Architectures for Mixed-Criticality Systems based on Networked Multi-Core Chips

R. Obermaisser, D. Weber
University of Siegen, Germany

***Abstract*—Mixed-criticality architectures with support for modular certification make the integration of application subsystems with different safety assurance levels both technically and economically feasible. Strict segregation of these subsystems is a key requirement to avoid fault propagation and unintended side-effects due to integration. Also, mixed-criticality architectures must deal with the heterogeneity of subsystems that differ not only in their criticality, but also in the underlying computational models and the timing requirements. Non safety-critical subsystems often demand adaptability and support for dynamic system structures, while certification standards impose static configurations for safety-critical subsystems. Several aspects such as time and space partitioning, heterogeneous computational models and adaptability were individually addressed at different integration levels including distributed systems, the chip-level and software execution environments. However, a holistic architecture for the seamless mixed-criticality integration encompassing distributed systems, multi-core chips, operating systems and hypervisors is an open research problem. This paper describes the state-of-the-art of mixed-criticality systems and discusses the ongoing research within the European project DREAMS on a hierarchical mixed-criticality platform with support for strict segregation of subsystems, heterogeneity and adaptability.**

I. Introduction

In many application areas such as avionics, industrial control, or healthcare there is an increasing trend for integrating functions with different certification assurance levels using a shared computing platform. For example, healthcare systems encompass monitoring functions for observing the patient's vital signs at home or at a point of care to clinicians. These services are vital for treatment and a critical aspect of the patient's safety needs, while they can be combined with less critical applications such as multimedia and entertainment.

Mixed-criticality is the concept of allowing applications at different levels of criticality to seamlessly interact and coexist on the same networked distributed computing platform. Mixed-criticality systems must meet multiple assurance requirements up to the highest criticality levels (e.g., DAL A in RTCA DO-178B [1], ASILD in ISO26262, SIL4 in EN ISO/IEC 61508 [2]). The foundations for this integration are mechanisms for temporal and spatial partitioning [3], which establish fault containment and the absence of unintended side-effects between components. Partitions encapsulate resources temporally (e.g., latency, jitter, duration of availability during a scheduled access) and spatially (e.g., prevent components from altering code or private data in other partitions).

Partitioning is a prerequisite for modular certification, where each application subsystem is certified to the respective level of criticality [4]. Also, generic arguments of a verified platform can be used bottom-up as part of the safety-case [5]. Here, the overall safety-case results from the composition of the safety arguments of the individual subsystems.

Mechanisms for temporal and spatial partitioning have been introduced at various levels including software execution environments, distributed systems and multi-core chips.

Hypervisors are layers of software that exploit the features of the hardware platform to establish independent software execution environments. Several virtualization solutions can be distinguished [6] such as full virtualization, operating-system virtualization support and bare metal hypervisors.

Temporal and spatial partitioning was also addressed in communication networks at chip level and in distributed systems. Time-triggered networks use Time-Division Multiplexing (TDM) to establish virtual communication links where components cannot interfere with the interactions of other components in the value and time domain. TTNoC [7] and AEthereal [8] are examples of time-triggered on-chip networks with support for temporal and spatial partitioning. Examples of time-triggered communication protocols at the level of distributed systems are TTEthernet, SAFEbus, TTP and FlexRay [9]. Event-triggered protocols establish a weaker degree of temporal partitioning, since they do not render the temporal behaviour of a component fully independent from other components. Event-triggered protocols establish bounded mutual effects of different application subsystems (e.g., AFDX [10]).

Multi-core chips represent a major challenge to their adoption in mixed-criticality systems, because of unintended interference among cores in the use of shared resources such as memories, caches, bus arbitration policies and input/output. These resources introduce a significant source of indeterminism in the execution time analysis and they make multi-core processors harder to analyse than single-core processors. In particular, there is no clear understanding how interference between cores affects timing analysis, which is of paramount importance for safety-critical systems.

This paper discusses the state-of-the-art and ongoing research on mixed-criticality architectures at different integration levels ranging from distributed systems to multi-core chips. The starting point is the discussion of fundamental requirements of mixed-criticality systems in Section II. Mixed-criticality integration using operating systems and hypervisors is the focus of Section III. Section IV gives an overview of multi-core architectures for mixed-criticality systems. Distributed-system architectures with communication

networks supporting temporal and spatial partitioning are addressed in Section V. Section VI highlights ongoing research within the European project DREAMS, which addresses open research gaps of today's architectures in order to establish a mixed-criticality platform for networked multi-core chips.

II. Requirements of a Platform for Mixed-Criticality Systems

This section discusses fundamental requirements of a platform for mixed-criticality systems in order to independently develop and integrate heterogeneous application subsystems of different safety-assurance levels and to enable modular certification.

A. Partitioning

Logically, a mixed-criticality systems consists of application subsystems with corresponding criticality levels (e.g., steer-by-wire, powertrain, comfort in a car), which can be further decomposed into components (e.g., engine control and transmission control in case of the powertrain subsystem). A platform for mixed-criticality systems provides resources for the execution of multiple application subsystems and their components. For each component, the platform must provide an execution environment with corresponding resources (e.g., processing, memory, input/output). Depending on the platform the execution environment can be a node computer in a distributed system, a processor core of a multi-core chip or a (software) partition established by a hypervisor.

Partitioning ensures the independence of these execution environments regardless of design faults affecting the components. Hence, a component within a partition is a Fault Containment Region (FCR) [11] that delimits the immediate impact of a fault. Other components can only be affected by faulty inputs and not via shared resources such as the same underlying processor core.

Spatial partitioning ensures that one FCR cannot alter the code or private data of another FCR. Spatial partitioning also prevents a FCR from interfering with control of external devices (e.g., actuators) of other FCRs.

Temporal partitioning ensures that a FCR cannot affect the ability of other FCRs to access shared resources, such as the common network or a shared processor. This includes the temporal behavior of the services provided by resources (e.g., latency, jitter, duration of availability during a scheduled access).

A primary goal of temporal and spatial partitioning is the support for modular certification [4]. In order to limit certification efforts, a mixed-criticality system should allow the independent certification of different subsystems, instead of considering the system as an indivisible whole in the certification process. Certification efforts can also be significantly reduced by separating the safety arguments of the platform from the applications that build on top of the platform. The platform can thereby establish a baseline safety argument for the certification of the overall system [12]. This modular certification approach is supported by certification standards (e.g., independent functions in DO-297, compliant items in IEC-61508 [2]).

The degree of physical integration determines the required fault-containment capabilities of the platform. If a distributed computer system is shared among subsystems of different criticality but every node computer serves only for a single criticality, then network-level fault containment is sufficient. In case application code of more than one criticality level is executed in a node computer, then also fault containment at chip level and operating-system level are necessary to ensure partitioning.

Different types of resources need to be considered in temporal and spatial partitioning such as processor cores, memory, communication resources and input/output resources. Furthermore, partitioning must address significant extra-functional attributes of the mixed-criticality system such as energy integrity, security and diagnosability.

B. Timing Requirements

Many safety-critical application subsystems are hard real-time control applications, where the achievement of control stability and safety depends on the completion of activities (like reading of sensor values, performing computations, communication activities, actuator control) in bounded time. In such hard real-time systems, missed deadlines represent system failures with the potential of consequences as serious as in the case of providing incorrect results. Thus, hard real-time systems must ensure a guaranteed response even in the case of peak load and fault scenarios. A timing and resource analysis must assess the worst-case behavior of the system in terms of communication delays, computational delays, jitter, end-to-end delays, and temporal interference between different activities.

In order to simplify this timing and resource analysis, mixed-criticality platforms should ensure determinism and temporal independence of safety-critical subsystems from non safety-critical ones. Temporal predictability and low jitter (i.e., difference between maximum and minimum computational and communication delays) also improves the quality of control in mixed-criticality systems. Control algorithms can often be designed to compensate a known delay, whereas jitter brings an additional uncertainty into a control loop that has an adverse effect [13]. In case of low jitter or a global time-base with a good precision, state estimation techniques allow to compensate known delays.

C. Heterogeneity

Mixed-criticality systems consist of heterogeneous application subsystems that differ not only in their criticality, but also exhibit dissimilar requirements in terms of timing (e.g., firm, soft, hard, non real-time) and different models of computation (e.g., dataflow, time-triggered messaging, distributed shared memory). Also, subsystems can have contradicting requirements for the underlying platform such as different trade-offs between predictability, certifiability and performance in processors cores, hypervisors, operating systems and networks.

For example, at chip-level today's platforms are either based on the shared memory paradigm or on message-based networks. The shared memory model simplifies data sharing and dynamic load distribution. However, shared memories typically also result in temporal unpredictability, since the access of components is not preplanned and concurrent access needs to be resolved dynamically. Message passing eliminates the communication overhead and hardware complexity of the coordination protocol needed for cache coherence [14]. Memory hierarchies and coherence protocols significantly contribute to temporal unpredictability [15]. Message passing is also superior to shared memories in case of a high computation/communication ratio, which is typical for safety-critical application subsystems [16].

D. Adaptability

Primarily due to safety concerns, reconfiguration of safety-critical subsystems is often reduced to selecting system-wide modes out of statically defined scheduling tables, which impairs the flexibility of resource management, but provides good analyzability and determinism. An important use case requiring adaptability in safety-critical subsystems is adaptive fault-tolerance based on spare resources or analytic redundancy [17].

Non safety-critical subsystem often demand higher flexibility and dynamic system structures, where components enter and leave at run-time, interact among each other in variable setups and realize different global application services.

A mixed-criticality system should support the reconfiguration upon foreseen and unforeseen changes in its operational and environmental conditions. Therefore, platforms require autonomous detection mechanisms for operational and environmental changes (e.g., monitoring deadlines, overload detection). Adaptability mechanisms have to safely reconfigure the system, without interrupting or interfering with the behavior of subsystems that are not subject to the reconfiguration activities. It is critical to complete the reconfiguration activities within bounded time and with predictable results, which is also known as the problem of assured reconfiguration [17]. It is also necessary to consistently switch between different configurations, where relevant internal state must be preserved. Intermediate configurations need to be avoided where parts of the systems are in the new configuration, while other parts remain in the old configuration.

E. Multiple Integration Levels

Many mixed-criticality systems encompass multiple integration levels ranging from multi-core processors managed by operating systems with time and space partitioning [18] to multi-cluster distributed systems.

There is a trend towards the widespread use of multi-core processors in safety-relevant embedded systems [19], although in safety-relevant industrial applications typically only one core is used when highly-critical tasks are involved [20]. For instance, this misuse is to prevent delays in concurrent access to caches by the different cores.

Mixed-criticality systems encompassing clusters of networked multi-core chips will be required to satisfy resource requirements exceeding the resources of a single node computer. In addition, today's technology does not support the manufacturing of electronic devices with failure rates low enough to meet the reliability requirements of ultra-dependable systems. Since failure rates of node computers are usually in the order of 10^{-5} to 10^{-6}, ultra-dependable applications require the system as a whole to be more reliable than any one of its node computers. This can only be achieved by utilizing fault-tolerance strategies that enable the continued operation of the system in the presence of node failures.

III. Mixed-Criticality Integration using Operating Systems and Hypervisors

The software execution environments for components of mixed-criticality systems are realized by operating systems and hypervisors with time and space partitioning. In the temporal domain, partitioning is achieved by applying scheduling mechanisms that assign available resources to tasks based on time slots. Spatial partitioning can be achieved by allocating a partition to a unique address space which is not accessible by other partitions.

Several software execution environments are available on the market reaching from bare metal (type 1) hypervisors to full featured operating systems with the capability of hosting heterogeneous guest operating systems in their partitions. A representative set of software execution environments capable of running mixed-criticality applications is provided in Table I.

PikeOS [21] is a bare metal hypervisor with certified compliance to safety standards such as DO-178B, EN50128 and IEC61508. It provides a partitioned environment for multiple operating systems with different criticality levels. The partitioning is achieved by a PikeOS separation micro-kernel located in kernel space complemented by the PikeOS system software in userspace constituting the basis for the separated partitions on top.

PikeOS combines static resource partitioning and virtualization. It operates with a static configuration of resource allocations which exclusively assigns the different resources (e.g. memory) to partitions. The architecture of PikeOS ensures that partitions can only access assigned resources. Spatial isolation of memory resources on the hardware layer is typically realized by Memory Management Units (MMUs) or Memory Protection Units (MPUs) for memory access control.

For a strict separation of temporal resources, exactly one time partition is active at a time. PikeOS uses time partition scheduling at top level combined with a priority-based FIFO inside a partition. The top level scheduler combines time and priority-driven scheduling to ensure that hard real-time requirements for critical applications are met while still providing best effort scheduling for non-critical tasks. The scheduling configuration itself can be changed on the fly to provide adaptability for dynamic changes of resources.

Several hardware architectures such as PowerPC, x86, ARM, MIPS and SparcV8/LEON are supported by PikeOS as

well as multi-core systems since version 3.1. Different guest operating systems (e.g., Linux, Android, RTEMS, OSEK) can be hosted inside the partitions as well as applications. On application level, PikeOS provides support for different APIs, such as POSIX, real-time Java and Ada. PikeOS is used in the Integrated Modular Avionics (IMA) of the Airbus A350.

VxWorks [22] is a networked real-time operating system that has been designed to be applied to distributed environments. It is conformant to the requirements of safety standards such as DO-178C/EUROCAE ED-12C Level A, ARINC 653 and IEC 61508. Spatial partitioning is achieved by using memory-protected containers for the partitions that are based on virtual memory contexts. These contexts are managed by VxWorks and use the processor's MMU to map them to the physical address space and to restrict access. Different options are provided for inter-task communication as for instance message queues, shared memory, sockets or pipes.

Temporal partitioning is achieved similarly to PikeOS by applying time preemptive scheduling mechanisms. In case of VxWorks, priority-based scheduling is combined with round-robin scheduling. This two-level scheduling architecture works with a low overhead of context switching between partitions. The schedule is static in contrast to PikeOS but a change of task priorities is possible at runtime.

VxWorks runs on different hardware architectures (e.g., Intel i960, Intel i386, MC680x0, MC683xx, R3000, SPARC, Fujitsu SPARClite) and offers different application related APIs as for instance POSIX, PSE52, ARINC API, Ada and C. VxWorks is applied in the Airbus A400M or the iDrive system of BMW for instance.

LynxOS [23] is another real time operating system that offers the integration of mixed-criticality applications at execution-environment level. LynxOS defines virtual machine brick-wall partitions making it impossible for applications in one partition to interfere with applications in another one.

Like PikeOS and VxWorks, it provides determinism to satisfy the hard real-time performance requirements and applies scheduling mechanisms to ensure the temporal isolation between partitions. The scheduling policy is preemptive and priority based meaning that the current process is preempted when a higher priority thread is ready to be executed.

Besides the round-robin scheduling policy which is applied by VxWorks, LynxOS also offers quantum and FIFO policies. This offers the possibility to use dynamic time-slicing for the execution of tasks. Spatial isolation and the protection of private data of the partitions is guaranteed by using a hardware MMU for managing memory access. Adaptability of the memory allocation of a partition at runtime is not supported since it is fixed at design time and the configured memory size may not be changed during system execution.

The operating system is available for x86, ARM and PowerPC platforms and supports different APIs such as POSIX, C++, Ada and Java. Application examples for LynxOS are avionics (e.g., F-35 joint strike fighter) and healthcare (e.g., radiotherapy).

The bare-metal hypervisor *XtratuM* [24] was designed for safety-critical real-time systems. It uses para-virtualization of the underlying hardware and encapsulates applications and guest operating systems in strongly separated partitions. Two types of partitions are supported: System partitions can manage as well as monitor the state of the overall system, while the other partitions serve for applications and hosted operating systems.

Spatial separation is achieved by allocating the virtual memory space of partitions to independent physical memory regions. Hence, a partition only has access to its own memory area and may not influence other partitions. XtratuM uses a MMU or a MPU to establish the memory separation at hardware level.

The basic architecture of XtratuM defines three layers: a hardware-dependent layer provides drivers for the underlying hardware, an internal-service layer is used only by the XtratuM core services, and a virtualization-service layer connects the para-virtualization services to the partitions.

Temporal partitioning is achieved by fixed cyclic scheduling for the different partitions. Inter-partition scheduling is handled in a hierarchic way by the application or the guest operating system that provides an internal implementation of its own scheduling algorithm.

Several hardware architectures are supported by XtratuM such as x86, LEON2 and LEON3 (SPARC v8), ARM Cortex-R4F. An XtratuM Abstraction Layer (XAL) offers support for bare-C applications. Also guest operating systems based on Linux (x86) or RTEMS may be installed in partitions. Furthermore, POSIX, ARINC-653 and Ada APIs are supported.

IV. Mixed-Criticality Integration in Multi-Core Processors

Many mixed-criticality systems involve high performance embedded computing applications [25], which require significant computational power from the underlying platform. Since the performance of single core processors has reached its boundaries [26], architectures with multi-core processors are becoming a design choice for embedded systems.

On the other hand, multi-core systems usually have a more complex hardware architecture. Typical sources of indeterminism are caches, buses and shared memories making the system less predictable and complicating the computation of worst-case execution times.

These challenges are addressed by deterministic chip-level architectures that allow to map mixed-criticality applications to multi-core processors. An overview of the state-of-the-art architectures from different research projects is given in Table II. The different architectural approaches are classified similarly to the software execution environments in Section III. Temporal and spatial partitioning at chip-level are considered in the first category. Of interest in regard to timing are the applied communication models (e.g., shared memory, NoC), the timing models (e.g., periodic, aperiodic, sporadic) and the temporal guarantees (e.g., bounded latency, jitter).

Heterogeneity is important with respect to the supported computational models (e.g., time-triggered, data flow, client-

Execution Environment	Partitioning	Timing/Scheduling	Supported Hardware	Supported Guest Operating Systems and APIs	Adaptability
PikeOS	Static configuration of resource allocation, memory partitioning through MMU/MPU	Combination of time and priority driven scheduling	PowerPC, x86, ARM, MIPS, SparcV8/LEON	POSIX, real-time Java, Ada, Linux, Android, RTEMS, OSEK	Scheduling schemes can be changed on the fly
VxWorks	Space partitioning through memory management services	Priority based two-level (preemptive and round-robin) scheduling	MC680x0, MC683xx, Intel i960, Intel i386, R3000, SPARC	POSIX PSE52 ARINC API Ada, C	Task priorities can be changed at runtime
LynxOS	Space partitioning through MMU which manages memory access	Priority based preemptive cyclic scheduling inside partitions	x86, ARM, PowerPC	POSIX, Linux 2.6, C++, Ada and Java	Configured memory size of a partition cannot be changed at runtime
XTRATUM	Partitions are spatially separated through MMU and do not share memory.	Fixed cyclic real-time scheduler for partition scheduling	x86, LEON, ARM Cortex-R4F	Partikle, LITHOS, RTEMS, Linux, POSIX, Ada	Switching between static scheduling schemes possible at runtime

TABLE I
EXECUTION ENVIRONMENTS

server) and supported implementation technologies (e.g., types of cores). Multi-core architectures enable adaptability via runtime reconfiguration of the cores and the Network-on-Chip (NoC).

GENESYS [27] defines a generic embedded system platform which was implemented at chip-level in ACROSS [28]. The hardware architecture provides inherent spatial and temporal segregation combined with determinism based on time-triggered schedules. ACROSS is designed for hard real-time applications, while also supporting the communication needs of non safety-critical subsystems. Time-triggered messages are transported with temporal guarantees as well as minimal delay, event-triggered messages are transported following the best effort strategy without specific temporal guarantees and data streams are intended for multimedia applications. A sparse global time base is available for all parts of the architecture and serves as the basis for all time-triggered services. Reconfiguration at runtime is possible using a trusted network authority that changes the time-triggered communication schedules.

COMPSOC [29] is another architectural approach for mixed-criticality integration on a multi-core chip. The main interest of COMPSOC is to establish composability and predictability. Hardware resources, such as processor tiles, NoC and memory tiles are virtualized to ensure temporal and spatial isolation. The result is a set of virtual platforms connected via virtual wires based on the Aethereal NoC [30]. A static TDM scheme with preemption is used for processor and NoC scheduling and shared resources have a bounded WCET to ensure determinism. CoMPSOC is independent from the model of computation used by the application but a reconfiguration of hardware resources at runtime is not supported.

PARMERASA [31] is the continuation of MERASA [32], [33] with a focus on multi-cores and the parallel execution of one application on different cores. The simultaneous execution of mixed-critical applications (hard and non real-time) is one of the primary topics.

Guaranteed resource partitions are provided at hardware level to ensure spatial isolation. Therefore, PARMERASA uses a clusterized architecture that allows the organisation of cores in virtual clusters connected by an NoC. A private memory resource per cluster is foreseen in this architecture. The NoC uses a TDMA hardware arbitration scheme. For strict separation, all I/O devices are implemented as memory mapped I/O with registers directly accessible by the NoC.

Shared memories are part of the architecture to allow the exchange and shared access to large input data for different cores. Furthermore, PARMERASA provides a hardware architecture that simplifies WCET analysis based on predictable cache coherence mechanisms to ensure determinism.

The architecture proposed by *IDAMC* [34], [35] is used in ARAMIS as well as RECOMP. It focusses on a many core model and is fully synthesizeable. Spatial isolation is achieved by address translation and the assignment of virtual channels for communication. Isolation in the temporal domain is ensured by arbitration mechanisms for all shared resources.

Partitioning is realized at tile level with a system controller reserving the resources for each application. A tile can have up to 16 cores and each tile is part of a node, which consists of four tiles and a central supervising system controller. The overall architecture supports up to 64 nodes. Tiles include one or more multi-processor cores, on-chip memory, DDR2 based memory with a dedicated controller as well as other peripherals interconnected via a bus-based subsystem.

The platform is based on the Globally-Asynchronous Locally-Synchronous (GALS) approach with an NoC composed of nodes that support different topologies (e.g., mesh, ring, star, tree). Each of the nodes contains a router that serves as the interconnect to the tiles of the node. Quality of Service (QoS) guarantees are provided for multiple traffic classes including bandwidth and latency properties for safety-critical applications.

A back suction mechanism [36] improves the latency for non-critical applications. It is able to exploit the predictable traffic patterns of the safety-critical applications. As a result,

the throughput of non-safety-critical applications is enhanced while guarantees for safety-critical parts are still provided.

Furthermore, the system controller acting as a trusted entity supervises all safety-related features. It is the only instance with access to configuration tables of the safety-critical functions. To adapt to changes of running applications, the system controller can change the configuration of shared resources during runtime.

V. Mixed-Criticality Integration in Distributed Systems

In the avionic domain, *Distributed Integrated Modular Avionics (DIMA)* introduces the concepts and guidelines for mixed-criticality integration based on distributed systems. DIMA is an evolutionary step beyond IMA by combining concepts of federated and integrated avionic systems.

IMA was based on centralized racks with standardized hardware for the sharing of cabine's computational and input/output resources for multiple avionic functions. In contrast, DIMA physically distributes integrated modules and connects them using a fault-tolerant real-time communication system [37]. Hence, the avionic electronic system is separated into integration areas such as cockpit, flight control and engines [38].

The communication in DIMA needs to support *temporal and spatial partitioning* between integrated modules. Therefore, suitable communication protocols like Avionics Full DupleX Switched Ethernet (AFDX) [10] are used. In addition, the communication system must support fault-tolerance such as active redundancy (e.g., self-checking pairs, triple modular redundancy) and replicated communication channels.

The *timing analysis* of AFDX-based systems has been extensively performed using different formalisms and system assumptions. For example, end-to-end communication with multi-cluster AFDX systems has been analyzed based on the Network Calculus (NC) [39]. A trajectory approach was proposed for reduced pessimism in worst-case bounds [40]. Deterministic and stochastic petri nets were used to model event messages and periodic communication in AFDX [41], [42].

An objective of DIMA is to avoid *heterogeneity* by introducing standardized distributed processing units. These include Core Processing Modules (CPMs) with processing capabilities, as well as Core Processing I/O Modules (CPIOMs) with processing and I/O resources. Heterogeneity is addressed by technology-transparency, which is based on hardware-independent interfaces [38].

Adaptability is supported by DIMA based on reconfiguration mechanisms. The purpose of reconfiguration in DIMA is to to reassign functions inside the physical architecture. For example, the use of computational and communication resources is optimized during different mission phases. Another goal is to preserve the availability of critical services upon the occurrence of hardware failures [38]. Therefore, resource management modes are introduced that refer to the resource allocation management based on resource demands [38].

A key aspect is the virtual backplane with its Virtual Links (VLs), which is configurable using software. Thereby, the virtual backplane can react to changes in operating modes and react to faults. The reconfiguration is typically performed in a preconfigured manner for certification reasons.

Different integration levels are partly addressed in DIMA. DIMA supports the link to the operating system and hypervisor level. The establishment of software partitions and the communication between partitions is based on message-based channels in sampling and queuing modes [43]. Also, different clusters with distinct communication networks can be coupled with remote data concentrators [44]. However, the integration of multi-core architectures in cluster-level architectures is insufficiently addressed. The previously highlighted aspects of partitioning, adaptability and real-time are addressed at the level of off-chip communication networks and at the operating system level. For example, the seamless virtualization of resources in networked multi-core chips is currently not solved in DIMA.

In the automotive domain, mixed-criticality integration using distributed systems is supported based on time-triggered communication networks. FlexRay is a time-triggered communication protocol with a static segment that supports segregation of the messages from different nodes in the time and value domains [45]. Local and central guardians protect the message transmission in static time slots based on a priori knowledge about the identify of the sender nodes and the predefined message timing with respect to a global time base. Temporal and spatial partitioning is also addressed in future Ethernet-based automotive communication networks. For example, a timing analysis of switched Ethernet topologies for mixed-criticality traffic is done in [46].

VI. DREAMS Project on Mixed-Criticality Architectures

This section presents ongoing research on mixed-criticality systems as part of the European research project DREAMS (Distributed REal-Time Architecture for Mixed Criticality Systems)[1], which aims at developing a cross-domain architecture and design tools addressing the research challenges from Section II. The architecture is cross-domain in nature and supports multiple domains including avionics, industrial applications and health-care systems.

An important goal is to close the gap between different integration levels of mixed-criticality systems (i.e., cluster, chip and software execution environment), while ensuring temporal and spatial partitioning for computational and communication resources. Resource managers achieve virtualization for heterogeneous platforms comprising networks with different communication protocols and topologies, as well as node computers with different processor cores, operating systems and input/output resources. In addition, integrated resource management will result in higher flexibility, adaptability and energy efficiency.

[1] http://dreams-project.eu

Architecture	Partitioning	Timing	Heterogeneity	Adaptability
GENESYS (ACROSS)	Strict temporal and spatial partitioning at NoC level based on TDMA, enforced by the network interfaces	Periodic, time triggered	Restricted to time-triggered model of computation, agnostic to the types of computational components	Trusted network authority can change the time-triggered communication schedules
COMPSOC	Temporal and spatial partitioning through virtualization of hardware resources (processor tiles, NoC and memory)	Static TDM scheme for processor and NoC scheduling	Architecture is independent from computational model used by application	Reconfiguration of hardware at runtime not supported
PARMERASA	Spatial isolation by guaranteed resource partitions on hardware level, NoC with TDMA arbitration for temporal isolation	hard real time support through deterministic hardware	Support for different Commercial-off-the-Shelf (COTS) multi-core processors	Provides configurable memory hierarchy
IDAMC	Spatial isolation by NoC address translation and virtual channel assignment resources, temporal isolation ensured by arbitration mechanisms for all shared resources	Hard real time support	Architecture fully synthesizeable in different FPGAs	Configuration of shared resources can be changed at runtime by the system controller

TABLE II
MIXED-CRITICALITY INTEGRATION IN MULTI-CORE ARCHITECTURES

A. Cross-Domain Waistline Architecture with Partitioning and Real-Time Support

In order to support cross-domain usability and an independent development of platform services, the platform services of the DREAMS architecture are structured in a waistline as shown in the left hand side of Figure 1. The core services are a stable waist encapsulating all those capabilities that are required in all targeted application domains for the realization of mixed-criticality systems. These core services also lay the foundation for exploiting the economies of scale as they can be implemented in a space and energy efficient way in hardware for a multitude of application domains. The core services offer capabilities that are required as the foundation for the construction of higher platform services and application services. Different underlying implementation options exist for each of the core services. For example, the core communication services can be realized using different protocols in NoCs or off-chip networks.

Four core services are mandatory and part of any instantiation of the DREAMS architecture, since they represent capabilities that are universally important for mixed-criticality systems and all considered application domains. The core services are absolutely necessary to build higher services and to maintain the desired properties (e.g., temporal and spatial partitioning, real-time support) of the architecture.

The *secure and fault-tolerant global time base* of DREAMS provides to each component a local clock, which is globally synchronized within the system of networked MPSoCs and within each MPSoC. The main rational for the provision of a global time is the ability for the temporal coordination of activities, the establishment of a deterministic communication infrastructure and the ability for establishing a relationship between timestamps from different components.

Another core service is the *timely and secure communication service with time and space partitioning*, which enables the interaction between components using heterogeneous networks for the realization of global mixed-criticality application services.

For the sharing of processor cores among mixed-criticality applications, including safety-critical ones, DREAMS offers *timely and secure execution services with time and space partitioning*.

The purpose of *integrated resource management* is system-wide adaptivity of mixed-criticality applications consuming several resources via global integrated resource management. The approach is based on the separation of system-wide decisions to meet global constraints from the local execution on individual resources.

B. Multiple Integration Levels

DREAMS supports *end-to-end channels over hierarchical, heterogeneous and mixed-criticality networks*. DREAMS introduces gateways for a system perspective of mixed-criticality applications combining the chip-level and cluster-level. On the one hand side, DREAMS supports uniform communication between heterogeneous and mixed-criticality networks. Gateway services enable this horizontal integration at the cluster-level across different off-chip communication networks with different protocols (e.g., TTEthernet, EtherCAT, etc.), different reliabilities (e.g., fault-tolerant networks with media redundancy and active star couplers, low-cost fieldbus networks). In addition, gateway services between NoCs and off-chip networks enable vertical integration through the seamless communication in hierarchical networks respecting mixed-criticality safety requirements.

The DREAMS architecture incorporates networking building blocks of time-triggered communication networks (e.g., network interfaces of NoCs with time-triggered communication plans, time-triggered communication controllers, and star

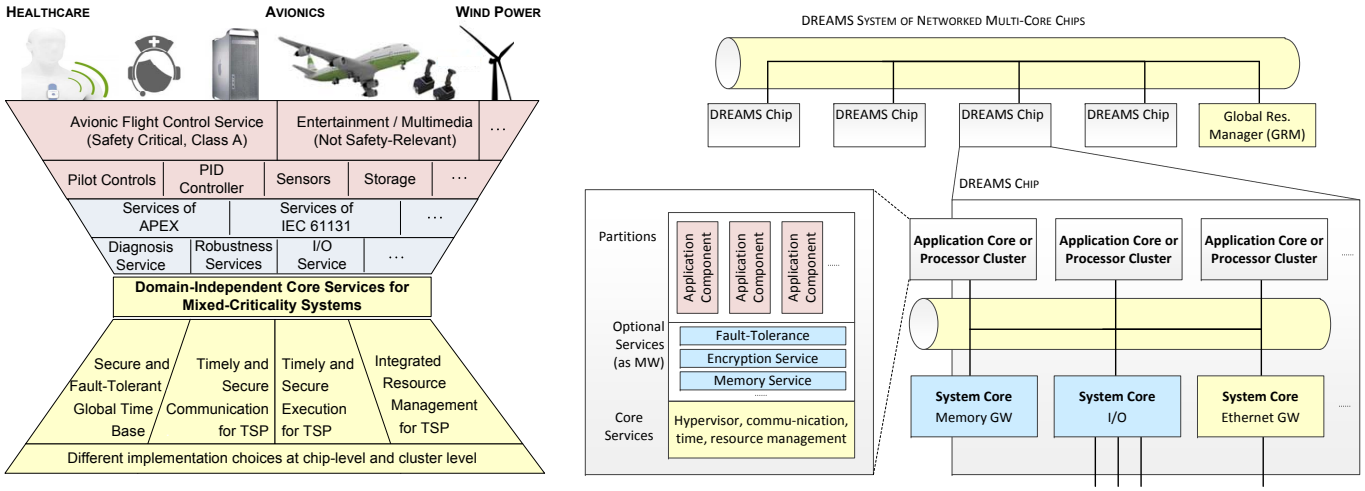

Fig. 1. DREAMS Waistline Architecture (left) and Corresponding Physical Platform of Networked Multi-Core Chips (right)

couplers) and event-triggered networks (e.g., traffic shaping). Based on an intelligent communication system with a priori knowledge about the allowed behaviour of components in the value and time domain, DREAMS ensures time and space partitioning.

For the *computational resources*, DREAMS integrates solutions of different types of hypervisors into a single system based on different processor clusters on a single chip and using different nodes at the cluster level. On one hand, DREAMS supports static scheduling, where an offline tool creates a schedule with precomputed scheduling decisions for each point in time. Furthermore, dynamic scheduling is supported using a quota system in the scheduling of tasks to limit the consequences of faults. Safety-critical partitions will establish execution environments that are amenable to certification and worst-case execution time analysis, whereas partitions for non safety-critical partitions will provide more intricate execution environments (e.g., based on Linux). In addition, the separation between safety-critical and non safety-critical applications will be supported using dedicated on-chip processor clusters with respective operating systems.

C. Heterogeneity

Another research objective of DREAMS is the support for heterogeneous application subsystems with different underlying models of computation (in addition to mixed-criticality). Examples of supported models of computation are dataflow, time-triggered messaging and event-triggered distributed shared memory.

DREAMS investigates the integration of computational activities with different timing models and heterogeneous interaction mechanisms at on-chip and off-chip networks. The following temporal types of activities are addressed:

- *Periodic activities:* Such an activity realizes a time-triggered, synchronous repetitive process with an activation period and phase set at the process creation. The period and phase can be specified with respect to a system-wide synchronized global time base. In case of computations, the task is only restarted after it is over at the next activation period. In case of communication, the instants of periodic message transmissions are specified by an a priori planned conflict-free communication schedule to ensure determinism and temporal properties such as latency, latency jitter, bandwidth, and message order.
- *Sporadic activities:* Such an activity is pseudo aperiodic with a minimum interarrival time in between successive activations. It realizes an event-triggered, asynchronous repetitive process. In case of communication, sporadic messages establish rate-constrained data-flows with a maximum bandwidth used to provide bounded latency.
- *Aperiodic activities:* This type of activity is aperiodic and realizes an asynchronous non-repetitive process triggered by some external events.

DREAMS combines *message-based communication* with the *shared memory model*. In time intervals not used for periodic communication, the shared memory model is established on top of message-based NoCs and message-based off-chip networks. Thereby, application subsystems are able to exploit programming models based on shared memory, while temporal and spatial partitioning of the message-based network infrastructure ensures segregation.

In addition, the waistline architecture supports heterogeneous underlying implementation technologies based on the technology-independent definition of the core platform services.

D. Adaptability

DREAMS provides services for system-wide adaptivity of mixed-criticality applications consuming several resources via global integrated resource management. A Global Resource Manager (GRM) performs global decisions with information from resource monitors. It provides new configurations for Local Resource Managers (LRMs) to virtualize resources

(e.g., partition scheduling tables, resource budgets). The GRM configuration can include different pre-computed configurations of resources (e.g., time-triggered schedules) or parameter ranges (e.g., resource budgets). Alternatively, the GRM can dynamically compute new configurations.

The LRMs adopt the configuration from the GRM at particular resources (e.g., processor core, memory, I/O). Resource monitors observe the resource availability (e.g., energy) and the timing of components (e.g., detection of deadline violations). The application behaviour is checked (e.g., stability of control) and intrusion detection is performed.

VII. Discussion and Conclusion

The state-of-the-art of mixed-criticality architectures provides support for partitioning and real-time guarantees, but architectures are limited to specific integration levels. Due to performance requirements of applications and trends of the semiconductor industry, the importance of multi-core chips is increasing in mixed-criticality systems. Hypervisors and operating systems are used for the virtualization of processors and the establishment of software execution environments. The integration of mixed-criticality nodes in distributed systems is enabled by communication networks with built-in partitioning mechanisms.

At present, a mixed-criticality architecture is missing that encompasses all of these integration levels. For example, consider a software component that is located in a partition established by a hypervisor on a multi-core platform. If this component requests access to a remote I/O resource on another chip, then end-to-end virtualization is required involving a software virtualization layer of the operating system, gateways between on-chip and off-chip networks (i.e., vertical integration) and possibly gateways between different types of off-chip networks (i.e., horizontal integration).

Heterogeneity and adaptability are partly supported. Primarily due to safety concerns, reconfiguration of safety-critical subsystems is often reduced to selecting system-wide modes out of statically defined scheduling tables, which impairs the flexibility resource management, but provides good analyzability and determinism. Several operating systems for mixed-criticality systems support adaptability by changing the scheduling configuration (e.g., PikeOS) or by changing task priorities (e.g., VxWorks), while memory and input/output allocations are typically fixed at design time. Likewise, basic reconfiguration abilities are supported by architectures for distributed systems and at chip-level.

An important research problem is the development of specific adaptability mechanisms for mixed-criticality applications combining static safety-critical applications and dynamic non safety-critical applications. In addition, safety-critical application subsystems could benefit from adaptability in the form of certifiable fault recovery strategies.

The research project DREAMS aims at providing contributions to these research challenges based on a mixed-criticality architecture for networked multi-core chips.

References

[1] *DO-178B: Software Considerations in Airborne Systems and Equipment Certification*, Radio Technical Commission for Aeronautics, Washington, DC, Dec. 1992.

[2] *IEC 61508 Edition 2.0: Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems*, IEC: Int. Electrotechnical Commission, 2010.

[3] J. Rushby, "Partitioning for avionics architectures: Requirements, mechanisms, and assurance," NASA Langley Research Center, NASA Contractor Report CR-1999-209347, Jun. 1999.

[4] J. Rushby, "Modular certification," Computer Science Laboratory SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025, USA, Tech. Rep., Sep. 2001.

[5] E. Althammer, E. Schoitsch, G. Sonneck, H. Eriksson, and J. Vinter, "Modular certification support – the DECOS concept of generic safety cases," in *Proc. of the 6th IEEE Int. Conference on Industrial Informatics*, 2008.

[6] SCOPE Alliance, "Promoting open carrier grade base platforms. virtualization: State of the art," April 2008, http://www.scope-alliance.org.

[7] R. Obermaisser, H. Kopetz, and C. Paukovits, "A cross-domain multiprocessor system-on-a-chip for embedded real-time systems," *IEEE Transactions on Industrial Informatics*, vol. 6, no. 4, pp. pp. 548–567, 2010.

[8] A. Hansson, K. G. M. Bekooij, and J. Huisken, "Compsoc: A template for composable and predictable multi-processor system on chips," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 14, no. 1, pp. 1–24, 2009.

[9] G. Bauer, K. Bilic, K. Driscoll, C. E. Salloum, P. Eles, W. Elmenreich, A. Goller, B. Hall, R. Kammerer, H. Kantz, H. Kopetz, R. Obermaisser, M. Paulitsch, P. Pop, T. Pop, C. Scherrer, E. Schmidt, and W. Steiner, *Time-Triggered Communication*, ser. Embedded Systems Series, R. Obermaisser, Ed. CRC Press, Taylor & Francis Group, 2011.

[10] Aeronautical Radio, Inc., "Arinc specification 664: Aircraft data network part 1 – systems concepts and overview," Radio Technical Commission for Aeronautics, Tech. Rep., Jan. 2002, 2551 Riva Road, Annapolis, Maryland 21401.

[11] J. Lala and R. Harper, "Architectural principles for safety-critical real-time applications," in *Proc. of the IEEE*, ser. 1, vol. 82, Jan. 1994, pp. 25–40.

[12] M. Nicholson, P. Conmy, I. Bate, and J. McDermid, "Generating and maintaining a safety argument for integrated modular systems," in *Proc. of 5th Australian Workshop on Safety Critical Systems and Software*, Nov. 2000, pp. 31–41.

[13] H. Kopetz, *Real-time systems – Design Principles for Distributed Embedded Applications*, 2nd ed. Springer, 2011.

[14] J. Leverich, H. Arakida, A. Solomatnikov, A. Firoozshahian, M. Horowitz, and C. Kozyrakis, "Comparing memory systems for chip multiprocessors," in *Proc. of the International Symposium on Computer Architecture (ISCA)*, 2007, pp. 358–368.

[15] Y. Li, V. Suhendra, Y. Liang, T. Mitra, and A. Roychoudhury, "Timing analysis of concurrent programs running on shared cache multi-cores," in *Proc. of the 30th IEEE Real-Time Systems Symposium*, Dec. 2009, pp. 57 –67.

[16] F. Poletti, A. Poggiali, D. Bertozzi, L. Benini, P. Marchal, M. Loghi, and M. Poncino, "Energy-efficient multiprocessor systems-on-chip for embedded computing: Exploring programming models and their architectural support," *IEEE Transactions on Computers*, vol. 56, no. 5, pp. 606 –621, may 2007.

[17] E. Strunk and J. Knight, "Dependability through assured reconfiguration in embedded system software," *Dependable and Secure Computing, IEEE Transactions on*, vol. 3, no. 3, pp. 172 –187, july-sept. 2006.

[18] B. Leiner, M. Schlager, R. Obermaisser, and B. Huber, "A comparison of partitioning operating systems for integrated systems," in *Proc. of Int. Conference on Computer Safety, Reliability and Security*, vol. 4680. Springer, Lecture Notes in Computer Science Volume, 2007, pp. 342–355.

[19] R. Ernst, "Certification of trusted mpsoc platforms," in *International Forum on Embedded MPSoC and Multicore*, 2010.

[20] J. Herman, C. Kenna, M. Mollison, J. Anderson, and D. Johnson, "Rtos support for multicore mixed-criticality systems," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2012 IEEE 18th*, April 2012, pp. 197–208.

[21] Theiling Henrik, "White Paper PikeOS and Time-Triggering," 2013.

[22] Parkinson, Paul and Kinnan, Larry, "Safety-Critical Software Development for Integrated Modular Avionics," 2007.

[23] Lynuxworks, "LynxOS User's Guide, Release 4.0,DOC-0453-02," http://www.lynuxworks.com/support/lynxos/docs/0453-02-los4_ug.pdf, 2005, last visited on 01/03/2014.

[24] A. Crespo, I. Ripoll, and M. Masmano, "Partitioned embedded architecture based on hypervisor: The xtratum approach." in *EDCC*. IEEE Computer Society, 2010, pp. 67–72.

[25] S. Mu, C. Wang, M. Liu, D. Li, M. Zhu, X. Chen, X. Xie, and Y. Deng, "Evaluating the potential of graphics processors for high performance embedded computing," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, March 2011, pp. 1–6.

[26] P. Gelsinger, "Microprocessors for the new millenium, challenges,opportunities, and new frontiers," in *Proc. of the Solid State Circuit Conference*. IEEE Press, 2001.

[27] R. Obermaisser and H. Kopetz, *GENESYS: An ARTEMIS Cross-Domain Reference Architecture for Embedded Systems*. Suedwestdeutscher Verlag fuer Hochschulschriften, 2009. [Online]. Available: http://books.google.de/books?id=ddZ2QgAACAAJ

[28] ACROSS Consortium, "D2.2 Functional Specification of Middleware and System Components," http://www.across-project.eu/download/ACROSS_D2.2.pdf, 2010, last visited on 01/04/2014.

[29] K. Goossens, A. Azevedo, K. Chandrasekar, M. D. Gomony, S. Goossens, M. Koedam, Y. Li, D. Mirzoyan, A. Molnos, A. B. Nejad, A. Nelson, and S. Sinha, "Virtual execution platforms for mixed-time-criticality systems: The compsoc architecture and design flow," *SIGBED Rev.*, vol. 10, no. 3, pp. 23–34, Oct. 2013. [Online]. Available: http://doi.acm.org/10.1145/2544350.2544353

[30] K. Goossens and A. Hansson, "The aethereal network on chip after ten years: Goals, evolution, lessons, and future," in *Proceedings of the 47th Design Automation Conference*, ser. DAC '10. New York, NY, USA: ACM, 2010, pp. 306–311. [Online]. Available: http://doi.acm.org/10.1145/1837274.1837353

[31] T. Ungerer, C. Bradatsch, M. Gerdes, F. Kluge, R. Jahr, J. Mische, J. Fernandes, P. G. Zaykov, Z. Petrov, B. Boddeker, S. Kehr, H. Regler, A. Hugl, C. Rochange, H. Ozaktas, H. Cassé, A. Bonenfant, P. Sainrat, I. Broster, N. Lay, D. George, E. Quiñones, M. Panic, J. Abella, F. J. Cazorla, S. Uhrig, M. Rohde, and A. Pyka, "parmerasa - multi-core execution of parallelised hard real-time applications supporting analysability." in *DSD*. IEEE, 2013, pp. 363–370.

[32] T. Ungerer, F. Cazorla, P. Sainrat, G. Bernat, Z. Petrov, C. Rochange, E. Quiñones, M. Gerdes, M. Paolieri, J. Wolf, H. Casse?, S. Uhrig, I. Guliashvili, M. Houston, F. Kluge, S. Metzlaff, and J. Mische, "Merasa: Multicore execution of hard real-time applications supporting analyzability," *Micro, IEEE*, vol. 30, no. 5, pp. 66–75, Sept 2010.

[33] J. Wolf, M. Gerdes, F. Kluge, S. Uhrig, J. Mische, S. Metzlaff, C. Rochange, H. Casse?, P. Sainrat, and T. Ungerer, "Rtos support for parallel execution of hard real-time applications on the merasa multicore processor," in *Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2010 13th IEEE International Symposium on*, May 2010, pp. 193–201.

[34] B. Motruk, J. Diemer, R. Buchty, R. Ernst, and M. Berekovic, "Idamc: A many-core platform with run-time monitoring for mixed-criticality." in *HASE*. IEEE Computer Society, 2012, pp. 24–31.

[35] S. Tobuschat, P. Axer, R. Ernst, and J. Diemer, "Idamc: A noc for mixed criticality systems," in *RTCSA*, 2013, pp. 149–156.

[36] J. Diemer and R. Ernst, "Back suction: Service guarantees for latency-sensitive on-chip networks," in *Networks-on-Chip (NOCS), 2010 Fourth ACM/IEEE International Symposium on*, May 2010, pp. 155–162.

[37] R. Wolfig and M. Jakovljevic, "Distributed ima and do-297: Architectural, communication and certification attributes," in *Digital Avionics Systems Conference, 2008. DASC 2008. IEEE/AIAA 27th*, Oct 2008, pp. 1.E.4–1–1.E.4–10.

[38] T. Wang and G. Qingfan, "Research on distributed integrated modular avionics system architecture design and implementation," in *Digital Avionics Systems Conference (DASC), 2013 IEEE/AIAA 32nd*, Oct 2013, pp. 1–53.

[39] M. Tawk, G. Zhu, Y. Savaria, X. Liu, J. Li, and F. Hu, "A tight end-to-end delay bound and scheduling optimization of an avionics afdx network," in *Digital Avionics Systems Conference (DASC), 2011 IEEE/AIAA 30th*, Oct 2011, pp. 7B3–1–7B3–10.

[40] H. Bauer, J. Scharbarg, and C. Fraboul, "Improving the worst-case delay analysis of an afdx network using an optimized trajectory approach," *Industrial Informatics, IEEE Transactions on*, vol. 6, no. 4, pp. 521–533, Nov 2010.

[41] Z. Jiandong, L. Dujuan, and W. Yong, "Modelling and performance analysis of afdx based on petri net," in *Future Computer and Communication (ICFCC), 2010 2nd International Conference on*, vol. 2, May 2010, pp. V2–566–V2–570.

[42] D.-j. Li, J.-D. Zhang, and B. Liu, "Periodic message-based modeling and performance analysis of afdx," in *Wireless Communications, Networking and Information Security (WCNIS), 2010 IEEE International Conference on*, June 2010, pp. 162–166.

[43] *ARINC Specification 653: Avionics Application Software Standard Interface, Part 1 - Required Services*, Aeronautical Radio, Inc., 2551 Riva Road, Annapolis, Maryland 21401, Mar. 2006.

[44] Aeronautical Radio, Inc., "ARINC 655 remote data concentrator (RDC) – generic description," Radio Technical Commission for Aeronautics, Tech. Rep., 1999, 2551 Riva Road, Annapolis, Maryland 21401.

[45] *ISO 17458-1 Road vehicles – FlexRay communications system*, British Standard / International Organization for Standardization, 2013.

[46] D. Goswami, M. Lukasiewycz, R. Schneider, and S. Chakraborty, "Time-triggered implementations of mixed-criticality automotive software," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, March 2012, pp. 1227–1232.