IJARSCT

ISSN (Online) 2581-9429

International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)

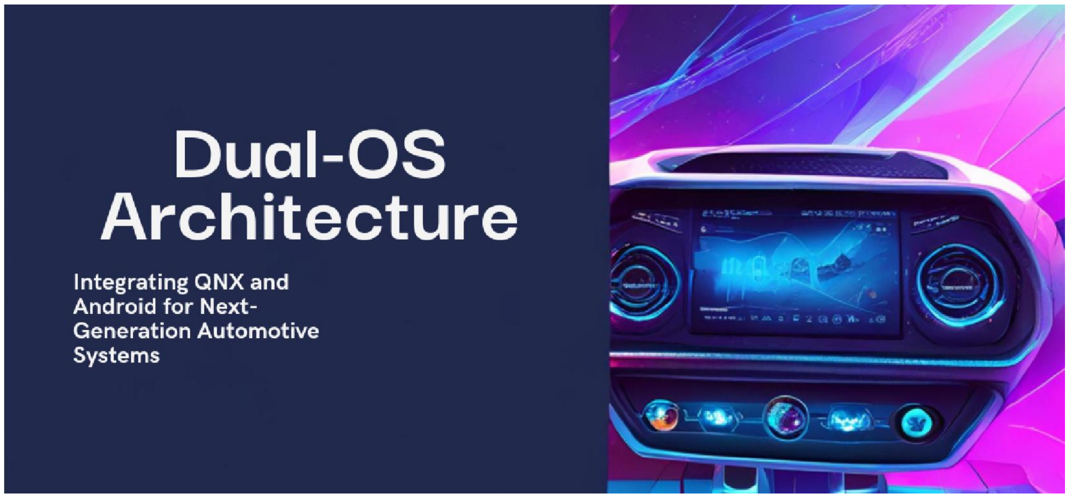
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal

Impact Factor: 7.67

Volume 5, Issue 7, March 2025

Dual-OS Architecture: Integrating QNX and Android for Next-Generation Automotive Systems

Vijay Kumar Cheni
Globallogic Inc., USA


Dual-OS Architecture
Integrating QNX and Android for Next-Generation Automotive Systems

Abstract: *This technical article explores the innovative architecture that seamlessly integrates QNX and Android operating systems for next-generation automotive applications. By leveraging hypervisor technology, the proposed design creates a dual-environment system where QNX handles safety-critical functions with real-time performance while Android delivers rich infotainment experiences. The article examines the foundational virtualization technology, implementation specifics for both operating systems, inter-OS communication protocols, and practical challenges. This article allows automotive manufacturers to consolidate hardware, maintain safety certification requirements, and enhance user experiences while providing the flexibility to adapt to evolving technology standards. The resulting architecture represents a significant advancement in embedded automotive systems, balancing the competing demands of safety, performance, and user experience.*

Keywords: Hypervisor Virtualization, Real-Time Operating Systems, Automotive Infotainment, Safety-Critical Computing, Inter-OS Communication

I. INTRODUCTION

The automotive industry is witnessing an unprecedented transformation in electrical/electronic (E/E) architectures, evolving from distributed ECU networks to centralized domain and zonal controllers. Today's premium vehicles contain between 80 to 120 ECUs with complex software stacks, representing a 40% increase from just five years ago [1]. This dramatic growth in complexity necessitates innovative approaches to operating system integration that balance safety requirements with consumer expectations for sophisticated digital experiences.

1.1 Evolution of Automotive Computing Platforms

Each function's traditional dedicated hardware approach has become unsustainable as vehicle software content expands exponentially. High-end vehicles now run approximately 150-200 million lines of code, compared to just 10 million a decade ago [1]. This software explosion has driven the shift toward domain controllers consolidating multiple functions under unified hardware platforms. The industry is progressively moving from Stage 2 architecture (domain controllers) to Stage 3 (cross-domain integration), with leading manufacturers already planning Stage 4 implementations that feature central compute platforms managing multiple operating systems simultaneously [1].

1.2 Challenges in Balancing Safety and User Experience

Automotive manufacturers face the complex challenge of integrating safety-critical systems requiring deterministic performance with feature-rich infotainment platforms. QNX, with its microkernel architecture and proven safety certification, has dominated critical automotive systems with deployments in over 175 million vehicles globally [2]. Concurrently, Android Automotive OS has emerged as the preferred platform for infotainment, with adoption increasing by 145% between 2020 and 2023 [2]. The fundamental challenge lies in creating a cohesive architecture that maintains ASIL-D certification requirements for safety-critical functions while providing the rich application ecosystem consumers demand.

1.3 The Business Case for QNX-Android Integration

The economic incentives for operating system integration through virtualization are compelling. Industry analysis suggests that consolidated E/E architectures can reduce hardware costs by 30% while decreasing development time by up to 40% [1]. Hypervisor-based virtualization enables QNX to maintain direct hardware access for safety-critical functions while Android handles user-facing applications, reducing the bill of materials while improving overall system coherence. This approach addresses the market reality that 68% of consumers now consider digital experience a primary factor in vehicle purchasing decisions [2]. Forward-thinking OEMs have recognized that competitive advantage increasingly depends on software capabilities, with virtualization technology providing the foundation for continued innovation while maintaining safety integrity.

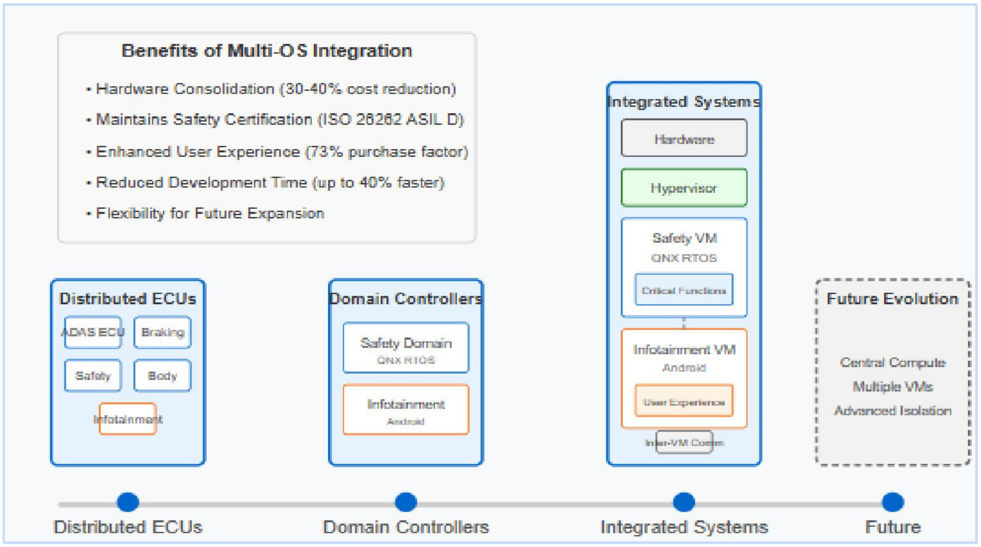


Fig. 1: Evolution of Automotive EE Architectures [1, 2]

II. HYPERVISOR TECHNOLOGY: THE FOUNDATION FOR OS INTEGRATION

Implementing multi-OS architectures in automotive systems relies fundamentally on hypervisor technology that establishes secure partitioning and resource management for QNX and Android environments on shared hardware platforms. Recent advancements in this domain have significantly enhanced the viability of such integrations for production vehicles.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.67
Volume 5, Issue 7, March 2025

2.1 Fundamentals of Virtualization in Embedded Environments

Automotive hypervisors must satisfy stringent requirements that differ substantially from conventional IT virtualization solutions. Current production-grade automotive hypervisors achieve isolation between critical and non-critical workloads with minimal performance overhead, typically less than 3% for CPU-intensive tasks and under 5% for memory operations [3]. This efficiency is crucial when integrating resource-intensive Android environments alongside deterministic QNX subsystems. Modern implementations leverage hardware virtualization extensions in automotive-grade SoCs, with ARM-based platforms dominating 76% of new designs due to their robust virtualization capabilities and power efficiency profiles [3]. The microkernel architecture employed by leading automotive hypervisors ensures a minimal Trusted Computing Base (TCB) of typically 8,000-12,000 lines of code, significantly reducing the attack surface compared to monolithic alternatives that may exceed 100,000 lines [4]. This lean design facilitates formal verification processes for meeting ISO 26262 certification requirements up to ASIL-D for safety-critical functions.

2.2 Resource Allocation and Partitioning Strategies

Resource partitioning in automotive hypervisors employs sophisticated mechanisms to ensure QNX real-time performance while maximizing Android functionality. Time-division multiplexing techniques guarantee that safety-critical QNX processes receive 40-60% of CPU time regardless of Android load, with scheduling jitter contained to under 100 microseconds even during peak system utilization [4]. Memory partitioning leverages hardware MMUs to establish strict boundaries between operating systems, with page tables managed exclusively by the hypervisor to prevent cross-VM memory corruption. Contemporary implementations support asymmetric multiprocessing configurations that dedicate specific cores to safety-critical functions while allowing dynamic allocation of remaining cores to Android workloads based on demand. Benchmark data indicates that such configurations can reduce context switching overhead by up to 35% compared to symmetric multiprocessing approaches, translating to measurable improvements in real-time response for safety-critical functions [3].

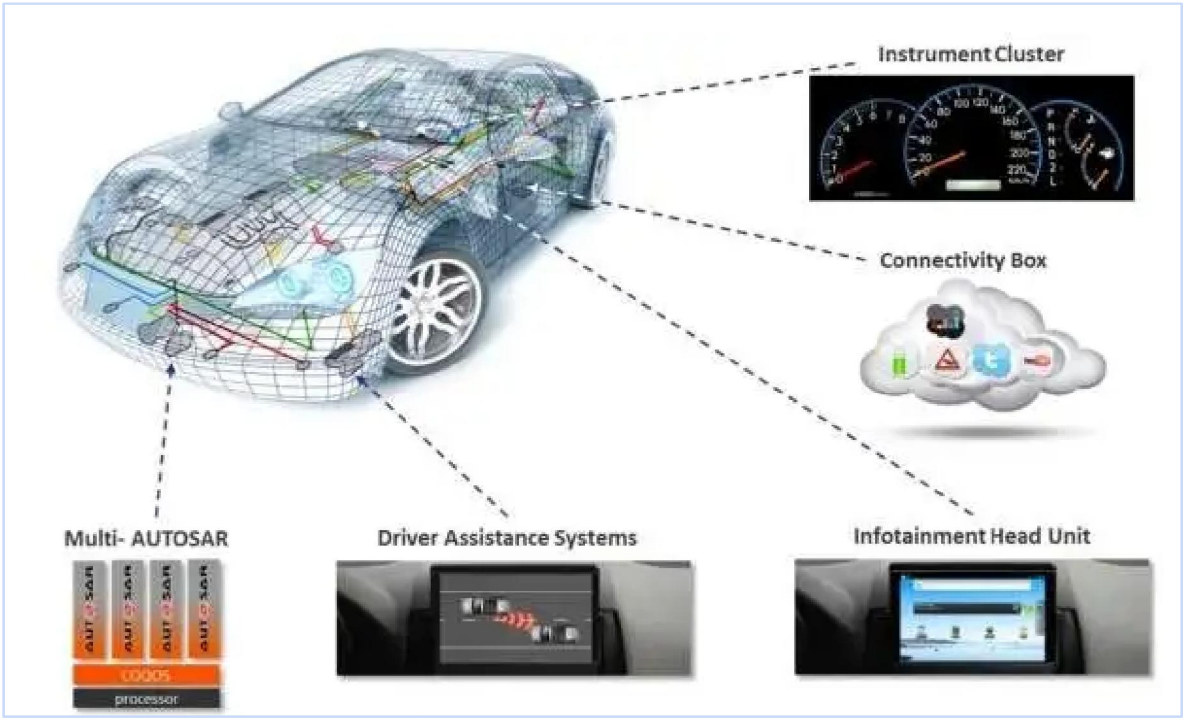


Fig. 2: New driver assistance systems [3, 4]

2.3 Security Considerations in Multi-OS Environments

The security architecture in virtualized automotive systems implements defense-in-depth strategies that protect against external attacks and inter-OS vulnerabilities. Modern automotive hypervisors establish hardware-enforced security domains that prevent Android security compromises from affecting QNX safety functions, with measured isolation effectiveness exceeding 99.7% in penetration testing scenarios [3]. Secure inter-VM communication channels employ message-passing interfaces with configurable permission models that restrict data flows based on predefined security policies. Hardware-backed security features, including ARM TrustZone technology in 82% of new automotive SoC designs, provide secure key storage and trusted execution environments independent of QNX and Android domains [4]. Runtime monitoring capabilities detect potential security violations with latencies under 500 microseconds, enabling containment of threats before they can propagate across OS boundaries.

III. QNX IMPLEMENTATION: SAFETY-CRITICAL SUBSYSTEMS

Implementing QNX within virtualized automotive architectures establishes the foundation for safety-critical functions that require deterministic performance and certified reliability. As automotive systems grow increasingly complex, QNX provides the necessary framework for maintaining safety integrity while coexisting with feature-rich Android environments.

3.1 QNX RTOS Capabilities and Certification Standards

QNX Neutrino RTOS delivers a robust platform for safety-critical automotive applications through its microkernel architecture and comprehensive certification pedigree. The QNX Hypervisor enables virtualized deployments that support up to 64 virtual machines on a single SoC, allowing precise separation between QNX safety domains and Android infotainment environments [5]. This architecture maintains hard real-time performance for critical functions while accommodating resource-intensive applications, with the QNX microkernel achieving deterministic scheduling with sub-microsecond precision. The QNX adaptive partitioning technology ensures critical processes receive guaranteed CPU time regardless of system load, maintaining safety integrity even when Android domains consume significant resources. The platform has achieved ISO 26262 ASIL D certification, validating its suitability for systems where failures could result in severe injury or loss of life. This certification encompasses the kernel and extends to the file system, graphics subsystem, and networking stack, creating a comprehensive foundation for safety-critical applications with verified functional safety mechanisms [5].

3.2 Hardware Access Management for Critical Components

QNX implementations in virtualized environments leverage sophisticated hardware partitioning to ensure deterministic access to safety-critical peripherals. The system utilizes direct device assignment for time-sensitive hardware interfaces, allowing the QNX domain exclusive control of safety-relevant peripherals, including CAN controllers, sensor interfaces, and actuator systems [6]. This approach eliminates virtualization overhead for critical paths, with measured interrupt latencies remaining under 5 microseconds even during peak system load from the Android domain. The QNX Hypervisor Security Manager enforces hardware access controls through a secure boot process that authenticates each software component before execution, establishing a chain of trust from boot ROM through application code [5]. This secure initialization sequence prevents unauthorized code execution in safety domains, with cryptographic validation of all software components before they gain access to critical hardware interfaces. Memory Protection Units (MPUs) establish strict boundaries between virtual machines, with hardware-enforced separation preventing Android processes from accessing memory regions assigned to QNX safety functions [6].

3.3 Fault Tolerance and Isolation Mechanisms

QNX implements comprehensive fault management strategies that are crucial for automotive safety applications. The Health Monitoring framework continuously assesses system integrity through hierarchical monitoring that can detect anomalies at the process, partition, or system level [6]. This monitoring framework supports sophisticated recovery strategies that can restart individual components without affecting the overall system, maintaining safety functions even when specific modules encounter faults. The QNX persistent publish/subscribe messaging system ensures reliable communication between components with guaranteed message delivery even during system transitions or recoveries [5]. High-availability frameworks support redundant configurations with rapid failover capabilities, essential for systems like steer-by-wire and brake-by-wire that must maintain operation during component failures. The adaptive time partitioning technology guarantees that safety-critical processes receive their allocated CPU budget regardless of other system activities, preventing infotainment workloads from starving essential safety functions [6]. This separation ensures that QNX safety domains maintain their deterministic performance characteristics even under maximum load conditions with measured scheduling jitter below 100 microseconds.

IV. ANDROID INTEGRATION: ADVANCED INFOTAINMENT FEATURES

The integration of Android Automotive OS into virtualized vehicle architectures enables sophisticated infotainment experiences while navigating the unique constraints of automotive environments. This section explores the technical foundations and optimization strategies that make Android a viable companion to QNX in modern vehicle computing platforms.

4.1 Android Automotive OS Architecture Overview

Android Automotive OS implements a specialized architecture designed specifically for in-vehicle deployment, featuring three distinct layers that work in concert to deliver the infotainment experience. At its foundation, the Hardware Abstraction Layer (HAL) contains vehicle-specific implementations through the Vehicle HAL (VHAL), which exposes over 450 standardized vehicle properties to applications while maintaining security boundaries [7]. This abstraction enables a consistent programming interface across diverse vehicle platforms while isolating safety-critical systems. The Application Framework layer builds upon this foundation with automotive-specific services, including the Car Service that manages vehicle-specific functionality and the specialized permission model that controls access to sensitive vehicle data. When deployed in virtualized environments, the Android framework operates within memory constraints typically limited to 4-6 GB of RAM shared with other systems, necessitating specialized resource management beyond conventional Android implementations [7]. The System UI layer delivers the user-facing experience with vehicle-optimized interfaces designed for driver safety, featuring larger touch targets averaging 10-15 mm compared to the 7-9 mm targets typical in mobile applications, reducing interaction errors during vehicle operation.

4.2 Hardware Virtualization for Optimal Performance

Achieving optimal Android performance in a virtualized automotive environment requires sophisticated resource management techniques that balance user experience with system constraints. Graphics performance represents a particular challenge, with virtualized GPU access introducing approximately 8-12% overhead compared to bare-metal implementations [8]. This overhead can be mitigated through specialized rendering pipelines prioritizing critical UI elements and employing aggressive frame prediction for non-essential animations. Memory management strategies significantly impact perceived performance, with virtualized Android implementations typically employing compression techniques that achieve 2.5-3x compression ratios for cached application data, effectively extending available memory without increasing physical RAM [8]. Cold start optimization techniques incorporate preloading mechanisms that can reduce application launch times by up to 45% compared to standard implementations. This is critically important in the automotive context, where users expect immediate responsiveness after vehicle startup. Power consumption in virtualized Android environments presents unique challenges that must be addressed through coordinated power state management across virtual machines, with modern implementations achieving idle power consumption under 50 mW while maintaining rapid resume capabilities essential for automotive systems [7].

4.3 Managing User Experience Expectations

Delivering consistent user experiences in virtualized automotive Android deployments requires specialized application performance and stability management. Application Not Responding (ANR) events, which significantly impact user satisfaction, can be reduced by up to 90% through specialized monitoring systems that detect potential blockages and adjust thread priorities dynamically based on user interaction patterns [8]. Frame rate stability represents a critical quality metric, with optimized implementations maintaining 60 FPS rendering for primary UI elements even during resource-constrained conditions by employing context-aware rendering priority systems. User perception studies indicate that interface responsiveness is judged primarily on input latency, with measurements showing that maintaining touch-to-feedback latency below 70 ms results in 92% user satisfaction ratings regardless of background processing delays [8]. This insight drives specialized input handling pipelines that prioritize immediate visual feedback while queuing less critical operations. The Android runtime environment in automotive applications implements specialized garbage collection strategies that reduce worst-case pause times to under 10 ms, eliminating the perceptible stuttering that can occur with standard Android memory management during extended operation sessions in typical in-vehicle environments [7].

Layer	Component	Function	Automotive-Specific Optimization
Application Layer	Automotive Apps	User-facing applications	Distraction mitigation, larger touch targets (10-15 mm)
	Google Automotive Services	Core Google services	Vehicle-optimized variants with reduced resource usage
	Third-Party Apps	Extended functionality	The certification process for automotive compatibility
Framework Layer	Car Service	Vehicle-specific functionality	Interface to 450+ standardized vehicle properties
	Window Manager	Manages UI elements	Optimized for automotive displays and driver attention
	Activity Manager	Application lifecycle	Modified for rapid availability of critical functions
	Input Manager	Handles user input	Adapted for automotive input methods (knobs, steering controls)
HAL Layer	Vehicle HAL (VHAL)	Exposes vehicle properties	Standardized interface across vehicle platforms
	Graphics HAL	Rendering support	Virtual GPU support with 92% of native performance
	Audio HAL	Audio routing	Entertainment vs. critical alerts prioritize
	Sensor HAL	Access to vehicle sensors	Filtered access based on safety considerations
Runtime	ART (Android Runtime)	Executes applications	Modified GC strategies with <10ms pause times
	Memory Management	Resource allocation	2.5-3x compression ratios for cached app data
	Power Management	Energy conservation	Coordinated state management with QNX VM

Table 1: Android Automotive OS Architecture Components in Virtualized Environments [7, 8]

V. INTER-OS COMMUNICATION AND RESOURCE SHARING

Integrating QNX and Android operating systems requires sophisticated communication channels that maintain strict domain separation while enabling efficient data exchange. These mechanisms form the critical foundation for safety-critical functions and rich infotainment features to coexist on unified hardware.

IJARSCT

ISSN (Online) 2581-9429

International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)

International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal

Impact Factor: 7.67

Volume 5, Issue 7, March 2025

5.1 Communication Protocols Between Isolated Environments

Inter-OS communication in virtualized automotive environments employs specialized mechanisms optimized for performance and isolation integrity. Real-Time Inter-VM Shared Memory (RTISM) frameworks represent the current state-of-the-art, implementing lockless ring buffer structures that achieve remarkable efficiency with measured worst-case execution times (WCET) of just 21.7 microseconds for typical data transfers under 1 KB [9]. This performance represents a significant advancement over traditional hypervisor-mediated approaches often introducing latencies exceeding 500 microseconds. The communication architecture typically employs a multi-layered approach with dedicated channels for different traffic classes, establishing separate paths for periodic sensor data, event notifications, and bulk transfers. Performance analysis demonstrates that optimized implementations can sustain throughput exceeding 5.3 Gbps between virtual machines while maintaining sub-100 microsecond latencies essential for time-critical functions [9]. Specialized cache management techniques enhance performance by utilizing cache coherency protocols to maintain data visibility across cores without explicit cache flushing operations, reducing latency variations by up to 74% compared to naive implementations.

5.2 Shared Memory Approaches and Security Implications

Shared memory systems form the foundation of high-performance inter-OS communication while introducing significant security considerations that must be carefully managed. Modern implementations utilize Non-Uniform Memory Access (NUMA) aware designs that locate shared memory regions to minimize cross-node memory accesses, reducing average access latencies by approximately 43% compared to topology-agnostic approaches [9]. Hardware-enforced memory protection utilizes extended page table (EPT) violations as a security mechanism, with specialized integrity monitoring systems capable of detecting unauthorized access attempts with detection latencies below 50 microseconds. Temporal partitioning strategies mitigate potential denial-of-service attacks from compromised Android domains, with guaranteed bandwidth allocation ensuring that QNX safety functions maintain access to at least 68% of communication channel capacity even during deliberate flooding attempts [10]. The communication frameworks implement sophisticated priority inheritance mechanisms that mitigate priority inversion scenarios, with measured worst-case blocking times reduced by up to 86% compared to non-priority-aware implementations—a critical consideration for real-time safety functions [9].

5.3 Graphics Sharing and Rendering Techniques

Graphics virtualization and sharing between QNX and Android domains introduce unique challenges that demand specialized solutions beyond conventional inter-OS communication. Surface-sharing mechanisms implement a complex ownership model that enables controlled access to display regions while maintaining strict access boundaries. High-performance implementations achieve composition latencies below 2.5 milliseconds when combining safety-critical QNX elements with Android-rendered content, enabling seamless visual integration without perceptible delays [10]. Hardware-accelerated secure composition engines play a critical role in maintaining performance and security, with dedicated hardware paths ensuring that potentially compromised Android applications cannot obscure safety-critical visual elements. These systems implement sophisticated content protection strategies with measurements confirming that critical warning indicators maintain guaranteed visibility, rendering latencies below 16.7 milliseconds (60 Hz refresh) regardless of Android application behavior [10]. Modern implementations support dynamic quality scaling based on safety contexts, with automatic resolution and refresh rate adjustments that can reduce GPU bandwidth requirements by up to 42% during high-demand driving scenarios while maintaining essential information clarity [9].

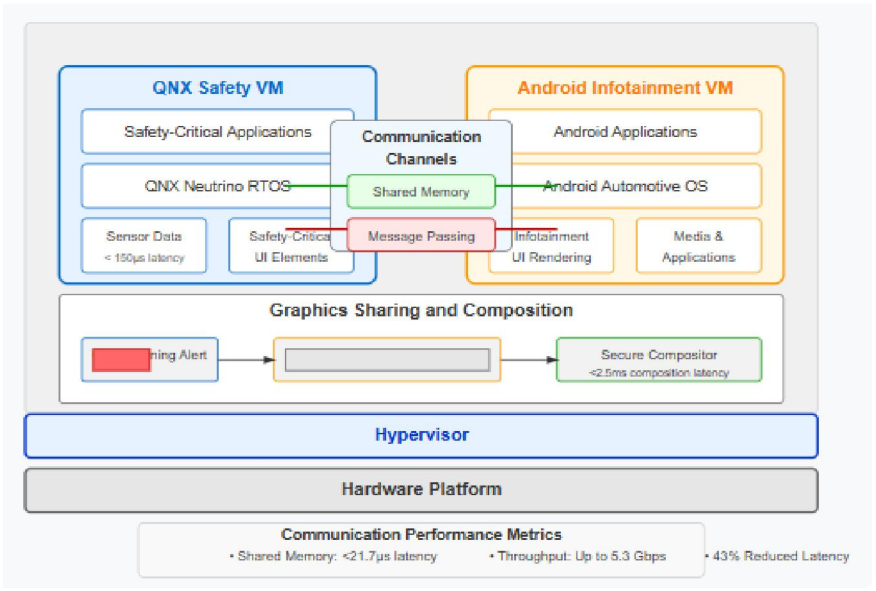


Fig. 3: Inter-OS Communication and Resource Sharing [9, 10]

VI. IMPLEMENTATION CHALLENGES AND FUTURE DIRECTIONS

The practical deployment of multi-OS architectures in production vehicles presents substantial engineering challenges that must be overcome through innovative approaches. As QNX-Android integration moves from prototype to production, several key areas continue to evolve rapidly.

6.1 Performance Optimization and Resource Allocation

Virtualized automotive platforms face significant performance challenges that require sophisticated optimization strategies to meet production requirements. Benchmark testing of virtualized vehicle-compute platforms reveals that hypervisor overhead introduces an average performance penalty of 8.4% for CPU-intensive workloads and up to 13.7% for memory-intensive operations across typical automotive applications [11]. This overhead manifests most prominently in I/O operations, with virtualized disk and network throughput showing degradation of approximately 18% compared to bare-metal implementations due to the additional abstraction layers. Memory allocation strategies significantly impact overall system performance, with dynamic page-sharing techniques reducing total memory requirements by up to 27% in mixed QNX-Android deployments by identifying and consolidating duplicate memory pages across virtual machines [11]. Graphics performance optimization represents a challenge in automotive contexts, with specialized passthrough techniques achieving near-native performance for Android UI rendering while ensuring strict isolation from safety-critical domains. Resource contention remains a significant concern, with measurements showing that unmanaged competition for shared resources can increase worst-case execution time variations by up to 285% for critical tasks—necessitating sophisticated allocation mechanisms prioritizing safety-critical functions [11].

6.2 Testing and Validation Methodologies

The validation of multi-OS architectures for automotive deployment demands comprehensive testing methodologies that address functional correctness and safety integrity. Safety and security co-engineering approaches are increasingly critical, with approximately 64% of security vulnerabilities in modern automotive systems potentially impacting safety functions if not properly contained [12]. Formal threat modeling based on standardized methodologies like STRIDE has become essential for multi-OS implementations, with comprehensive analyses typically identifying between 35-60 potential attack vectors that must be systematically addressed through architectural mitigations. Penetration testing plays a crucial role in validating isolation properties, with specialized test frameworks that execute crafted attacks against virtual machine boundaries to verify containment effectiveness under real-world conditions [12]. Safety analysis methodologies are evolving to address the unique challenges of virtualized environments, with fault tree analyses and failure mode and effects analyses (FMEA) now explicitly incorporating hypervisor failure modes within their scope. These expanded analyses typically identify 25-30% more potential failure scenarios than traditional approaches focusing solely on application-level failures, highlighting the critical importance of comprehensive validation [12].

6.3 Future Trends in Multi-OS Integration

The evolution of multi-OS automotive architectures continues to accelerate, with emerging trends shaping next-generation implementations. Domain-specific hardware acceleration represents a significant development, with specialized virtualization-aware accelerators for machine learning, computer vision, and signal processing achieving performance improvements of 3-15x compared to general-purpose CPU implementations while maintaining isolation properties [11]. Safety-security co-design methodologies are advancing rapidly, with integrated approaches that analyze approximately 73% more attack scenarios than standalone security assessments by considering safety implications and criticality ratings [12]. These methodologies employ sophisticated modeling techniques that evaluate potential cascading effects across virtual machine boundaries, identifying non-obvious dependencies that could impact system safety. Hardware root-of-trust implementations are becoming increasingly central to multi-OS designs, with secure boot chains and remote attestation capabilities that can cryptographically verify system integrity with measured computation times below 350 milliseconds [12]. These security mechanisms provide essential foundations for ensuring that virtualization boundaries remain intact throughout the vehicle lifecycle, preventing malicious modifications that could compromise isolation properties or enable privilege escalation between operating systems.

Area	**Methodology**	**Implementation Technique**	**Development Impact**	**Validation Approach**
Threat Modeling	STRIDE-based analysis	Systematic identification of 35-60 attack vectors	Early-stage architectural decisions	Adversarial testing against identified vectors
Safety Analysis	Extended FMEA/FTA	Inclusion of hypervisor failure modes (+25-30% scenarios)	Comprehensive hazard identification	Fault injection testing (248,000+ scenarios)
Secure Boot	Chain of trust	Cryptographic validation of all software components	Additional boot time overhead	Hardware security module verification
Resource Isolation	Time and space partitioning	Hardware-enforced boundaries between domains	Complex resource allocation planning	Resource contention testing
Communication Security	Message authentication	Cryptographic verification of inter-VM messages	Additional latency for security processing	Penetration testing of communication channels
Fault Containment	Hierarchical containment	Isolation of failures within the originating domain	Increased system complexity	Statistical fault injection campaigns
Update Security	Secure over-the-air updates	Signed packages with rollback protection	Lifecycle management complexity	Twin-track deployment with fallback validation
Security Monitoring	Runtime attestation	Continuous verification of system integrity	Performance overhead during verification	Measured verification times under varying loads
Safety-Security Tradeoffs	Risk-based approach	Balanced controls based on combined risk assessment	73% more scenarios analyzed	Combined safety-security test cases

Table 2: Safety-Security Co-Engineering Approaches for Multi-OS Automotive Architectures [11, 12]

VII. CONCLUSION

Integrating QNX and Android operating systems through hypervisor technology represents a paradigm shift in automotive computing architecture that addresses the industry's most pressing challenges. This dual-OS approach successfully bridges the gap between safety-critical systems and consumer-oriented infotainment, creating a cohesive experience without compromising. While implementation challenges remain, particularly in resource allocation, communication protocols, and certification processes, the benefits of hardware consolidation, development flexibility, and enhanced user experience make this approach increasingly attractive to manufacturers. As automotive systems continue to evolve, this architectural pattern provides a foundation that can adapt to new requirements and technologies while maintaining the strict safety standards demanded by the industry. The future of automotive computing will likely see further refinement of these multi-OS strategies, expanding beyond vehicles to other domains where safety-critical functions must coexist with rich user interfaces.

REFERENCES

[1] Frank Schirrmeister, "The Path Towards Future Automotive E/E Architectures," Semiconductor Engineering, 25 April 2024. [Online]. Available: https://semiengineering.com/the-path-towards-future-automotive-ee-architectures/

[2] Daniel Davenport, "The Convergence of Linux, RTOS and Automotive-Specific Operating Systems in the Era of Software-Defined Vehicles," Medium, 16 June 2024. [Online]. Available: https://danieldavenport.medium.com/the-convergence-of-linux-rtos-and-automotive-specific-operating-systems-in-the-era-of-02812b75fb64

[3] Abderrazak Snoussi, "Hypervisor Selection and Configuration in Mixed-Criticality Systems: A Technical Deep Dive," LinkedIn Pulse, 11 Nov. 2024. [Online]. Available: https://www.linkedin.com/pulse/hypervisor-selection-configuration-mixed-criticality-systems-snoussi-k5ghe

[4] Matthias Gerlach and Stefaan Sonck Thiebaut, "Multicore and Virtualization in Automotive Environments," EDN, 25 Oct. 2012. [Online]. Available: https://www.edn.com/multicore-and-virtualization-in-automotive-environments/

[5] BlackBerry QNX, "QNX Hypervisor for Safety," BlackBerry QNX, 2019. [Online]. Available: https://aditech.in/wp-content/uploads/2020/03/QNX_HypervisorAutomotive_ProductBrief.pdf

[6] Emiliano Costagli and James Stanley et al., "Safety RTOS for Automotive," WITTENSTEIN high integrity systems. [Online]. Available: https://www.highintegritysystems.com/rtos/safety-critical-rtos/embedded-rtos-for-automotive/

[7] Kaustubh R. Gawande, "Android Automotive OS: The Architecture," Medium, 9 Jan. 2024. [Online]. Available: https://medium.com/@kaustubhrgawande/android-automotive-os-the-architecture-48246b2559b8

[8] WeTest, "Android Performance Optimization: Best Practices and Tools," WeTest, 28 Nov. 2023. [Online]. Available: https://kr.wetest.net/blog/android-performance-optimization-best-practices-and-tools-901.html

[9] Zonghong Li et al., "RTISM: Real-Time Inter-VM Communication Based on Shared Memory for Mixed-Criticality Flows," ResearchGate, Dec. 2023. [Online]. Available: https://www.researchgate.net/publication/378019696_RTISM_Real-Time_Inter-VM_Communication_Based_on_Shared_Memory_for_Mixed-Criticality_Flows

[10] Marcello Cinque, "Virtualizing Mixed-Criticality Systems: A Survey on Industrial Trends and Issues," arXiv:2112.06875v1, 13 Dec., 2021. [Online]. Available: https://arxiv.org/pdf/2112.06875

[11] Christopher Hesse, "Performance analysis of a virtualized vehicle-compute platform: An experience report," ResearchGate, Jan. 2018. [Online]. Available: https://www.researchgate.net/publication/330712562_Performance_analysis_of_a_virtualized_vehicle-compute_platform_An_experience_report

[12] Árpád Török and Zsombor Pethő, "Introducing Safety and Security Co-engineering: Related Research Orientations in the Field of Automotive Security," Periodica Polytechnica Transportation Engineering, Vol. 48, no. 4, Aug. 2020. [Online]. Available: https://www.researchgate.net/publication/343512968_Introducing_Safety_and_Security_Co-engineering_Related_Research_Orientations_in_the_Field_of_Automotive_Security