference delay analysis for cots multicore systems. In: 2015 27th Euromicro Conference on Real-Time Systems, pp 184–194, 2015.

29. Yun, H., Yao, G., Pellizzoni, R., Caccamo, M., Sha, L. Memguard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms. Real-Time and Embedded Technology and Applications Symposium, IEEE, pp. 55-64, 2013.

30. Yun, H., Yao, G., Pellizzoni, R., et al. Memory Bandwidth Management for Efficient Performance Isolation in Multi-Core Platforms. IEEE Transactions on Computers 65(2):562–576, 2016.