Distributed Real-time Architecture for Mixed Criticality Systems

Architectural Style of DREAMS
D 1.2.1

Project Acronym	DREAMS	**Grant Agreement Number**		FP7-ICT-2013.3.4-610640	
Document Version	1.0	**Date**	2014-07-31	**Deliverable No.**	1.2.1
Contact Person	Donatus Weber	**Organisation**		USIEGEN	
Phone	+49 271 740 3334	**E-Mail**		Donatus.weber@uni-siegen.de	

Contributors

Name	Partner
Roman Obermaisser	USIEGEN
Mohammed Abuteir	USIEGEN
Hamidreza Ahmadian	USIEGEN
Zaher Owda	USIEGEN
Donatus Weber	USIEGEN
Thomas Koller	USIEGEN
Mian Muhammad Hamayun	VOSYS
Leire Rubio	IKERLAN
Jon Pérez	IKERLAN
Miltos Grammatikakis	TEI
Michael Soulie	ST
Gerhard Fohler	TUKL
Simara Pérez	TUKL
Arjan Geven	TTT
Alfons Crespo	UPV
Simon Barner	FORTISS
Alexander Diewald	FORTISS
Gebhard Bouwer	TUV
Robert Heinen	TUV

Table of Contents

Abstract

This deliverable presents the architectural style of DREAMS, which was defined based on the requirements and constraints for a mixed-criticality architecture of networked multi-core chips.

The architectural style includes a system model, which describes the physical system structure of a platform that consists of networked multi-core chips, as well as a logical system structure of the application. Platform services for communication, global time, execution and resource management are part of a waistline architecture, which enables different underlying implementation options and a broad spectrum of refinements for different applications and industrial areas (e.g., avionics, healthcare, wind power). Building blocks are introduced for these platform services and mapped to the logical and physical system structure.

The communication services provide services for the message-based real-time communication among components. We establish end-to-end channels over hierarchical, heterogeneous and mixed-criticality networks respecting mixed-criticality safety and security requirements. The shared memory model is supported on top of message-based NoCs and message-based off-chip networks.

The time services offer a global time base, which is globally synchronized within the system of networked multi-core chips and within each multi-core chip. Therefore, internal and external clock synchronization is supported in each chip. This global time is foundation for the temporal coordination of activities and the establishment of a deterministic communication infrastructure.

The execution services enable the sharing of on-chip tiles with one or more processor cores among mixed-criticality applications. The introduced software architecture and the virtualization layer support both type 1 and type 2 hypervisors, while ensuring real-time guarantees, time/space partitioning, health monitoring and security. The execution services also provide APIs for the other platform services such as communication, resource management and time.

The resource management services provide services for system-wide adaptivity of mixed-criticality applications. The resource management services separate system-wide decisions from the local execution on individual resources. Resources are monitored individually with abstract information provided to the global resource manager. The global resource manager can take global decisions to be adopted by the local resource management. Thereby, system-wide constraints (e.g., end-to-end timing) can be addressed without incurring the complexity of individual negotiations among resources directly.

The certification strategy outlines a modular safety case with the relationship and scope for IEC 61508-2 / IEC 61508-3. Modularization of the DREAMS architecture including the hypervisor, the networks and COTS components is considered for the hardware and software safety argumentation.

Introduction

This deliverable describes the architectural style of a mixed-criticality system built according to DREAMS. The architectural style provides structuring rules according to several integration levels. At the chip-level a multi-core chip is decomposed into tiles interconnected by a Network-on-a-Chip (NoC) where each tile contains partitions established by a hypervisor. At the cluster-level the system is decomposed into nodes interconnected by off-chip networks. At the different levels, the architecture provides platform services, which separate the application functionality from the underlying platform technology in order to reduce design complexity and to facilitate the achievement of temporal and spatial partitioning, real-time support, reliability, security and energy-efficiency. The architectural style supports the integration of applications with different timing models and different safety assurance levels.

This deliverable introduces architectural building blocks and services including on-chip and off-chip communication services, global time services, execution services as part of a mixed-criticality software architecture as well as local and global resource management services. Certification is discussed using a safety concept based on these building blocks.

Structure of the Deliverable

Part I introduces the system model for mixed-criticality systems with a logical and physical system structure. A waistline structure of services is established with communication, time, execution and resource management services that are mapped to this system structure. Finally, fault assumptions are presented with the assumption w.r.t. fault containment units, failure modes and threats.

Part II of the deliverable describes these services. The communication services encompass on-chip and off-chip networks, network interfaces, IO services and gateways. The time services include on-chip and off-chip synchronization services. The focus of the execution services is a software architecture with a DREAMS virtualization layer. The resource management services consists of services for global resource management, local resource management and resource monitoring at the different levels of the hierarchical system comprised of networked multi-core chips. The optional services are discussed based on an example, namely a voting service for active redundancy.

Part III is an annex that presents the Application Programming Interface (API) that is provided to the DREAMS applications. Each interface is in detail described by providing information such as Synopsis, category, declaration, description, return value and usage examples.

Process for Preparation of the Deliverable

Working groups lead by experts of the respective area were established to work on the different parts of the architectural style (see Table 1). The establishment of the architectural style involved numerous meetings and telephone conferences for discussions within the working groups as well as WP1 meetings and telephone conferences for integration and alignment between the working groups for the overall architectural style. The document was prepared in several iterations with internal reviews within DREAMS. In particular, the feedback from the industrial partners of the application domains lead to improvements w.r.t. the suitability for the considered industrial areas.

Working Group	***Lead Partner***	***Responsible***
System model	USIEGEN	Roman Obermaisser

Communication services	USIEGEN	Roman Obermaisser
Global time	TTT	Arjan Geven
Resource management	TUKL	Gerhard Fohler
Optional services	FORTISS	Simon Barner
Certification strategy	IKL	Leire Rubio

Table 1: Working Groups

Architectures from previous projects served as the starting point and input. For example, the GENESYS architectural style and corresponding time-triggered architectures from ACROSS and INDEXYS provided a starting point for the communication and time services. Hypervisors and certification concepts from MULTIPARTES were analyzed to define the DREAMS execution services. The results of the input projects FRESCOR and ACTORS were considered for the definition of the resource management services. Although these inputs served as a starting point, substantial contributions and extensions beyond these prior results were provided in order to establish the DREAMS architectural style for networked multi-core chips.

Relationship to other DREAMS Deliverables

The requirements document D1.1.1 served as the primary input for the architectural style. The system model, the architectural services and the safety concept were defined to comply with the respective requirements. The architectural style is the foundation for the subsequent, parallel work within DREAMS. Based on the defined services, WP2 will develop on-chip communication, time, execution and resource management services. The development of these services at the cluster level is the focus of WP3. D1.2.1 is also an important input for WP4, where the adaptation strategies are based on the overall system model and the configurability of the architectural services as defined in D1.2.1. WP5 requires D1.2.1 as an input for the definition of the modular safety-case as well as for developing the simulation and fault injection framework. Furthermore, D1.2.1 is essential for the development of the demonstrators in WP6-8.

Consideration of Requirements

The definition of the architectural style was driven by the requirements from D1.1.1. In the following, an overview of the relationship to the requirements is given. The detailed analysis of the satisfaction of the requirements by the DREAMS architecture is the goal of the assessment (cf. D1.8.1, D1.8.2).

The *overall system model* provides the logical and physical system models for networked multi-core chips with corresponding architectural services. This hierarchical system model is essential for the evolvability, scalability and complexity management of mixed-criticality systems. The waistline architecture with domain and technology independent core services addresses the corresponding requirements from D1.1.1. The core services allow to abstract from the platform technology such as network protocols and types of processor cores as long as the architectural properties and services of DREAMS can be established. The introduced fault hypothesis is the foundation for satisfying the requirements on fault detection, containment and masking.

The *communication services* support the timing models (i.e., periodic, sporadic and aperiodic) demanded in D1.1.1 along with real-time support, security and temporal/spatial partitioning based on the enforcement of time-triggered schedules, rate-constraints and permitted address information. The gateway services address the requirements for a system perspective of mixed-criticality applications by combining the chip-level and cluster-level networks and by performing protocol transformations between heterogeneous networks. Heterogeneity is also considered by mapping heterogeneous networks to technology-independent network interfaces. The communication services further include the required reconfiguration support as the basis for resource management.

The *global time services* fulfill the demand for a consistent global time base in a system of networked multi-core chips with bounded precision and bounded accuracy. In addition, the global time services are the foundation for satisfying the requirement of synchronized activities. Therefore, the deliverable also explains the use of the global time base for communication and execution services.

The *execution services* introduce a software architecture with a virtualization layer to support the requirements for real-time, security, temporal/spatial partitioning, fault isolation and management. The requirement for technology independence is addressed by considering different types of hypervisors such as bare-metal hypervisors (e.g., Xtratum) and type 2 hypervisors that are hosted by an operating system (e.g., Linux KVM).

The *resource management* services support the requirements for resource management in networked multi-core chips based on local resource monitoring, local resource scheduling, local resource management and global resource management. We introduce a hierarchy with a global resource manager at the top, which directly supervises and controls a set of local resource managers and has a complete view of the system. A primary focus of D1.2.1 is the characterization of these resource management building blocks, the description of the interactions between global and local activities as well as the resource management for different resource types (e.g., communication resources, computational resources, I/O).

The *optional services* allow the refinement of the architecture towards different applications and industrial domains. They are essential for fulfilling the requirement of domain-independence. In addition, we introduce fault-tolerance services to mask component failures according to the DREAMS fault hypothesis.

The *certification s*trategy provides a safety concept that is driven by the requirements for certification from D1.1.1. It addresses a modular safety case, cross-domain dependability patterns as reference designs and product line certification.

Part I
System Model of Mixed-Criticality Architecture

1 System Model of a Mixed-Criticality System

Foundation for the architectural style is a clear definition of the system model of a mixed criticality system. Therefore the following sections provide necessary definitions and explanations on system structure, waistline of services as well as architectural building blocks.

1.1 System Structure

This section describes the physical system structure of a platform that consists of networked multi-core chips. In addition, a logical system structure of the application and a corresponding namespace is defined (see Figure 1)

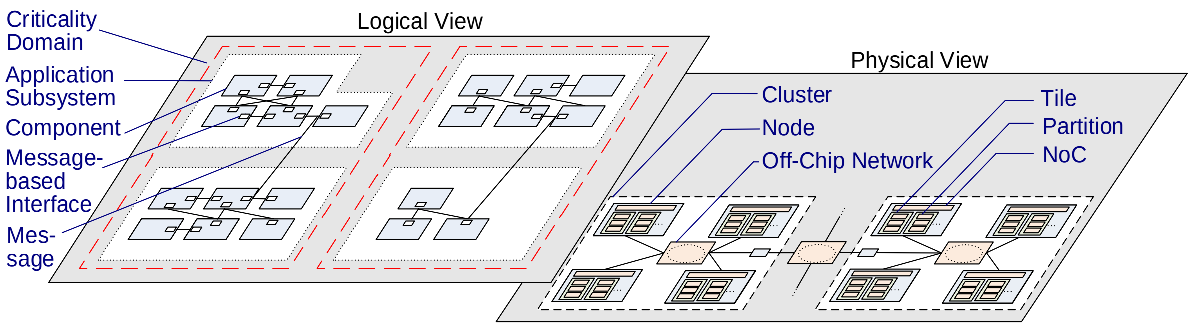


Figure 1: System Structure of Application (Logical View) and Structure of Platform (Physical View)

1.1.1 Structure of the Platform and Resources

The overall system is physically structured into a set of clusters, where each cluster consists of nodes that are interconnected by a real-time communication network in a corresponding network topology (e.g., bus, mesh, star, redundant star, ring). Inter-cluster gateways serve as the connection between clusters.

Each node is a multi-core chip containing tiles that are interconnected by a Network-on-Chip (NoC). Each tile provides a Network Interface (NI) to the NoC and can have a complex internal structure. The NI offers ports each of which serves for the transmission or reception of the NoC's messages. According to application and architecture requirements the NoC has a corresponding topology with interconnected on-chip routers (e.g., mesh, torus, folded torus, hypercube, octagon).

A tile can be processor cluster with several processor cores, caches, local memories and I/O resources. Alternatively, a tile can also be a single processor core or an IP core (e.g., memory controller that is accessible using the NoC and shared by several other tiles).

A chip-to-cluster gateway is responsible for the redirection of messages between the NoC and the off-chip communication network. In analogy to the cluster-level, the NoC exhibits timing properties determined by the communication protocol and the topology of the NoC.

Off-chip and on-chip networks are responsible for time and space partitioning between nodes or tiles respectively. They ensure that a node or tile cannot affect the guaranteed timing (e.g., bounded latency and jitter, guaranteed bandwidth) and the integrity of messages sent by other nodes and tiles.

The processor cores within a tile can run a hypervisor that establishes partitions, each of which executes a corresponding software component (or component for short). The hypervisor establishes time and space partitioning, thereby ensuring that a software component cannot affect the availability of computational resource in other partitions (e.g., time and duration of execution on processor core, integrity and timing of memory).

1.1.2 Logical System Structure of the Application

The overall system is logically structured into criticality levels. Several criticality levels are distinguished in different application domains such as Classes A to E in avionics, ASILA to D in automotive and SIL1-4 in multiple domains according to IEC-61508.

For each criticality level, there can be multiple application subsystems. In the automotive domain, steer-by-wire and brake-by-wire would be examples of subsystems belonging to the highest criticality level (ASILD). An application subsystem can be further subdivided into components, which interact by the exchange of messages via ports.

Each component provides services to its environment. The specification of a component's interface defines its services, which are the intended behavior as perceived by the transmission of messages as a response to inputs, state and the progression of time.

Three types of messages are distinguished based on their timing:

1. *Periodic messages* represent time-triggered communication. Their timing is defined by a period and phase with respect to a global time base.
2. *Sporadic messages* represent rate-constrained communication with minimum interarrival times between successive message instances.
3. *Aperiodic messages* have no timing constraints on successive message instances and no guarantees with respect to the delivery and the incurred delays.

1.1.1 Namespace

Based on the structure of the application and platform, we introduce the following namespace:

$$\underbrace{Criticality.Subsystem.Component.Message}_{Logical\ Name\ (Application)} = \underbrace{Cluster.Node.Tile.Port}_{Physical\ Name\ (Platform)}$$

Examples of names are as follows:

- ClassA.FlightControl.SensorComponent1.Altitude
- ClassA.FlightControl.SensorComponent1.Velocity
- ClassD.Cabin.SensorComponent1.Temperature
- FuselageCluster.Node1.ARMCore1.Port0

Components are only aware of logical names, whereas the platform requires physical names for the routing of messages. The conversion between logical and physical names can occur using a translation layer (in software or hardware) between the components and the communication system. Alternatively, components and messages can be hard-bound to the platform by fixing the translation to the physical namespace at development time.

1.1.2 Mapping of Application to Platform

In order to provide the services, components require resources of the underlying platform as identified in the physical system structure. Each component must be assigned to a partition with suitable computational resources (e.g., CPU time, memory). Messages must be mapped to the communication networks with suitable timing and reliability properties. Since components can be mapped to partitions residing on different nodes and even different clusters, messages must be transmitted over different on-chip and off-chip networks.

Virtual Links (VLs) are an abstraction over these networks and hide the physical system structure of the platform from the components. The timing and reliability of the VL is determined by the properties of the constituent physical networks.

A VL is an end-to-end multicast channel between the output port of one sender component and the input ports of multiple receiver components. This end-to-end connection is identified using a Virtual

Link ID (VLID), which implicitly defines the source port, the destination ports, the path on the on-chip and off-chip networks, the message with its semantic content and the traffic type (i.e., periodic or sporadic) and the message timing.

VLID	Data

Table 2: Message Format: Periodic or Sporadic Message on Virtual Link

Time-triggered VLs serve for the time-triggered transmission of periodic messages at the specified period and phase with respect to a global time base. *Rate-constrained VLs* establish the transport of sporadic messages with minimum interarrival times. A rate-constrained VL also has a priority that determines how contention with other rate-constrained VL is resolved. Rate-constrained communication guarantees sufficient bandwidth allocation for each transmission with defined limits for delays and temporal deviations.

Aperiodic messages do not require VLs, but are subject to a connectionless transfer. Therefore, each aperiodic message must include naming information for routing through the network (see Table 3).

Logical Name of Sender	Physical Name of Receiver	Data

Table 3: Message Format – Aperiodic Message with Connectionless Transfer

The one-to-one mapping between ports and VLs enables the system to determine the parameters of a message (e.g., timing, receivers) by having either the VLID or any of the sender or receiver ports of the VL. As a consequence the gateways and NIs are able to establish the protocol-specific addresses for each network. Conceptually we pair each message with a VLID in order to extract the required address information.

For instance when it comes to the end-to-end path of a periodic or aperiodic message, the communication will be triggered at the NI by writing a message to the respective port (which resides physically at the NI). Based on the portID, the NI knows the physical address of the destination and generates a protocol-specific NoC address. In case the destination is physically located on the same node, the destination of the target-tile will be generated. Otherwise, the message, including the VLID will be redirected to the gateway. The off-chip path will then be generated at the gateway based on the VLID. In case the message is destined to a tile in another node, the on-chip/off-chip gateway will generate the address to the respective target node, while the on-chip/off-chip gateway on the target-node will generate another protocol-specific NoC address before the message enters the on-chip network.

In case of aperiodic messages the procedure is similar, but instead of VLIDs the physical address of the destination must be used. Figure 2 depicts the procedure as well as the address translations graphically.

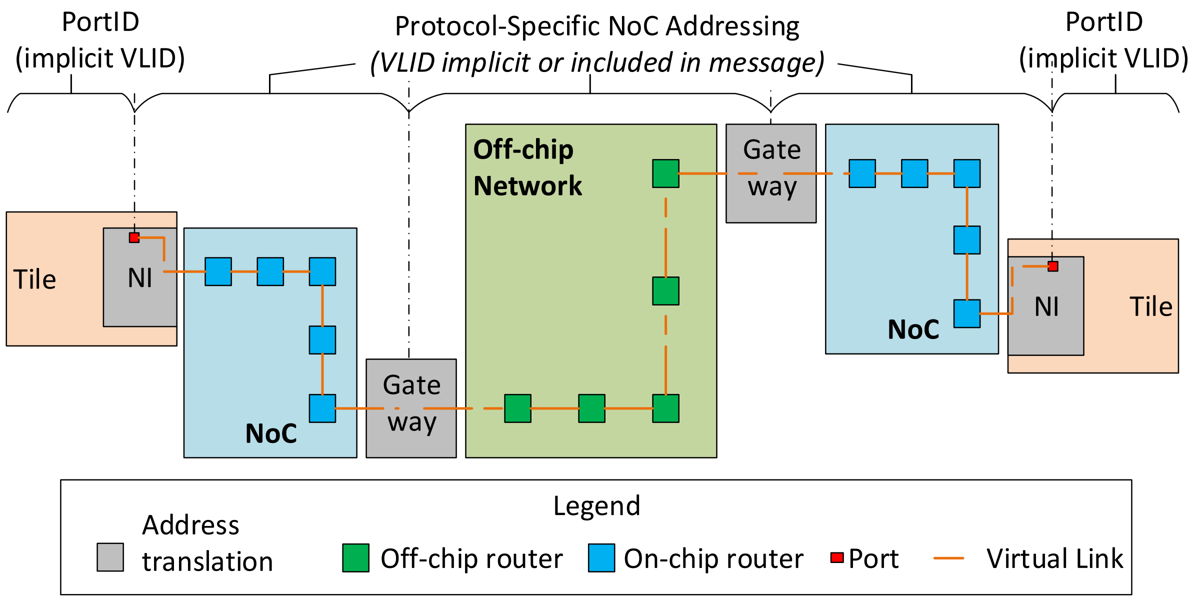


Figure 2: Address domains

1.2 Waistline Structure of Services

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 Figure 3. This waistline structuring of services is inspired by the Internet, where the Internet Protocol (IP) provides the waist for different communication technologies and protocols. Towards the bottom, a variety of implementation choices is supported. IP can be implemented on Ethernet networks, ATM networks, different wireless protocols, etc. Towards the top, different refinements to higher protocols depending on the application requirements occur. IP can be refined into UDP or TCP, thereafter into HTTP, FTP, etc.

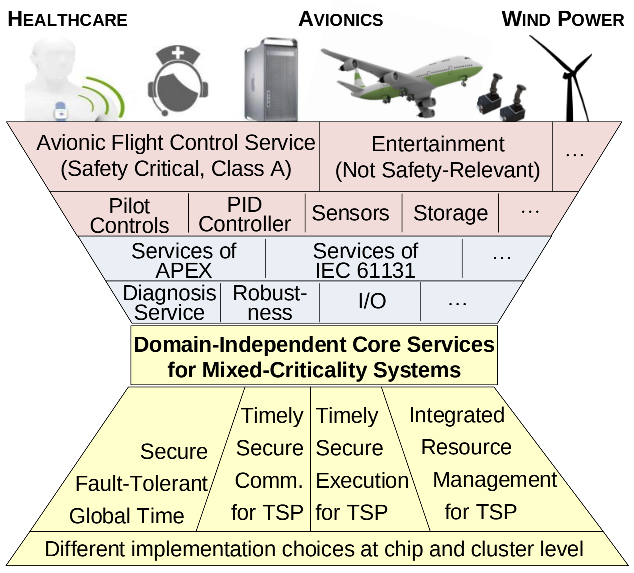

HEALTHCARE
AVIONICS
WIND POWER
Avionic Flight Control Service (Safety Critical, Class A)
Entertainment (Not Safety-Relevant)
…
Pilot Controls
PID Controller
Sensors
Storage
…
Services of APEX
Services of IEC 61131
…
Diagnosis Service
Robust-ness
I/O
…
Domain-Independent Core Services for Mixed-Criticality Systems
Secure Fault-Tolerant Global Time
Timely Secure Comm. for TSP
Timely Secure Execution for TSP
Integrated Resource Management for TSP
Different implementation choices at chip and cluster level

Figure 3: Waistline Structure of Services

In a similar way, the services of the DREAMS architecture are structured. 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. DREAMS is not restricted to specific protocols (e.g., TTNoC and Spidergon NoC as used in WP2), but any protocol providing the core services is a suitable foundation for the DREAMS architecture. In analogy, variability increases towards the waistline's top where the application services are implemented. Platform services can be successively refined and extended to construct more specialized platform services.

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., TSP) of the architecture.

1. *Secure and fault-tolerant global time base:* The global time service 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 rationale 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.

2. *Timely and secure communication services for time and space partitioning*: DREAMS provides services for the message-based real-time communication among components. The DREAMS communication services establish end-to-end channels over hierarchical, heterogeneous and mixed-criticality networks respecting mixed-criticality safety and security requirements. Based on an intelligent communication system with a priori knowledge about the allowed behaviour of components in the value and time domain, DREAMS ensures TSP. The shared memory model is supported 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 TSP of the message-based network infrastructure ensures segregation.

3. *Timely and secure execution for time and space partitioning:* For the sharing of processor cores among mixed criticality applications, including safety-critical ones, partitioning OSes and hypervisors (e.g., XtratuM and KVM) are used, which ensure TSP for the computational resources. The scheduling of computational resources (e.g., processor, memory) in DREAMS ensures that each task obtains not only a predefined portion of the computation power the processor core, but also that execution occurs at the right time and with a high level of temporal predictability. On one hand, DREAMS supports static scheduling, where an offline tool creates a schedule with pre-computed scheduling decisions for each point in time. In addition, we support dynamic scheduling by employing a quota system in the scheduling of tasks in order to limit the consequences of faults. Safety-critical partitions establish execution environments that are amenable to certification and worst-case execution time analysis, whereas partitions for non safety-critical partitions provide more intricate execution environments (e.g., based on Linux). In addition, the separation between safety-critical and non safety-critical applications is supported using dedicated on-chip tiles with respective OSes.

4. *Integrated resource management for time and space partitioning:* DREAMS provides services for 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: resources are monitored individually with abstract information provided to global resource management (GRM). If significant changes should demand adaptation, the GRM takes decisions on a system-wide level, based on offline computed configurations, with orders, such

as bandwidth assignment, or scheduling parameters for all resources, which are controlled by local resource management (LRM). Thus, system-wide constraints, such as end-to-end timing, reliability, of energy integrity, can be addressed without incurring the complexity and overhead of individual negotiations among resources directly.

The distinction between mandatory core services and optional higher services allows to prevent deep service chains that would make real-time guarantees difficult and increase the level of uncertainty. The modular DREAMS architecture introduces a minimal set of services for safety-critical subsystems for ensuring the required properties of the DREAMS architecture. Application with less stringent timing and certification requirements can use optional services with increased functionality and flexibility.

The DREAMS architecture supports the information exchange between safety-critical and non-safety critical subsystems. While the information flow from safety-critical towards non safety-critical parts is supported with no restriction, the reverse direction requires restrictions, namely the separation of interactions by communication channels with temporal and spatial partitioning. The DREAMS technologies contribute hardware and software solutions for this constraint.

1.3 Architectural Building Blocks for the Provision of the Platform Services in Networked Multi-Core Chips

The mapping of the DREAMS platform services of the waistline architecture to the networked multi-core chips is depicted in Figure 4.

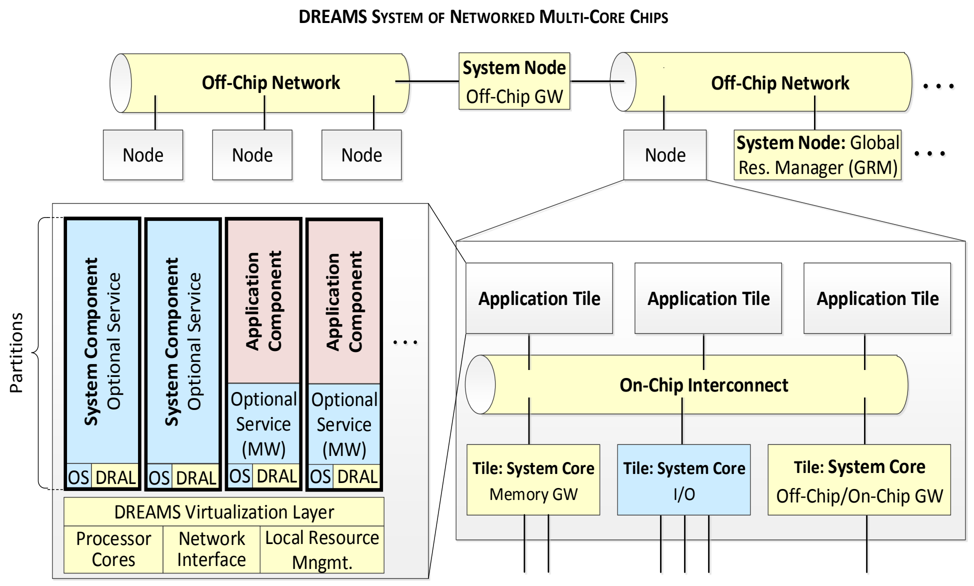


Figure 4: Realization of Platform Services in Networked Multi-Core Chips (core services in yellow, optional services in blue, application services in red)

1.3.1 Building Blocks for Core Services

The ***core communication services*** are realized at the chip-level by the (1) network interfaces, (2) the on-chip interconnect, (3) memory gateways and (4) on-chip/off-chip gateways. The network interface acts as the injection point for messages (and their constituting packets and flits) generated by a tile or core. The on-chip interconnect transports the messages between network interfaces inside one chip. The memory gateway establishes access to external memory (e.g., DRAM) and supports the shared memory paradigm on top of the message-based NoC. An on-chip/off-chip gateway relays selected messages from the NoC to an off-chip network and vice versa, while performing the necessary protocol transformations. At off-chip level, the (1) off-chip networks and (2) off-chip gateways belong to the core communication services. Each cluster has a corresponding off-chip network, where the networks of different clusters can be connected through an off-chip gateway.

The ***core execution services*** are realized by a virtualization layer inside a tile. Either each processor core run its own hypervisor or the virtualization layer manages the entire tile including one or more processor cores. The virtualization layer establishes the partitions for the execution of components with guaranteed computational resources. Within each partition, an operating system and a DREAMS Abstraction Layer (DRAL) are deployed to provide software-support for utilizing the platform services from the application software (e.g., including communication drivers, drivers for time services, domain-specific APIs such as ARINC653).

The ***resource management services*** are realized by a Global Resource Manager (GRM) in combination with local building blocks for resource management. A DREAMS system contains a single GRM, which can be realized by a single node or a set of nodes for improved fault-tolerance and scalability. The GRM performs global decisions with information from local resource monitors. It provides new configurations for the virtualization of 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.

Three local building blocks for resource management are distinguished: (1) Local Resource Managers (LRMs), (2) Local Resource Schedulers (LRSs) and (3) Resource Monitors (MON). These local resource management building blocks are located at the individual resources at chip and cluster level. The LRS is responsible for controlling the access to particular resource based on a configuration that has been set by the LRM. Each resource has a corresponding built-in LRS such as the on-chip network interface, the hypervisor layer inside a tile, the memory gateway, the I/O gateway, the on-chip/off-chip gateway and the off-chip network interfaces. For example, the LRS in the on-chip network interface is responsible for dispatching time-triggered messages according to the schedule tables in the network interface and for traffic shaping of sporadic messages.

The Local Resource Scheduler (LRS) performs the runtime scheduling of resource requests (e.g., execution of tasks on processor, processing of queued memory and I/O requests). The LRS in DREAMS will support different scheduling policies (e.g., dispatching of time-triggered actions, priority-based scheduling).

The LRMs adopt the configuration from the GRM at particular resources (e.g., processor core, memory, I/O). It is responsible for mapping global decisions to the local scheduling policy of the LRS. In some cases LRMs are able to take decisions for local reconfiguration.

The Resource Monitor (MON) monitors the resource availability (e.g., energy). Resource monitors also observe the timing of components (e.g., detection of deadline violations), check the application behavior (e.g., stability of control) and perform intrusion detection. Small changes will be handled locally, while significant changes will be reported to the GRM, who in turn can provide a different configuration at system-level.

1.3.2 Building Blocks for Optional Services

On top of the core services, the optional platform services establish higher-level capabilities for certain domains (e.g., control systems, multimedia). Optional services are capabilities that are not needed in all targeted applications, thus they can be integrated when needed or omitted if unnecessary to minimize resource consumption. In addition, using optional services we can support complex platform services for non safety-critical applications without affecting certification of safety-critical application subsystems.

Optional services are subject to partitioning and segregation performed by the core services. Hence, any fault of an optional service only affects the application and other optional services building on top of it. This fault containment is a key enabler for modular certification, because optional services not used by an application subsystem do not need to be considered in its certification.

One can distinguish three implementation choices for optional services:

1. ***System core***: Optional services are implemented as self-contained IP cores with a message-based interface towards the NoC. The segregation is established by the NoC.
2. ***System component in a partition***: An optional service is realized as a component within a partition. The optional service is provided to components in other partitions inside the tile using inter-partition communication mechanisms of the virtualization layer. In addition, the platform service can be made available using the NoC.
3. ***Middleware in a partition***: The platform service is realized as middleware within a component and provides services to the application component within the same partition.

1.3.3 Building Blocks for Application Services

An application service is realized by an application component inside a partition. The application service provides its service to other components using the core communication services, where a partition with an application service is a communication end point.

1.3.4 Technology Independence of Architectural Style

The architectural style including the logical system structure, the physical structure and the architectural services is not restricted to a particular implementation technology. Different types of processors, on-chip networks, off-chip networks and operating systems can serve as the starting point for the establishment of the DREAMS architecture.
At the chip-level, we can distinguish the following categories of instantiations of the architectural style depending on the type of the underlying multi-core processor:

- ***Shared memory-based chip architectures*** are a special case of the architectural style with a multi-core chip containing only a single tile. This single tile contains multiple cores that interact via a shared memory. Instead of realizing the off-chip gateway via the NoC, there exists a dedicated (memory-mapped) I/O peripheral for the off-chip network interface. Instantiations of the DREAMS architectural style using PowerPC P4080 and x86 belong to this category.
- ***NoC-based architectures*** are another instantiation where the multicore architecture contains tiles each of which contains only a single core. The tiles are interconnected by a message-based NoC. Instantiations of the DREAMS architectural style using TTSoC belong to this category.
- ***Full-scale architecture instantiations*** provide multiple tiles with multiple cores per tile. The tiles are interconnected by an NoC. Each tile can contain a shared memory for the interaction within the tile. The interaction between the tiles is message-based, although a shared memory interaction can be realized on top of message passing based on a tile serving as a memory gateway (e.g., DDR controller).

Likewise, different scales can be distinguished at the cluster level including single-cluster and multi-cluster DREAMS mixed-criticality systems. The latter types of systems depend on off-chip gateways in-between the off-chip networks of different clusters.
Based on the different integration levels, the architectural style supports different types of communication mechanisms:

1. The intrapartition communication between tasks within a partition is the responsibility of the application software or guest operating system within the partition and thus transparent to the DREAMS architecture.
2. Interpartition communication between partitions on the same tile is supported by the hypervisor and can be implemented using the local shared memory within the tile.
3. Interpartition communication between partitions on different tiles of the same chip occurs using the message-based NoC. Shared memory interactions are possible using a memory gateway and shared memory accesses on top of message passing.
4. Interpartition communication between partitions on different chips occurs using the on-chip/off-chip gateway. The gateway is accessed using either the NoC or via the tile's shared memory depending on whether a single-tile or multi-tile node is considered.

The application interface for interpartition communication is identical, regardless of which communication type (2 to 4) is used.

2 Fault Assumptions

The fault hypothesis specifies assumptions about the types of faults, the rate at which components fail and how components may fail [10]. The fault hypothesis is a central part in any safety-relevant system and provides the foundation for the design, implementation and test of the fault-tolerance mechanisms [11].

The consideration of security mechanisms for the DREAMS architectural style requires a clear definition of threats. Section 3.3 provides this information.

2.1 Fault containment regions

A Fault Containment Region (FCR) is a subsystem that operates correctly regardless of any arbitrary logical or electrical fault outside the region [10]. A FCR is a set of subsystems that share one or more common resources that one single fault may affect. Based on the distinction between design faults and physical faults, one can distinguish corresponding FCRs (see Table 4).

Fault		**Fault Containment Region**	**Containment Coverage (Correlated Failures per Hour)**
Design Fault	Design fault of the application component in the partition or the guest OS of the partition	Partition	$< 10^{-9}$
	Replicated design fault in copies of a component or the same guest OS	Multiple partitions containing the same application component or the same guest OS	$< 10^{-9}$
	Design fault of virtualization layer in an application tile	Application tiles with the virtualization layer	$< 10^{-9}$
	Design fault of a system tile (e.g., I/O gateway, memory GW)	System tile	$< 10^{-9}$
	Design fault of on-chip network (including network interfaces and on-chip routers)	Nodes with the on-chip network	$< 10^{-9}$
	Design fault of off-chip network	Cluster with the off-chip network	$< 10^{-9}$
	Design fault of global resource manager	Dynamically reconfigurable part of the platform and respective application subsystems	$< 10^{-9}$

Physical Fault	Affected physical resource of a node (e.g., power supply, clock source, on-chip network)	Node	$< 10^{-9}$
	Affected physical resource of the off-chip network (e.g., short circuit of physical link, clock source of router)	Off-Chip router with corresponding physical links	$< 10^{-9}$
	Affected physical resource only required for a tile (e.g., local memory)	Tile	10^{-5} to 10^{-6}

Table 4: Fault Containment Regions

An FCR restricts the immediate impact of a fault, but fault effects manifested as erroneous data can propagate across FCR boundaries. For this reason the system must also provide error containment [10] to avoid error propagation by the flow of erroneous messages. An Error Containment Region (ECR) is a subsystem of the mixed-criticality system that is encapsulated by error-detection interfaces such that there is a high probability that the consequences of an error that occurs within this subsystem will not propagate outside this subsystem without being detected and/or masked [2]. The error detection mechanisms must be part of different FCRs than the message sender. Otherwise, the error detection mechanism may be impacted by the same fault that caused the message failure.

2.1.1 Fault Containment Regions for Design Faults

Design faults include hardware and software faults that are introduced during the development of the platform and the application.

For design faults, we can distinguish between the faults affecting the DREAMS platform (e.g., system tiles, DREAMS virtualization layer, communication networks) and the application software within the partitions.

For design faults affecting the application software and guest operating systems, we regard a ***partition*** as a FCR. Mechanisms for temporal and spatial partitioning of the DREAMS virtualization layer provide design fault containment between partitions. If a software component is replicated along multiple partitions (possibly located on multiple tiles or nodes) as part of a fault-tolerance concept, the FCR includes all partitions with distributed replicas of the software component. Replicated software components cannot be assumed to fail independently, since all replicas of a software component are based on the same programs and use the same input data.

The role of software components as design FCRs holds also in case of software diversity. When design diversity is applied for addressing common mode failures, replicas are necessarily different and ideally employ different specifications in addition to separate implementations. Consequently, we denote these diverse replicas as separate software components. Nevertheless, the decision of regarding the respective partitions with these software components as different design FCRs depends on the independence of the diverse software versions. Practical analyses of software diversity have demonstrated that diverse implementations often exhibit correlation with respect to design faults.

Since all partitions hosted on a tile depend on the correct behaviour of the DREAMS virtualization layer, the partitions cannot be assumed to be unaffected by a fault affecting the virtualization layer. Therefore, ***all tiles on which a particular virtualization layer is deployed*** represent a common FCR for design faults affecting the virtualization layer. The virtualization layer is thus a critical resource in the mixed-criticality system. It is thus necessary to ensure the absence of software faults in the virtualization layer. In particular, the system software needs to be designed for validation and kept simple in order to permit a thorough validation (e.g., including formal verification). Moving functionality from the virtualization layer into the partitions is a viable strategy to achieve this goal, which is similar to the well-known concept of micro-kernels in operating system design.

A **system tile** is an FCR for a design fault of the respective higher platform service (cf. Section 1.2). An example is a design fault of an input/output gateway, which affects the corresponding higher platform service provided on top of the core platform services of DREAMS.

The entire **node** is an FCR for design faults for shared resources that are required for the correct operation of the node. For example, the on-chip network is a critical resource for the entire node where a design fault of a network interface or router has the potential to disrupt the timely communication of any tile.

In case of design faults affecting an off-chip communication network, the respective **cluster** is a FCR. Faults of the global resource manager can affect the **dynamically reconfigurable parts** of the platform and the respective applications, whereas static subsystems remain unaffected.

2.1.2 Fault containment Regions for Physical Faults

A physical fault affects physical resources, such as mechanical or electronic parts. Physical faults typically originate from conditions that occur during operation. Examples are physical deterioration (i.e. wear-out) and external interference through physical phenomena (e.g., lightning stroke). Early and premature wear-out failures are caused by the displacement of the mean and variability due to manufacturing, assembly, handling, and misapplication.

To form a fault containment boundary around a collection of hardware elements, one must provide independent power and clock sources and additionally electrical isolation and spatial separation. These requirements make it impractical to provide more than one FCR within a **node** at a safety-critical rigor (at a containment coverage with a probability of *correlated* failures of 10^{-9} failures per hours).

We also regard each **off-chip router** with the corresponding physical links to the nodes as a FCR. For example, a central guardian of a time-triggered network (e.g., TTEthernet switch) serves as a FCR [20].

For physical faults, the hardware approach can provide certain containment coverage by providing spatial separation of the tiles and cores and multiple clock domains and pin-out (e.g., grounding) on the chip layout (e.g., for SEEs [21]). These on-chip FCRs for physical faults (i.e., **tiles**) work only at single chip failure probabilities (e.g., around 10^{-5} to 10^{-6} correlated failures per hour [22]).

Physical fault containment and design fault containment are orthogonal properties. Physical fault containment does not assure design fault containment and vice-versa. For instance, one may use two separated chip processors (two FCRs for physical faults) to implement a function but both can fail simultaneously due to a single design fault on the software. In the same way, a hypervisor can assure design fault containment for two independent operating systems within the same chip and a single physical fault can make both fail.

2.2 Failure modes

The assumed failure modes include those identified by IEC-61508-2, according to which transmission errors, deletion, corruption, delay, repetitions, masquerading and insertion need to be addressed [13]. Furthermore, additional critical failure modes for mixed-criticality systems are introduced.

The following failure modes are distinguished:

- **Babbling idiot failure:** This failure occurs when an application core or an off-chip router starts sending untimely messages (e.g., insertions according to IEC-61508-2), possibly generating a high traffic load by generating more messages than specified.
- **Delay:** Faulty core or off-chip router can delay the transmission of messages.
- **Masquerading:** A masquerading failure is an erroneous core that assumes the identity of another core. In case of periodic and sporadic communication, a faulty core sends messages with the incorrect virtual link identification. For aperiodic messages, the core will send messages with an incorrect logical namespace.
- **Component crash**: The crash failure occurs when the DREAMS chip or the off-chip router exhibits a permanent fault and produces no outputs.
- **Link failures:** The link failure occurs when the link exhibits a permanent or transient failure and fails to redirect a message. In combination with the component crash, this failure corresponds to the transmission error according to IEC-61508-2.
- **Omission:** An omission failure is a transmission failure where a sender is not able to generate a message and/or a receiver is not able to receive a message. This failure corresponds to the deletion according to IEC-61508-2.
- **Slightly-off-Specification (SOS)**: Slightly-off-specification failures can occur at the interface between the analog and the digital world in the value and time domain. For example, consider the case that the specification requires every correct node to accept an analog input signal if it is within a specified receive window of a parameter (e.g., timing, frequency, or voltage). Every individual node will have a wider actual receive window than the one specified in order to ensure that even if there are slight variations in manufacturing it can accept all input signals as required by the specification. These actual receive windows will be slightly different for the individual nodes. If an erroneous FCR produces an output signal (in time or value) slightly outside the specified window, some nodes will correctly receive this signal, while others might fail to receive this signal [20].

2.2.1 Failure Rates and Persistence

Part of the fault hypothesis is a specification of the failure rate of FCRs. In general, a differentiation of failure rate with respect to different failure modes and the failure persistence is necessary. For example, fault injection experiments [23] have shown that restrictive failure modes, such as omission failures, are more frequent by a factor of 50 compared to arbitrary failures.

Related to the failure rates in industrial communication the residual error rate needs to be calculated according to IEC 61784-3. The residual error rate needs to stay below 1% of the PFH of the target SIL according to IEC 61508.

Also, failure persistence is an important factor in the differentiation of failure rates. In the temporal domain a fault can be transient or permanent. Whereas physical faults can be transient or permanent, design faults (e.g., software errors) are always permanent. While transient failures disappear without an explicit repair action, permanent failures prevail until removed by a maintenance engineer (e.g., software update in case of a software fault, replacement or repair of hardware in case of a hardware fault).

The permanent failure rate of a FCR with respect to hardware faults is typically considered to be in the order of 100 FIT, i.e., about 1000 years [20]. Motivated by literature on SER we assume that the transient failure rate of a FCR with respect to hardware faults is in the order of 10.000-100.000 FIT [24].

2.3 Threats

The DREAMS architecture defines four different core services as shown in Figure 3. These core services have different security requirements which have been already stated in D1.1.1 and there exist different potential attacks and threats which are described in this section. Threat Models as well as threats and attacks which are related to the cluster-level are described in more detail in D3.3.1, e.g., communication services, global time services and resource management services.

2.3.1 Threat Models

A threat model describes and analyses the security risks associated with the system. It identifies potential threats to the system as well as the vulnerabilities in the system which can be exploited.

There are four important questions which have to be considered while creating a threat model. [25]

1. Who is the attacker?

   There are two general types of attacker, a user and an application. Each one of them could be authorized or unauthorized to access a certain component. It is not always necessary to distinguish the attackers as users and/or applications. Considering attacks on the network layer (OSI Layer 3), the attacks are independent of the application layer (OSI layer 7). Hence, in the threat model for communication services, only the “internal” and “external” attackers are considered.

2. What is attacked?

   A system has different parts which could be attacked. These parts of the system are components and applications.

3. Where is the attacker?

   An attacker can attack a system from different locations. The attacker could be inside the system or he can attack the system from outside.

4. How is the attack performed?

   The attacker has different capabilities to perform an attack. Depending on the questions “Who is the attacker?”, “What is attacked?” and “Where is the attacker?”, the attacker has various options to realize an attack.

2.3.2 Threat Analysis for Communication Services

There are different types of communication services in the DREAMS architecture: the on-chip communication and the off-chip communication separated by the on-chip/off-chip gateway (refer to Figure 5). As described in section 1.1, there is a physical and a logical view of the communication system. This section focuses on the physical view. Since there is a distinction between on-chip and off-chip communication, the security aspects can also be divided into on-chip security and off-chip security with different threats which are discussed as follows.

The distinction between on-chip and off-chip security allows the division of attacks on the on-chip and the off-chip communication. This leads to the distinction between internal and external attackers which is based on [18]. The main difference between internal and external attackers is the access point to the system and the knowledge about secret information. The access point of an internal attacker is inside of a trusted part of the system. He has access to the cryptographic keys on the network layer including access to other secret information. Hence, he can generate valid messages and can act as a legal part of the system. In contrast to an internal attacker, an external attacker has no access to the trusted part of the system and does not know the cryptographic keys. Thus, an external attacker can intercept and replay existing messages but cannot generate new ones.

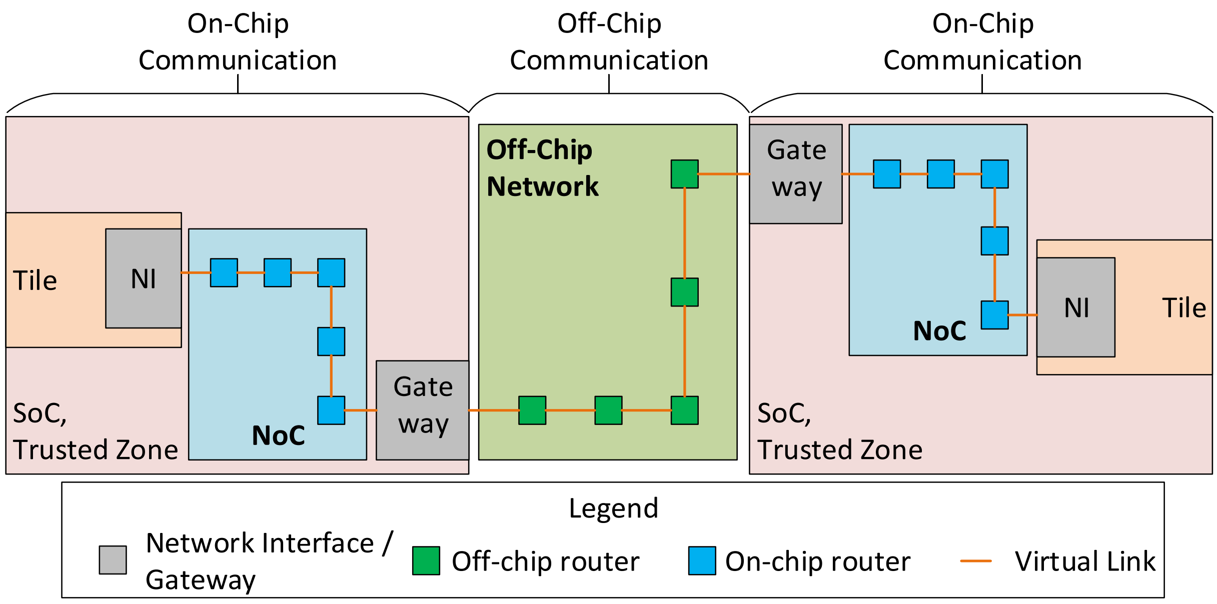


Figure 5: On-Chip/Off-Chip Communication

In DREAMS, the communications take place at the on-chip network (NoC) and at the off-chip network respectively. These two types of communications correspond to the internal and external attackers respectively. An attacker who has access to the on-chip communication is an internal attacker and an attacker who has only access to the off-chip communication is an external attacker. Therefore it is assumed that the SoC, including the NoC and the gateway, is a trusted zone and is inaccessible to an external attacker.

Hence, an internal attacker has access to the NoC and to the other parts of the SoC, e.g., the CPU-cores and the memory. If the cryptographic keys are stored in the memory which is accessible to all components connected to the NoC, then the attacker also has access to these keys.

An external attacker has only access to the off-chip network. He can intercept and replay previously sent messages, but cannot read encrypted messages. Also he cannot generate new legal messages.

The gateway between the on-chip and the off-chip network forms the border among the two network types. Therefore all communication leaving the gateway towards the off-chip network leaves the trusted zone of the DREAMS architecture. Hence, the gateway separates an internal attacker from an external attacker (Figure 4).
There are several types of attacks which can be performed on the communication services of the DREAMS architecture. An attacker can perform sniffing attacks, denial-of-service attacks, spoofing attacks, man-in-the-middle attacks, packet injection attacks and replay attacks. External and internal attackers have different opportunities performing an attack. These opportunities and the impact of the attacks are described in D3.3.1.

2.3.3 Threat Analysis for Global Time Services

The global time services should ensure that every local clock in the system has “about the same value” at “about the same points in real-time” (refer to section 2, Core Platform Services – Global Time).There are two main attack targets on the global time services. On the one hand there are attacks against the clocks or the time values in the components itself, on the other hand there are attacks against the time synchronization.

The attacks on the time synchronization are covered in the threat analysis for communication services. Authorized users as well as unauthorized users could perform attacks on the synchronization process. An attack could aim on a single target with the result that one component gets a false time value or it could aim on the entire synchronization process with the result that no component gets the proper time value.

A single target can be attacked with man-in-the-middle, packet injection and replay attacks. In a man-in-the-middle attack, the attacker can change the time value of the synchronization message before sending it to the receiver. In a packet injection attack, the attacker inserts new synchronization messages with false time values. The receiver of the new messages synchronizes to the false time value. In a replay attack, the attacker sends an old message again to the receiver and the receiver uses the old time value. Man-in-the-middle and packet injection attacks are only possible for authenticated users having access to keys needed to generate new valid messages. An unauthorized user can only perform replay attacks because he cannot generate new valid messages.

The entire synchronization process can be attacked by performing a denial-of-service attack on the master clock. Spoofing attacks can attack both a single target and the entire synchronization process. Denial-of-service attacks are possible for authenticated and unauthenticated users. A spoofing attack is only possible for an authenticated user if they have access to the needed keys masquerading as another user.

The impact of an attack against the clocks in the components itself is similar to the attacks on the time synchronization. However, the communication process for the time synchronization is not the objective of this type of attacks. Attacking a clock in a component acting as a slave in the synchronization process only affects the behavior of this component, e.g., the component sends untimely messages or causes untimely actions. If an attacker changes the master clock all clocks in the system synchronizing with the master clock get the false time value. This might lead to measurements taken at the false point in time or to incorrect behavior of the system relating to real-time. Changing the clock values needs additional access privileges and can be performed only by an authorized user or an attacker which can masquerade as an authorized user.

2.3.4 Threat Analysis for Resource Management Services

In the DREAMS architecture the resource management services are realized by a Global Resource Manager (GRM) as explained in D3.2.1. In addition to the GRM, there are Local Resource Managers (LRM), Local Resource Schedulers (LRS) and Resource Monitors (MON) located in the different Tiles. The GRM performs global decisions by selecting configurations. This decisions are based on the information received from the LRM. Decisions for new configurations are sent back to the LRM. The LRM gests information from the MON and maps the global decisions from the GRM to the LRS.

There are several attacks on the resource management services. On the one hand there are attacks against the resource management components. An attacker could masquerade as one of the GRM, LRMs, LRSs or MONs. Acting as a trustworthy GRM or LRM, an attacker apply wrong or invalid global or local configurations. If an attacker acts as an LRS, he can select other scheduling tables or he can use invalid scheduling parameters. The MON provides monitoring services. Hence, an attacker could send wrong availability, energy or error information to the LRM. In addition, there are pre-computed configurations. If an attacker can change these offline-computed configurations, a genuine resource management component selects wrong configurations. This could lead to wrong configurations of resources, e.g., false partition scheduling tables or false resource budgets. These attacks can only be performed by an authenticated user who is inside of the system. An unauthenticated user has no access to the components.

On the other hand there are potential attacks on the communication process of the resource management services. An attacker could perform sniffing attacks providing him more information about the behavior of the system. He could perform denial-of-service attacks suppressing the availability of a resource management component. Man-in-the-middle, spoofing and packet injection attacks could lead to wrong configurations and a wrong scheduling. The same risk applies for a replay attack, but at least the configuration or scheduling was valid before. Nevertheless, the system or a part of the system will not operate as intended.

2.3.5 Threat Analysis for Execution Services

The execution services provide basic operations to run the system. The service includes the virtualization layer as the software layer that abstracts the underlying hardware and provides virtualization of the CPUs.

The virtualization layer provides different properties that ensure protection against many attacks related to security.

Spatial isolation: The address space of a partition is not accessible to other partitions. No application of one partition can access the data from another partition. Hence, no unauthorized as well as authorized user or application from one partition can attack another partition. There could only be an attacker inside of the partition. Therefore he can only be an application running in the partition or an authorized attacker who can access the partition. But the system architect can define specific shared memory areas between partitions. In these areas, no confidential information should be stored.

Temporal isolation: The temporal isolation ensures that the execution of a partition is independent of the execution of other partitions. Hence, an attacker in one partition cannot prohibit the execution of another partition by performing attacks like sleep deprivation, where an attacker is keeping a partition active, so that he can prevent the calculation of other partitions than the active one form the attacker. Since no unauthorized user can access a partition, only authorized users or applications being inside of the partition can perform such attacks.

Part II
Architectural Services of DREAMS

The DREAMS architectural services

Part II of this document is dedicated to the introduction of the DREAMS architectural services and the related certification strategy. The architectural services are grouped in the four core platform service categories:

1. Communication
2. Global Time
3. Execution
4. Resource Management

An additional category is the group of *Optional Services* providing an example for an optional DREAMS service that builds upon the core services according to the DREAMS services waistline structure shown in Figure 3.

The four core service categories are represented by level one headings. Inside these categories, the services are allocated to service groups and subcategories with level two headings whereas the service descriptions are described in level three section.

1. Core services -Communication
   Group of services A
    1. Subcategory X
        1. Service I
        2. Service II
        3. Service III
        4. …
    2. Subcategory Y
        1. Service I
        2. Service II
        3. …
    3. …

   Group of services B

   …
2. Core services –Global Time

   …

This numbering convention allows the unique identification of each service by the section number.

1 Core Platform Services – Communication

One of the four DREAMS core services categories is communication. This section provides detailed information on the communication core platform services of DREAMS which are grouped into the subcategories and groups as shown in Figure 6.

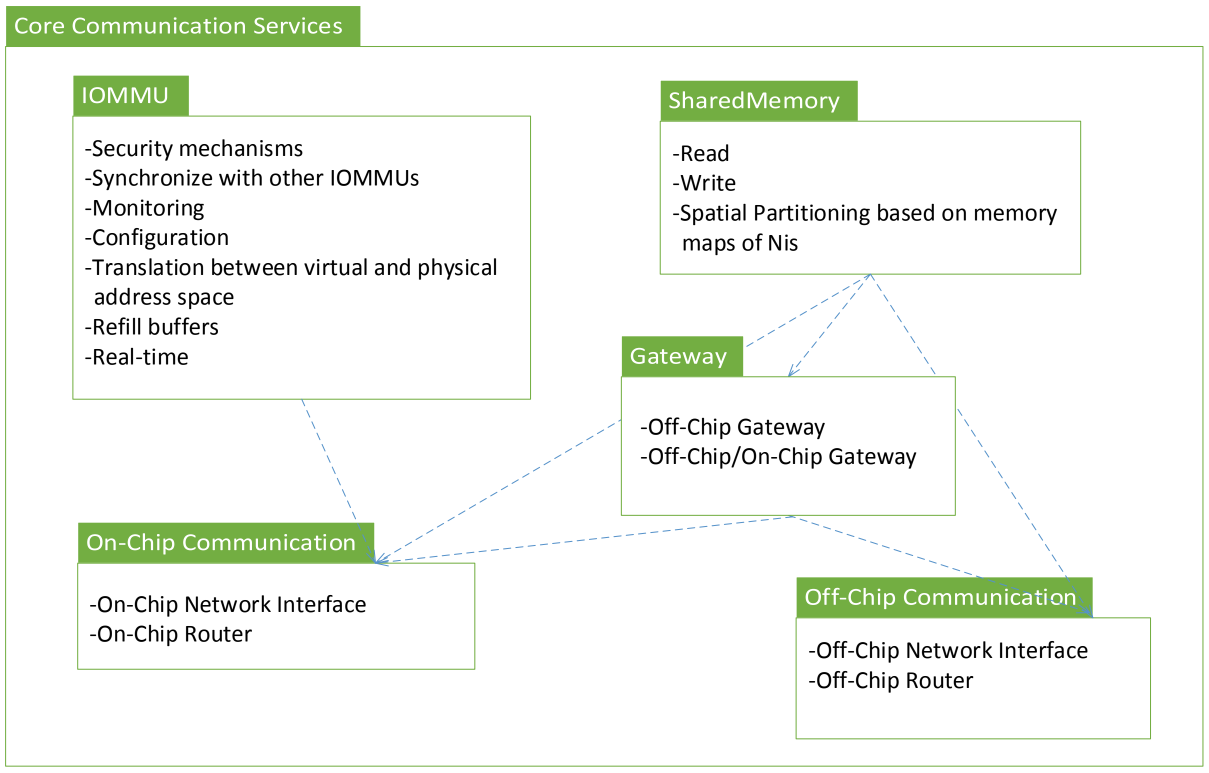

Core Communication Services
IOMMU
-Security mechanisms
-Synchronize with other IOMMUs
-Monitoring
-Configuration
-Translation between virtual and physical address space
-Refill buffers
-Real-time
SharedMemory
-Read
-Write
-Spatial Partitioning based on memory maps of Nis
Gateway
-Off-Chip Gateway
-Off-Chip/On-Chip Gateway
On-Chip Communication
-On-Chip Network Interface
-On-Chip Router
Off-Chip Communication
-Off-Chip Network Interface
-Off-Chip Router

Figure 6: Subcategories of communication

Group of On-Chip Communication Services

The message-based on-chip communication services are realized mainly by the *On-Chip Network Interface (NI), the On-Chip Router* and *On-Chip Physical Links* (See Figure 7). The NI serves as an interface to the NoC for the processing cores by injecting the messages from the cores into the NoC as well as delivering the received messages from the NoC to the cores. Routers on the other hand are responsible to relay the flits from the sender's NI to the destination NIs. The number of input and output units at each router and the connection pattern of these units represent the topology of the on-chip network (e.g., star, ring, spidergon). The physical links act as a glue element among NIs and routers and realize the interconnection among them.

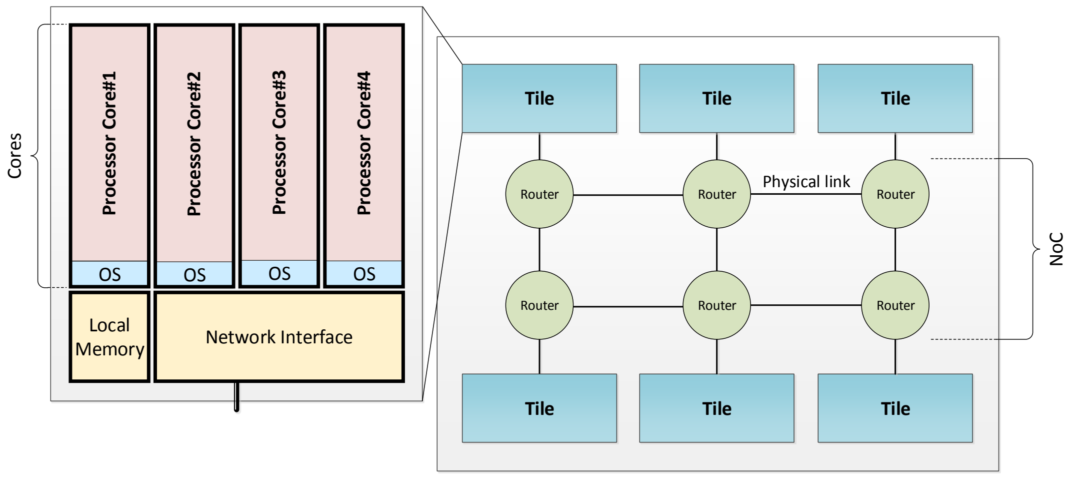


Figure 7: A typical Network-on-Chip with six tiles

In a message-based communication system, by switching the packets, there is no need for circuit switching. This means instead of assigning a static configuration to each router, we use a packet-based router configuration. In other words, to improve the efficiency of the resource allocation, we divide a message into *packets* for the allocation of control state and into flow control digits (*flits*) for the allocation of channel bandwidth and buffer capacity.

Figure 8 shows the units in which network resources are allocated. Since *messages* may be arbitrarily long, resources are not directly allocated to messages, but rather to packets that have a restricted maximum length. This restriction leads to a limited time and duration of resource allocation, which is often important for the performance and functionality of the flow control mechanisms.

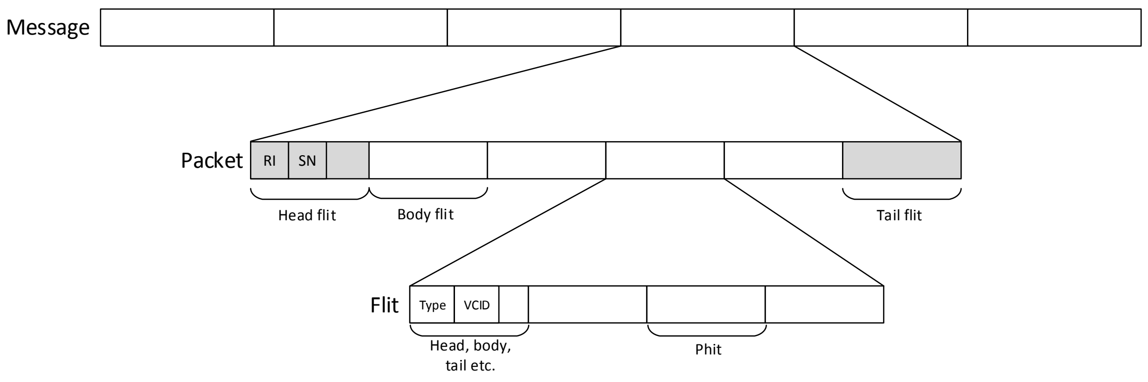


Figure 9 Units of resource allocation at the on-chip network

A flit is the basic unit of bandwidth and storage allocation. Flits carry no routing and sequencing information and thus must follow the same path and remain in order. However, flits may contain a virtual-channel identifier (VCID) to identify which packet the flit belongs to in the system, while multiple packets may be in transit over a single physical channel at the same time. Based on the position of the flit in packet, the flit may be the *head flit* which carries header information, the *body flit* which includes the payload and the *tail flit* which indicates the end of the packet and possibly contains the check sum information for error detection. A flit is further subdivided into one or more physical transfer digits or *phits*, which are the unit of information that is transferred across a channel in a single clock cycle.

The reason for subdividing the packet into the flits is that on the one hand, we would like to make packets large to amortize the overhead of routing and sequencing. On the other hand, we would like to make packets small to permit efficient, fine-grained resource allocation and minimize blocking latency. Introducing flits eliminates this conflict. We gain low overhead by long packets and achieve efficient resource utilization by very flits.

Phits are shaped by the physical link, i.e., the number of bits which can be transferred by the physical link in a single clock will define the phit. Hence the flit will be transferred in multiple clock cycles via the physical link [13].

In order to guarantee bounded delay and low jitter in the network, we define three main priority classes in the architecture, each of which can possibly be composed of different further priorities (see Figure 10). The highest priority class in the network belongs to the periodic messages. Since periodic messages are sent according to the predefined schedule, there is no priority needed between the periodic messages. The second priority class is assigned to sporadic messages. However there can be different levels of priorities among different sporadic messages. In case two sporadic messages of different priorities compete for using a resource, the one of higher priority will win and the other one will wait. Aperiodic messages possess the lowest priority class in the network. There is no guarantees whether and when these messages arrive at the destination. According to the implementation, further priorities for aperiodic message can be defined.

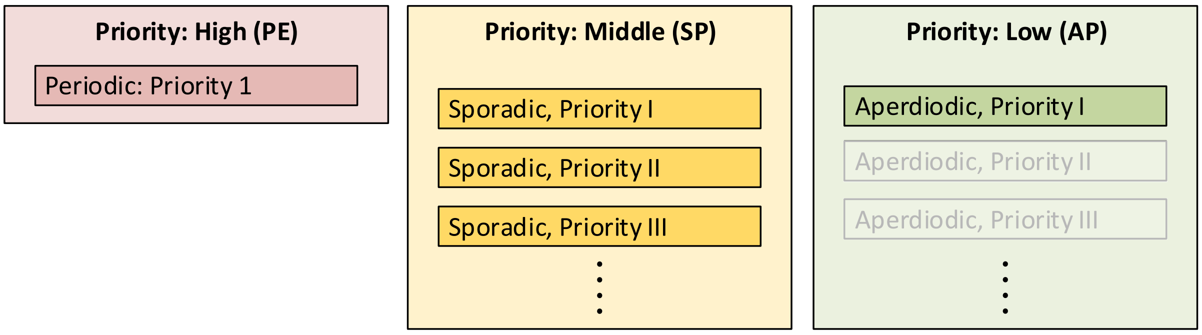


Figure 10: Different priorities within DREAMS

1.1 On-Chip Network Interface

The NI serves as an interface to the NoC for the processing cores by injecting the messages from the cores into the NoC as well as delivering the received messages from the NoC to the cores. In case the NI serves as a sender NI, it determines the path to the destination NIs according to the configuration information and generates the flits including the head flit, the body flits and the tail flit. In case the NI serves as a destination NI, it generates the messages out of received flits and provides the processor cores with the messages.

As shown in Figure 11, the services provided by the on-chip NI can be grouped into two main blocks based on the provided services, the LRS and the NoC interface. As defined in the "Waistline Structure of Service" (cf. Part I, Section 1.2) and the "Resource Management" (cf. Part II, Section 4), the services in the LRS perform the runtime scheduling of resource requests such as allocating bandwidth or processing queued messages.

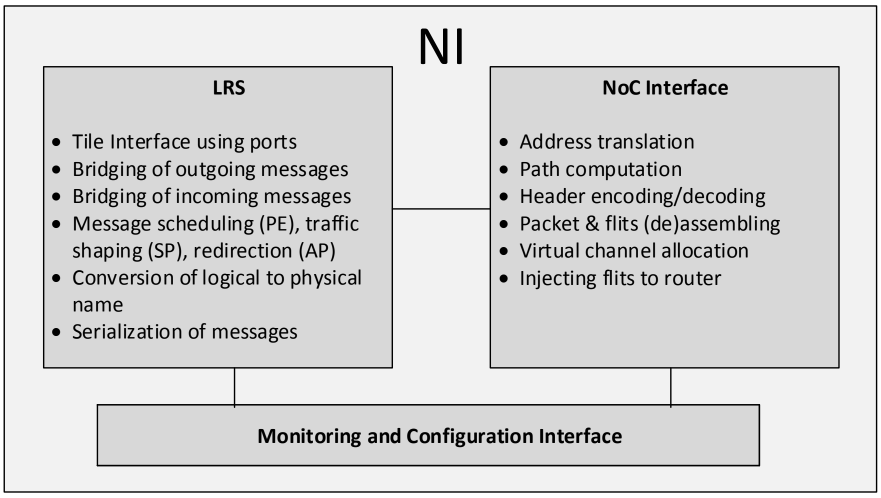

NI
LRS
Tile Interface using ports
Bridging of outgoing messages
Bridging of incoming messages
Message scheduling (PE), traffic shaping (SP), redirection (AP)
Conversion of logical to physical name
Serialization of messages
NoC Interface
Address translation
Path computation
Header encoding/decoding
Packet & flits (de)assembling
Virtual channel allocation
Injecting flits to router
Monitoring and Configuration Interface

Figure 11: Service for on-chip network interface

The services of the LRS control the incoming message traffic from the cores by buffering the messages with lower priority and relaying the messages with higher priority to the NoC Interface, thus providing the support for mixed-criticality systems. Inversely, they accept the messages coming from the NoC, classify them with respect to traffic types and destination ports and provide the respective core with the data.

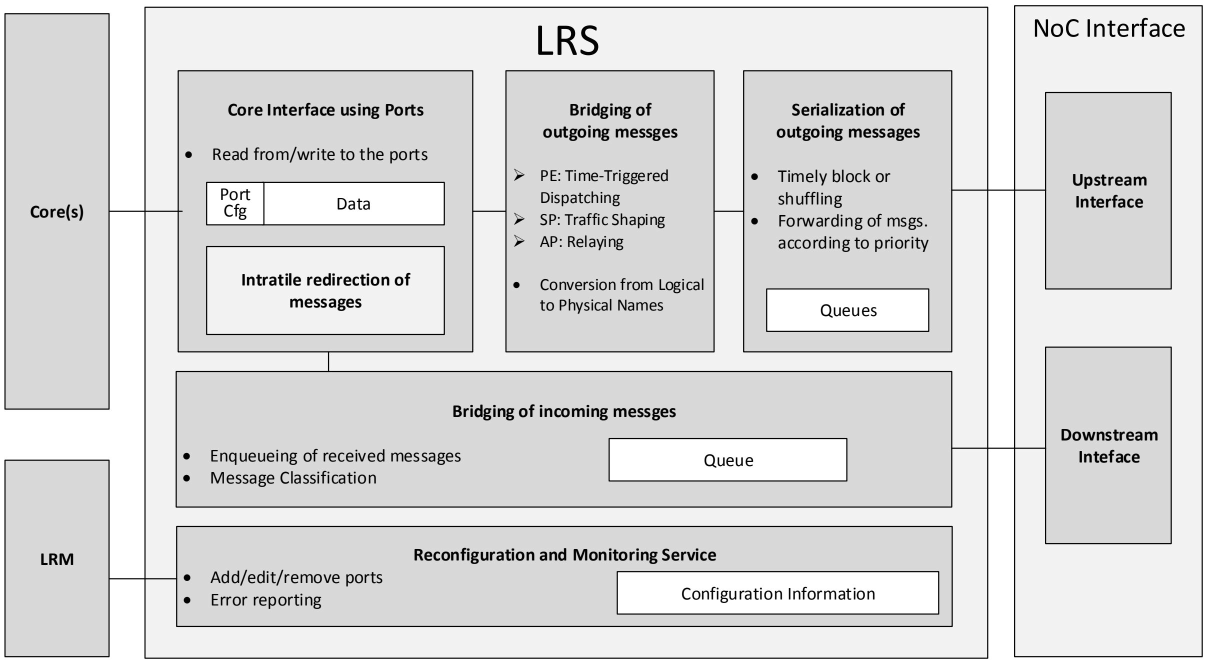

LRS
NoC Interface
Core(s)
LRM
Core Interface using Ports
Read from/write to the ports
Port Cfg
Data
Intratile redirection of messages
Bridging of outgoing messges
PE: Time-Triggered Dispatching
SP: Traffic Shaping
AP: Relaying
Conversion from Logical to Physical Names
Serialization of outgoing messages
Timely block or shuffling
Forwarding of msgs. according to priority
Queues
Upstream Interface
Bridging of incoming messges
Enqueueing of received messages
Message Classification
Queue
Downstream Inteface
Reconfiguration and Monitoring Service
Add/edit/remove ports
Error reporting
Configuration Information

Figure 12: Services for local resource scheduling of the on-chip network interface

The LRS is also responsible to resolve contention between messages with different traffic types, so the NoC interface need not care about the priority of the messages or even whether the message originated from a high-critical component or a non-critical one. This control is done by taking care of periods and phases for periodic messages, the minimum interarrival times as well as priorities for sporadic messages and in case of available bandwidth, the transmission of aperiodic messages.

Figure 12 depicts a logical model of the services in the LRS of the on-chip network interface and their relationships. The LRS realizes a set of services such as the core interface using ports, bridging of incoming and outgoing messages and serialization of messages. The interface between the bridging and the serialization of outgoing messages is provided by queues. Each queue is dedicated to a single priority to provide the serialization layer with an efficient access to the messages. More details are provided in section 1.1.5.

The services of NoC interface on the other hand, provide the LRS with an interface to the router by generating the packets and consequently flits and providing the routers with the flits. Inversely, they disassemble the packets and flits received from the NoC and send them to the LRS to be forwarded to the destination ports for incoming messages.

For example in case the NoC is realized based on STNoC, the NoC interfacing services will be realized by the “Shell and Kernel”.

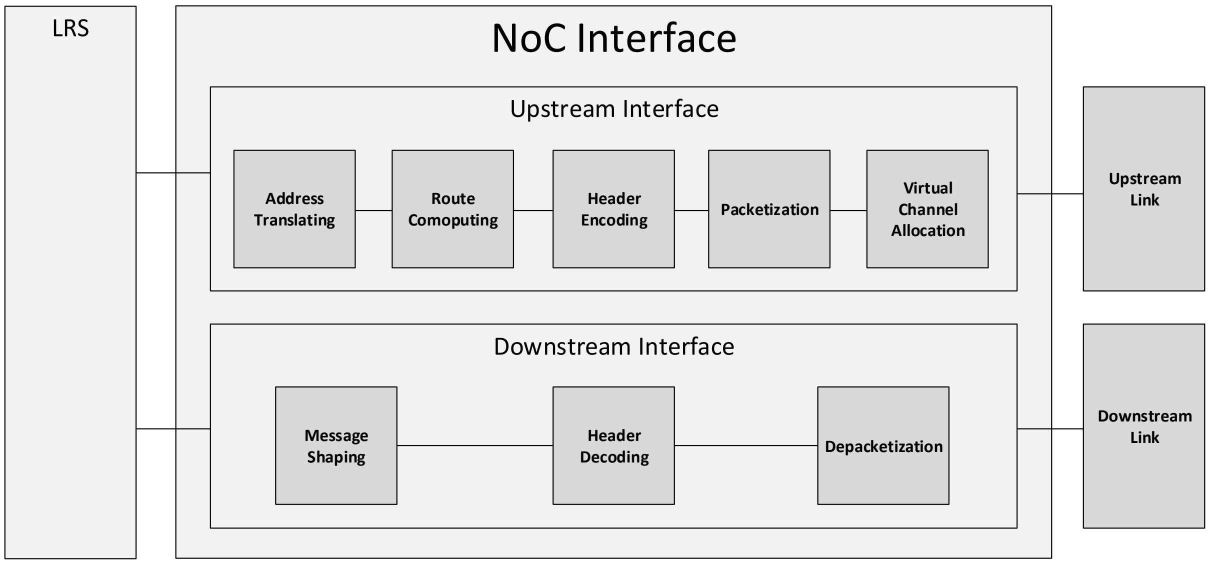


Figure 13: Services of on-chip NoC Interface

1.1.1 Core Interface using Ports

The core interface acts as the interface between the cores within the tile and the NoC by providing input and output ports. Each port is accessible by predefined partitions (established by the executions services) from the core side as well as the bridging layers from the NoC side. In case of an output port, the processor core writes the message into the respective outgoing port and thereafter the bridging services for outgoing messages read the port and disseminate the packets to be delivered to the NoC. In case of an input port, once the message reaches a destination NI, it will be placed at the input port by the bridging layer for incoming services to be read by the respective processing cores. In order to assure the segregation of different criticalities, requests shall be controlled based on the predefined configuration information (e.g., period and phase for PE messages, the MINT for SP messages, authorized partitions, etc.).

Each port composed of two main areas:

- ***Port configuration*** is associated with each port, including the port identification, the virtual link identification, the data direction (i.e., in or out), the traffic type (i.e., periodic, sporadic, aperiodic), the timing parameters depending on the traffic type (i.e., period, phase, minimum interarrival time), the priority and the message size.
- ***The data area*** is a buffer for messages which is either written by the tile in case of output ports or read by cores in case of input ports. The data area is a buffer with update-in-place semantic for periodic ports which is overwritten whenever new data becomes available from the core or the network. For sporadic and aperiodic messages, the buffer is a message queue.

The transmission of periodic messages is scheduled based on the predefined period and phase available in the port configuration. As the periodic messages carrying state values, they should not be queued; therefore the periodic ports employ a buffer for the data area. Since the buffer is accessible by processing cores as well as the bridging layers, a synchronization mechanisms such as double buffer or none-blocking write [HK2011:p.224] shall be employed to avoid inconsistency. In case of sporadic messages, instead of overwriting the data, cores can enqueue the new data and the bridging layer will send the data according to the timing constraints. In case of aperiodic messages, cores can enqueue the messages at any rate, but there will be no guarantees on whether and when the message arrives at the destination. The messages will be queued and in case of available unused bandwidth, the aperiodic message will be delivered to the NoC.

1.1.2 Bridging of Outgoing Messages

This service feeds the messages available at outgoing ports into the serialization layer. Based on the traffic type of each port, different actions will be taken:

- **Time-Triggered Dispatching of Periodic Messages:** The service reads the periodic ports and feeds the data into the dedicated queue for periodic messages in the serialization layer at the defined instant given by the time-triggered schedule.
- **Traffic Shaping of Sporadic Messages:** The service reads the sporadic messages from the ports and enqueues the respective buffers at the serialization layer . The sporadic messages will be read from the port only if the minimum interarrival time is already elapsed. This parameter is available in the port configuration.
- **Relaying of Aperiodic Messages:** Since the aperiodic messages have no timing constraints on successive message instances and no guarantees with respect to the delivery and the incurred delays, the service only forwards them once there is new message available at the respective port. Afterwards, the serialization layer will send the aperiodic messages only if there is bandwidth available which is has not been used by the periodic and sporadic messages.

1.1.3 Conversion from Logical to Physical Names

Components are only aware of logical names, whereas the platform requires physical names for the routing of messages. The conversion between logical and physical names bridges the gap between the application and the communication platform at the NI. This service performs the conversion of names and prepares the data needed for the header according to the information given by the port configuration.

As described in section 1.1.1 and 1.1.2, we pair periodic and sporadic messages with a *Virtual Link ID (VLID)* in order to extract the required address information. VLIDs implicitly define the source port, the destination ports, the message with its semantic content and the traffic type (i.e., periodic or sporadic) and the message timing. Aperiodic messages do not require VLs, but are subject to a connectionless transfer. Therefore, each aperiodic message must include naming information for routing through the network.

The NI is responsible to establish the protocol-specific addresses of the messages. More precisely, when it comes to the end-to-end path of a message, the NI looks up the defined addresses paired to the VLIDs (in port configuration) and generates a protocol-specific NoC address. In case the destination is physically located on the same node, the destination of the target-tile will be generated. Otherwise, the message, will be redirected to the gateway.

The destination address (including the tileID and the portID) of the message will be delivered to the NoC interface, so that the NoC interface generates the path to the destination to be inserted into the header.

1.1.4 Bridging of Incoming Messages

As the NI acts as a bridge between the NoC and the cores, it must be able to support the bidirectional communication. Bridging of incoming messages supports the communication from the NoC towards the cores. This bridging is done via packet classification and dispatching the messages to the destination ports. The one-to-one mapping between the VLs and the ports enables this service to classify the incoming messages and write them to the respective port. Thereafter the respective tile will be able to read the message.

1.1.5 Serialization of Messages

As described earlier, the bridging layer dequeues the messages at the core interface and feeds them into the serialization layer. The bridging layer is intended to apply the temporal constraints defined at each port, without taking into account the priorities; this part will be done by the serialization layer. In other words, the serialization layer consolidates all messages of the same priority and feeds them into the NoC interface, taking into account only the priorities.

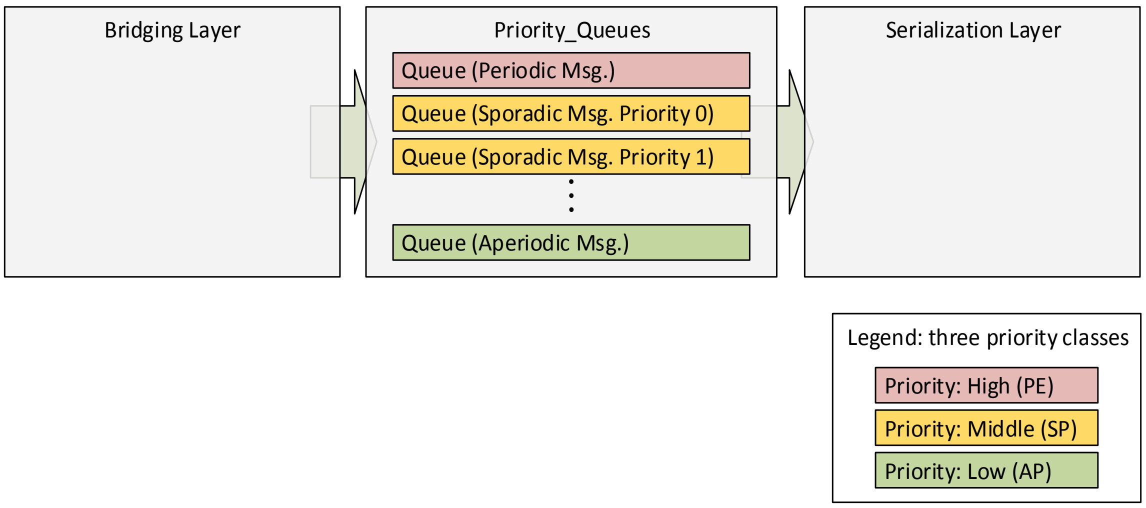


Figure 14: The interface between the bridging layer and the serialization layer at on-chip NI provided by priority-queues

As depicted in Figure 14, the interface between the bridging and serialization layer is provided by priority-queues. Each priority-queue is associated with a unique priority, which determines the order of writing them to the NoC interface. One queue, which has the highest priority, is used for all periodic messages. At any point in the time, there is at most one message in this queue due to the conflict-free time-triggered schedule that prevents contention between periodic messages. Multiple queues of middle priority class can be used for the sporadic messages, where each queue exhibits a corresponding priority level. The messages from the sporadic ports are relayed into the priority-queues matching the priority of the VLID. For aperiodic messages, there is one serialization queue with the lowest priority. The serialization layer reads the available message of the highest priority and feeds it into the NoC interface.

1.1.6 Timely Blocking and Shuffling

In order to assure a collision free communication, we need a mechanism to solve the collision between messages arriving at the NI at the same time. To resolve the collision between the periodic and sporadic messages we can use either *timely block* or *shuffling,* whereas to resolve the collision among sporadic messages and also between sporadic and aperiodic messages, only shuffling is employed.

- The ***timely block*** mechanism guarantees no collision between two messages of different priorities by blocking the bandwidth during a guarding window for a message of higher priority.
- For the ***shuffling*** mechanism, no guarding window is needed. In such a mechanism, a periodic message could arrive at the NI at the instant at which a sporadic message is traversing and therefore the periodic message has to wait until the ongoing sporadic message traverses the link. In the worst-case, the periodic message will be delayed for the transmission duration of a maximum size message.

This means each sporadic message or aperiodic message will be delayed for duration of a single message of maximum size.

1.1.7 Monitoring Services

The *Reconfiguration and Monitoring Services* support the online reconfiguration and monitoring of the NI, which is performed locally by the LRM or globally by the GRM. The reconfiguration and monitoring services can be employed to reconfigure the usage of the available resources based on the current status of the system. In addition to that these services can reflect the environmental changes into the system in order to either enhance the efficiency of the overall system or to switch the operation mode. Moreover the monitoring and reconfiguration services can be employed for fault recovery purposes. In all of above mentioned applications, the process of reconfiguration will be triggered by the monitoring services.

For example in case of fault recovery, the process of reconfiguration is triggered by the monitoring services, once a fault is detected. The fault can be for instance, the temporal violation of periodic messages at one core. Once this violation is detected monitoring services will report this fault to the LRM and the LRM will choose the new configuration either by its own or possibly report it to the GRM (see Figure 15).

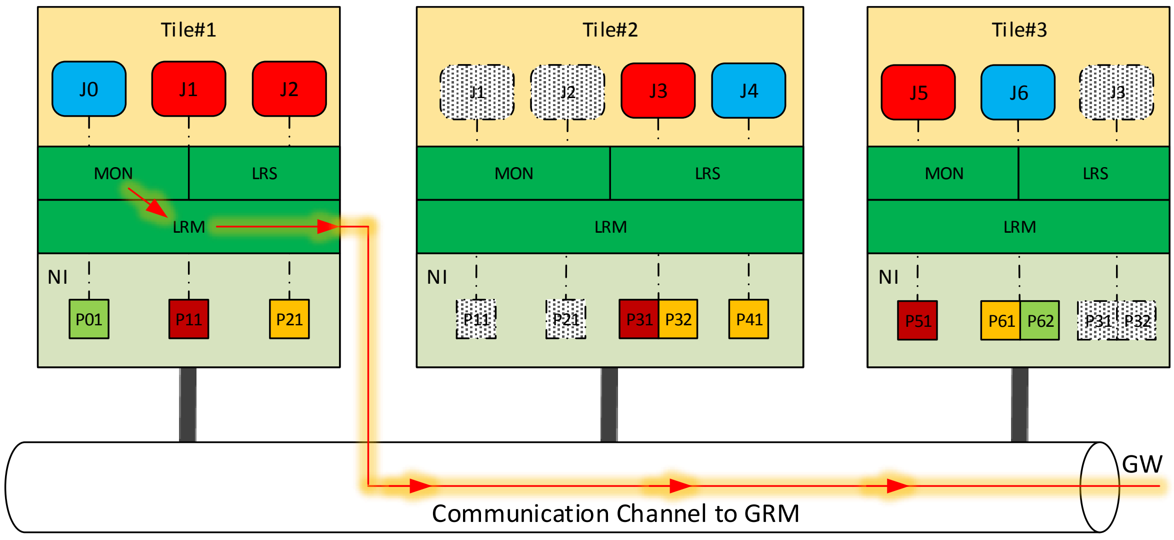


Figure 15 The LRM reports the monitored fault by the MON to the GRM

1.1.8 Reconfiguration Services

Reconfiguration services support the reconfiguration and rescheduling of the resource allocations defined by the LRM (either decided by its own or obtained from the GRM) by updating the listed parameters of the configuration information or port configuration.

- Port configuration:

  - Enabling or disabling of individual ports: individual ports can be switched off or on by the reconfiguration service. (see shaded ports in Figure 16)
  - Temporal parameters: the temporal parameters such as periods and phases for periodic messages, minimum interarrival times for sporadic messages can be updated
  - Priorities: the priority of the port is bound with the priority of VL and the message, thus changing the priority of the port will affect all of them
  - Buffer size: changing the size of buffer leads to the change of the length of queues for the messages
- Configuration information at the NI:
  - Logical physical address-mapping: in case one core is disabled or moved to another tile, the other NIs need to be aware of this modification to dispatch the respective message correctly
  - Timely block or shuffling: whether timely block or shuffling will be employed to resolve the collision between PE and SP messages

Following the example described in section 1.1.7, assume the LRM decides to shut down the erroneous tile and transfer its tasks into tile#2. According to the new configuration, the erroneous core (running J0) will be disabled, Tile#2 will take over tasks J1 and J2 and disable J3 and J4. Tile#3 enables J3. In addition to the tasks, the new configuration covers the reconfiguration of respective ports (see Figure 16). Notice that the lower priority tasks J0 and J4, together with their associated ports, are dropped in the new configuration.

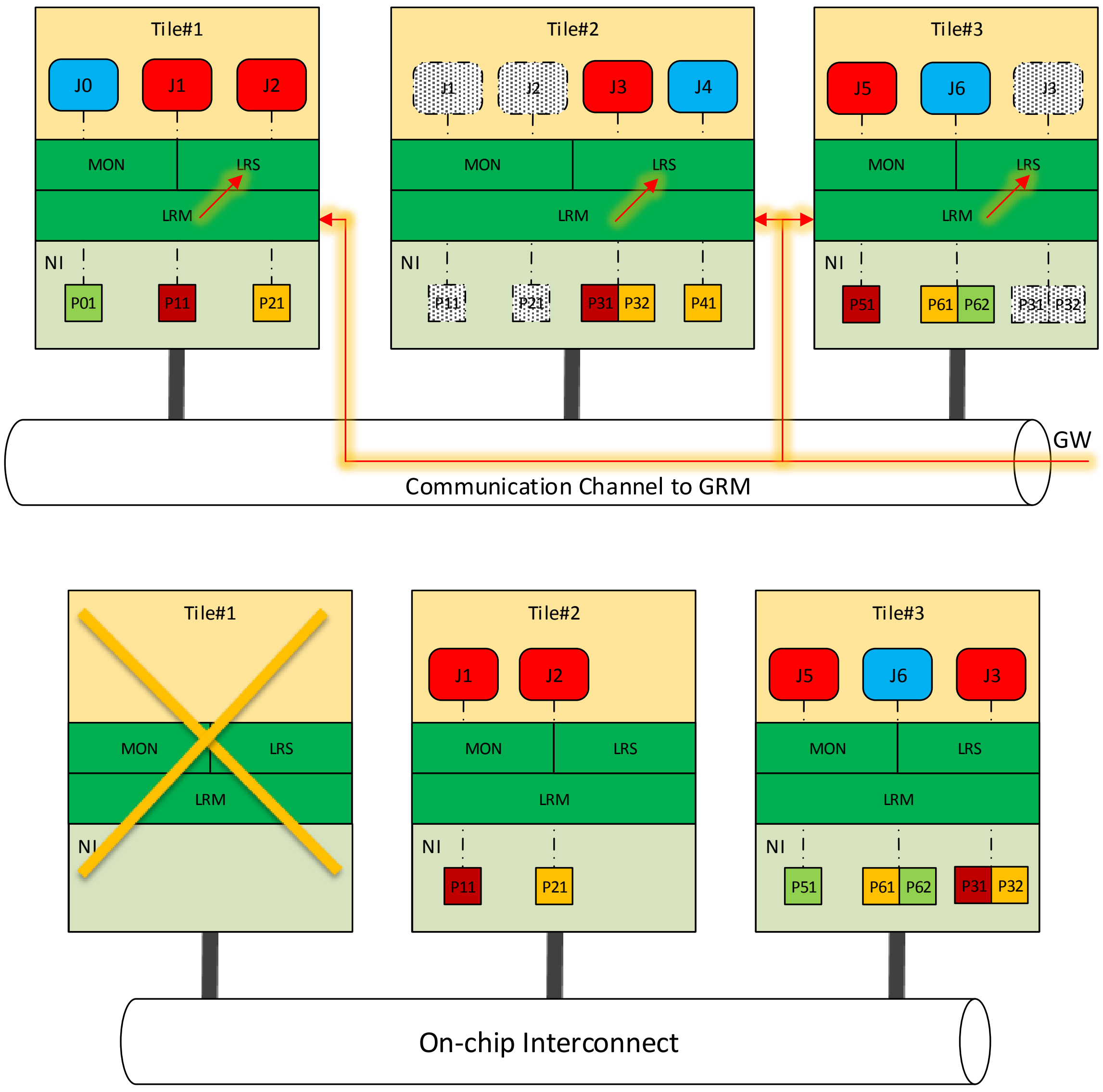


Figure 16: a) The new configuration chosen by the GRM, b) The new configuration applied by the LRMs

1.1.9 Address translation and route computation

As the DREAMS on-chip network is source-based controlled, the on-chip path is computed at the NI and the NoC interface inserts the path into the head flit before the packet leaves the NI. However the NI provides only the on-chip address and in case the message is targeted to another chip, the message will be sent to the gateway. Supporting source-based path computation simplifies the architecture of routers, as they do not need to compute the path.

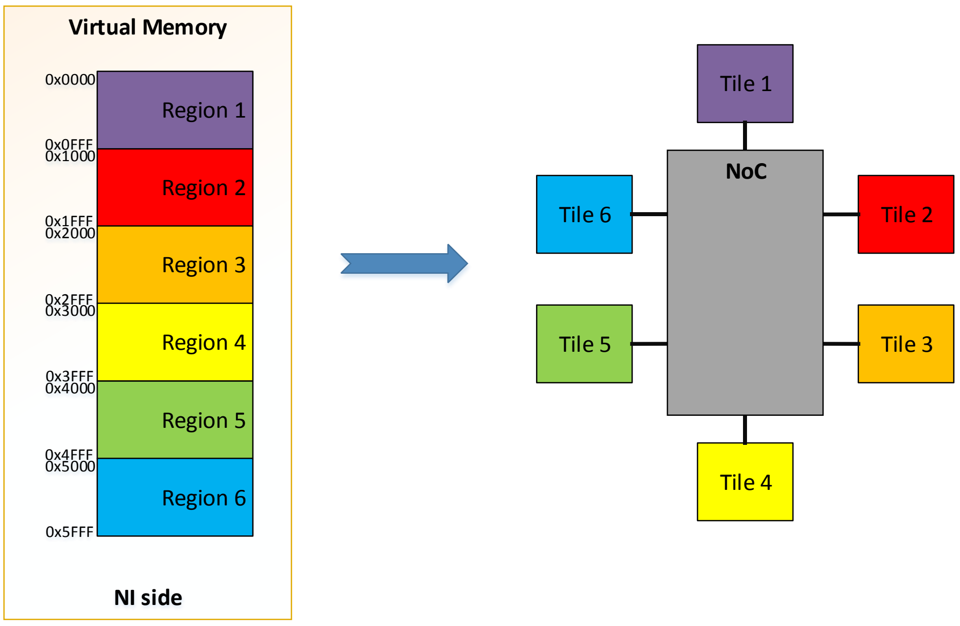


Figure 17: Tiles are seen as memory regions at NoC interface

As described in Figure 2 in section 1.1.2 of Part I of the document, different parts of the system can use different addressing types, i.e., a protocol-specific addressing can be used for the NoC. For example, in case of the Spidergon STNoC on-chip tiles are seen as memory regions from the point of view of the NoC interface. This means that each tile is mapped to a specific address region of a virtual address space and will be considered as a region of the address space with the “prefix” of the address defining the on-chip target. In other words, instead of using a destination address identifying the tile, the NI needs to give addresses in the region that are mapped to the destination tile. Figure 17 depicts this correspondence between the memory addresses and the physical addresses.

1.1.10 Header and packets handling services

As shown in Figure 13, the NoC interface is mainly composed of the upstream (which is towards the NoC) and the downstream interface (which is towards the processing core). The address translating service will translate the protocol-specific addressing. After the address has been translated, the route computing service will compute or look up the on-chip path to the destination tile. These two services will be employed in case the NoC uses its own protocol-specific addressing. Assembling the headers and packets is supported by a set of services in the upstream interface, which encode the headers and packetize the messages according to the results of prior services (address translating and route computation) to generate the flits. In downstream interface there are services for depacketization of the flits and decoding the headers.

1.1.11 Virtual channel allocation

In order to achieve segregation of mixed-criticality traffic and avoid contention, we employ *Virtual Channels*. Virtual channels (VC) employ the concept of virtualization and provide several channels out of a single physical link by using multiple buffers at both terminals of each physical link.

Utilizing the VCs in conjunction with the priorities delivers us the possibility to support different criticalities and guarantee bounded delay for high-critical messages. As shown in Figure 18, different buffers (which represent virtual channels) can be allocated to the priorities. For instance VC1 can be allocated for periodic messages, VC2, VC3 and VC4 to sporadic messages and VC5 to aperiodic messages. Pairing the VCs with the priorities at resources (for example at router) helps the virtual channel allocator to allocate the VCs based on the priorities. Arbitration between two VCs of the same priority (VC2 and VC3 in this example) is performed based on the round-robin scheme.

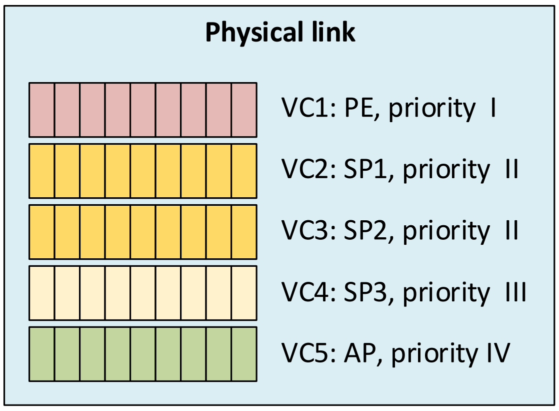


Figure 18: VCs in conjunction with priorities support mixed-criticality

The allocation of VCs is packet-based and after the packet is constructed by the prior services in the NoC interface, the virtual channel allocation service will allocate an available VC to the packet. After the VC has been allocated to a packet, the process of injection of the flits will start. This process will be controlled by the *credit-based* flow control. With credit-based flow control, the NoC interface keeps a count of the number of free flit buffers in each virtual channel at the next router. Then, each time the NoC interface forwards a flit, thus consuming a buffer at the router, it decrements the appropriate count. If the count reaches zero, all of the buffers are full and no further flits can be forwarded until a buffer becomes available. Once the router forwards a flit and frees the associated buffer, it sends a *credit* to the NoC interface, causing a buffer count to be incremented.

1.1.12 Message shaping

The message shaping block is placed after the depacketization and header decoding blocks in the downstream interface. This service extracts the data (e.g., the payload, the destination port) from the head flits and the body flits and constitutes the message to be forwarded to the bridging of incoming message.

1.1.13 Intratile routing of messages at NI

The NI provides the cores and the partitions the communication services, by which components in the partitions can communicate by messages. The destination of messages can be on the same tile, on another tile or on another node. In latter case the message will be forwarded to the on-chip/off-chip gateway to be forwarded to the destination node.

In case the destination of messages generated by the components resides in the same tile, there is no need to pass the message through the NoC and the message can be redirected to the destination based on the configuration information right at the NI. This redirection service is performed by the dedicated loop-back interface at the NI and is synchronized by the bridging layer. This interface redirects the messages to the respective port and in case of scenario III (see Figure 20) delivers it to the bridging service to be sent via the NoC.

The intratile routing of messages can occur in three different cases as follow:

1- As shown in Figure 19, two cores which reside on the same tile communicate with each other. In this case, the NI forwards the message directly at the core interface (see Figure 11).

2- In Figure 20, a message originated from the outside of the tile is destined to two different cores, both of them residing on the same tile. In this case the message will be duplicated at the NI and sent to both cores at the same time.

3- In the third scenario described in Figure 21, one core is sending a message to a core residing on the same tile as well as a destination outside of the tile. In this case, the NI duplicates the message and sends a copy to the core on the same tile.

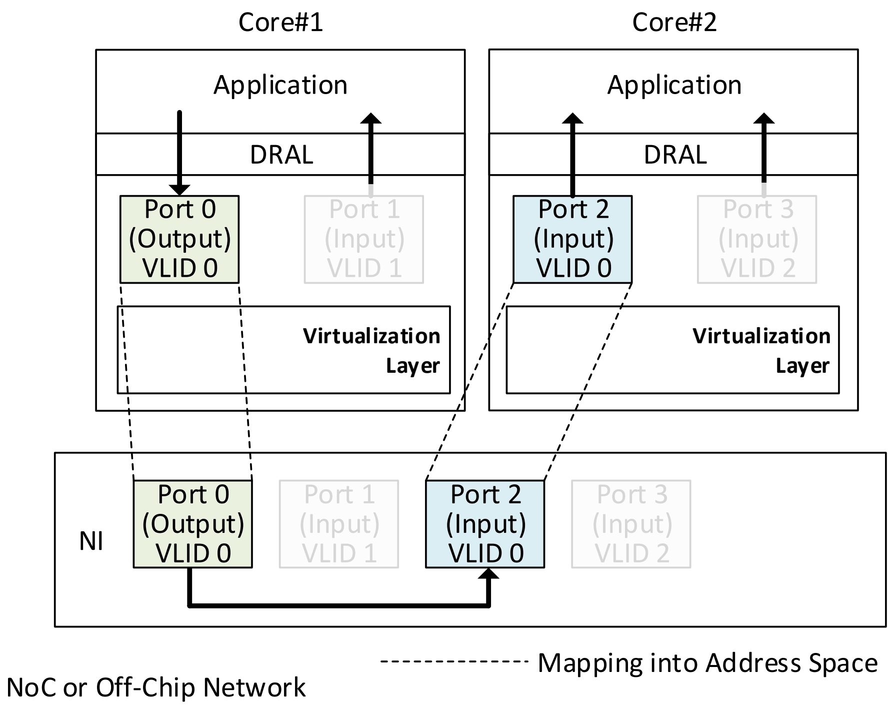


Figure 19: Scenario I: Core 1 talks to Core 2, which resides in the same tile

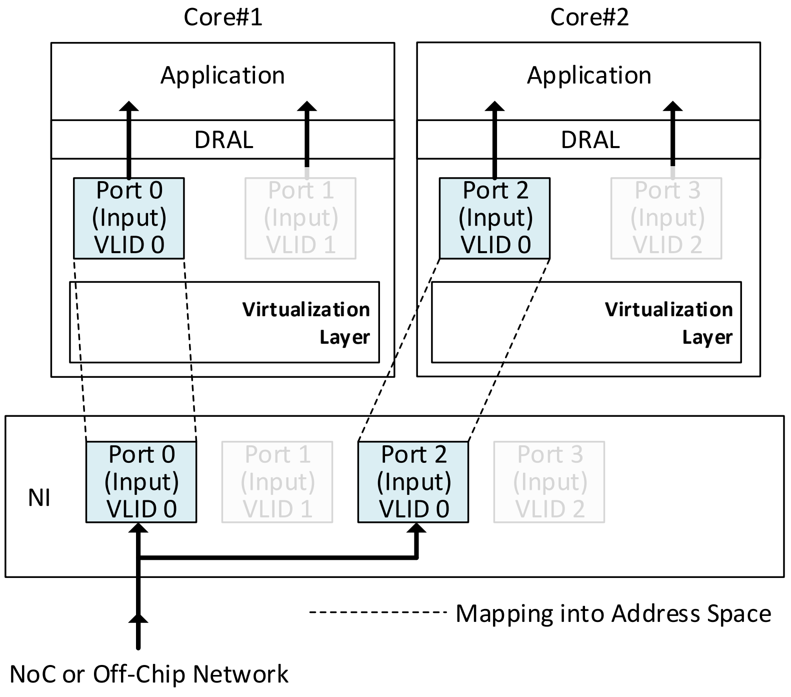


Figure 20 Scenario II: Incoming message to the tile is targeted to Core 1 and Core 2, both on the same tile.

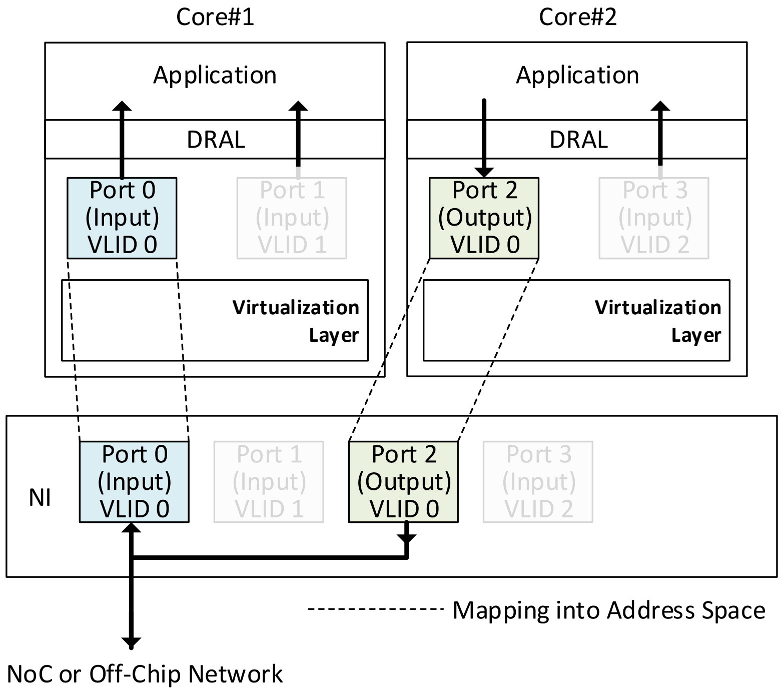


Figure 21 Scenario III: Originated message from Core 2 destined to Core 1 (on the same tile) and also to a core on the other tile

1.2 On-Chip Communication Router

On-chip routers realize the cross connection between network interfaces by building the network on-chip in combination with the physical links. The routers relay the body flits according to the configuration obtained from the head flit. This configuration, whose lifecycle terminates once the tail flit traverses the switch, is defined per VC (as described in section 1.1) and stored in the respective state fields shown in Figure 21.

Each router is composed of input and output units, configuration information, switch and control logics which collectively implement the flow control functions required to buffer and forward flits to their destinations. We will examine the services a typical on-chip router provides with regards to prioritization and segregation of virtual channels.

Figure 22 depicts the services of the router which realize the on-chip communication. On the left hand side, the interface for incoming flits serves the NI or another router by providing the interface for incoming flits, whereas the interface for outgoing flits provides the outgoing interface for the NI or the next router. The switching and VC allocator handle the flits according to the available credits and the temporal condition of switch. The detailed description will be given in the following sections.

1.2.1 Interface for incoming flits

Input units act as the interface for the routers and the NIs. Each input unit includes multiple buffers, each of which represents one VC and consequently a priority. Each VC is characterized by "VC State Fields", which include the current status of the VC (e.g., the number of available credits, the bound output VC, start and end address of buffer). Each unit is connected at one end to a physical link and at the other end to the switch. The number of interfaces depends on the architecture and the topology. For example, in case of a router with north, west, south and east directions, there would be four input units and four output units.

1.2.2 Virtual Channel Allocation

As described in section 1.2, the resource allocation at physical layer is done per packet as well as per flit. For instance each VC will be allocated to one packet at each time. This allocation is done by the *Virtual Channel Allocator (VA)* at the instant the head flit of the packet arrives at the router under the condition that there is any available VC at the output unit of the router. The VC will be deallocated once the tail flit traverses the switch. The availability of the VC is controlled by the credits.

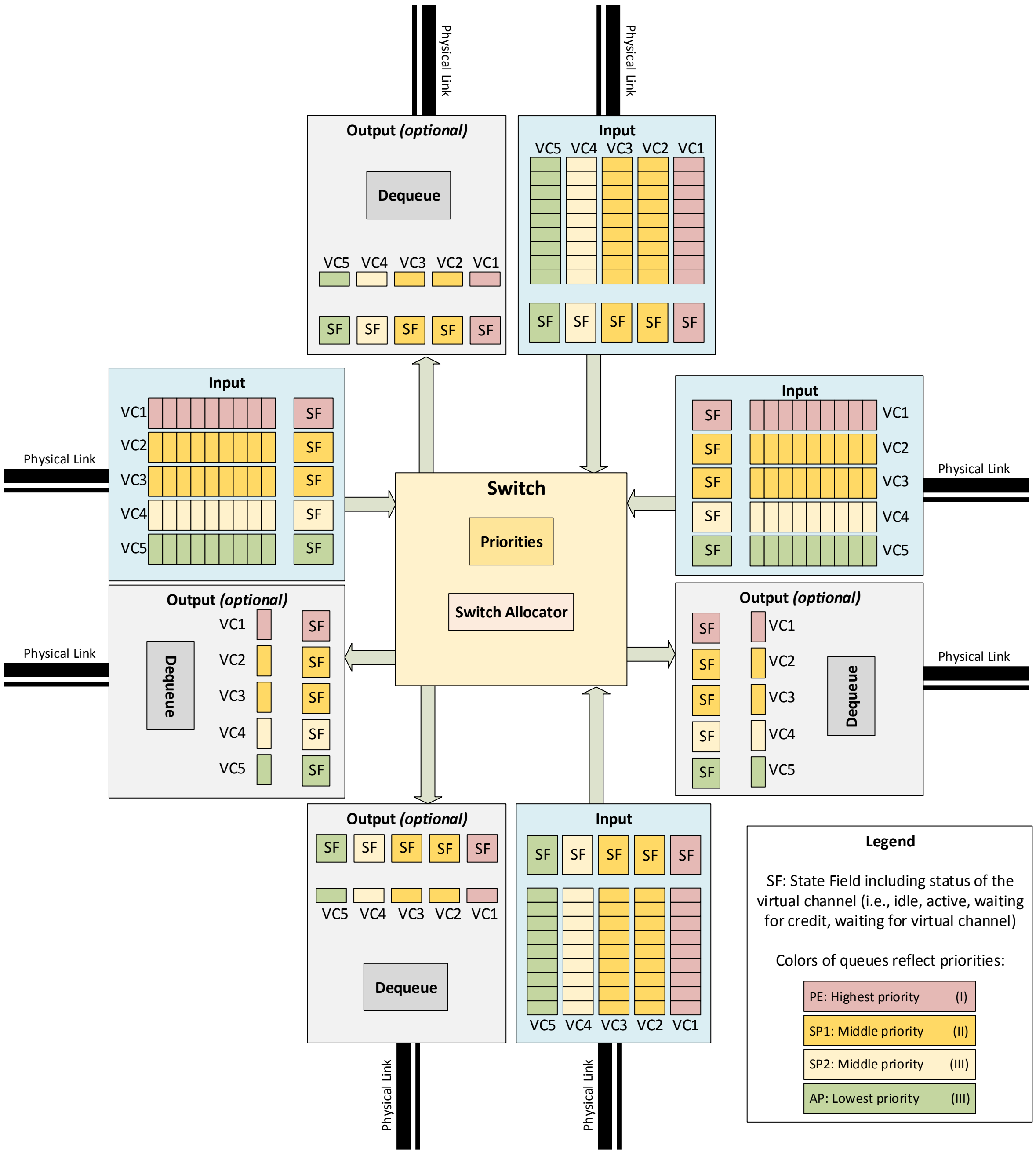

Physical Link
Output (optional)
Dequeue
VC5 VC4 VC3 VC2 VC1
SF
Input
Switch
Priorities
Switch Allocator
Legend
SF: State Field including status of the virtual channel (i.e., idle, active, waiting for credit, waiting for virtual channel)
Colors of queues reflect priorities:
PE: Highest priority (I)
SP1: Middle priority (II)
SP2: Middle priority (III)
AP: Lowest priority (III)

Figure 22: The on-chip router

1.2.3 Switching

As shown in Figure 22, the switch is the central part of the router, which is fully configured by the *Switch Allocator (SA)*. After the virtual channel was allocated to the packet by the request issued by the head flit, each flit of the packet needs to traverse the switch. The allocation of switch is per flit which means each flit needs to request a time slot of the switch from the SA. The SA will schedule the switch among the competing flits (in other words competing virtual channels), taking into account two criteria. First, in case multiple requests come to the SA at the same instant, the flit belonging to the packet of higher priority will win the competition. The second point which needs to be checked is the availability of a vacant buffer at the output of the switch. In case two or more requests of the same priority arrive the SA at the same time, the SA will allocate the switch based on the round robin.

After the flit won the competition for the switch allocation, it can traverse the switch and be stored either directly into the input buffer of the adjacent router or optionally in the single buffer at the output unit.

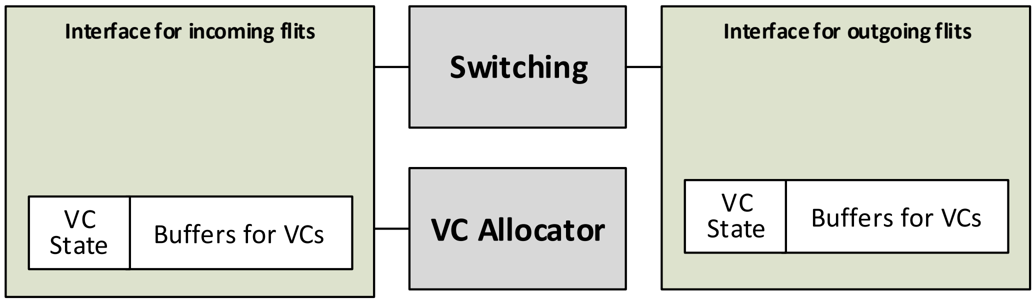


Figure 23: Services of on-chip router

1.2.4 Interface for outgoing flits

The interface for outgoing flits serves as an intermediary place for the flits, which have traversed the switch and waiting for the next router or the NI, to be dequeued. In case the switch has an output speedup of one (switch bandwidth equals output bandwidth) the switch and the output channel can be synchronized, otherwise, the output unit typically incorporates a queue (as shown in Figure 21) to decouple the switch from the output unit [14].

1.2.5 Monitoring service

Monitoring services require the availability of registers that the OS can access to understand the current status of the traffic. This solution is not possible at the on-chip router level due to a high cost of implementation (it might be tens to hundreds of routers in a NoC) but also due to the necessity of a distributed bus all over the SoC to access these registers.

Instead traffic monitoring registers are available at network boundaries, i.e. Network interfaces, and the OS can get information here on the status of the traffic. Being aware of the routing paths and network topology, it will be able to react.

1.2.6 Configuration service

Unlike Wide Area Network routers, On-Chip networks must be as low-cost area as possible. While providing full switching and arbitration policies to forward incoming packets to output ports, their implementation must be reduced to the strict necessary. That’s why there are no possibilities to reprogram on-chip routers.

However, it does not mean that the routing itself cannot be reprogrammed. Indeed, as well as for the monitoring, the reprogramming function is moved to the network boundary, i.e. the Network Interface. Using source routing, NoC reprogram the routing in the network interface routing registers, and this routing information is then embedded in the NoC packet header part. The routers will react differently to a new route indicated in the header of the encapsulated packet.

In the same way, reprogramming the QoS for a packet consists in reprogramming the QoS settings in the NI registers and then the QoS information is part of the header. The router will react to this QoS information.

Group of Off-Chip Communication Services

In the following the message-based off-chip communication services are described including the off-chip network interface and the off-chip routers. The off-chip network interface provides the interface of a node to an off-chip network with a suitable communication protocol (e.g., TTEthernet). The connection between network interfaces occurs using one or more off-chip routers in a given topology (e.g., star, ring).

1.3 Off-Chip Communication Network Interface

The off-chip communication network interface is a building block to realize the gateways between the network-on-chip and the off-chip networks. In addition, the off-chip communication network interface can be used in DREAMS nodes that do not contain network-on-chip (e.g., GALILEO interface to DREAMS).

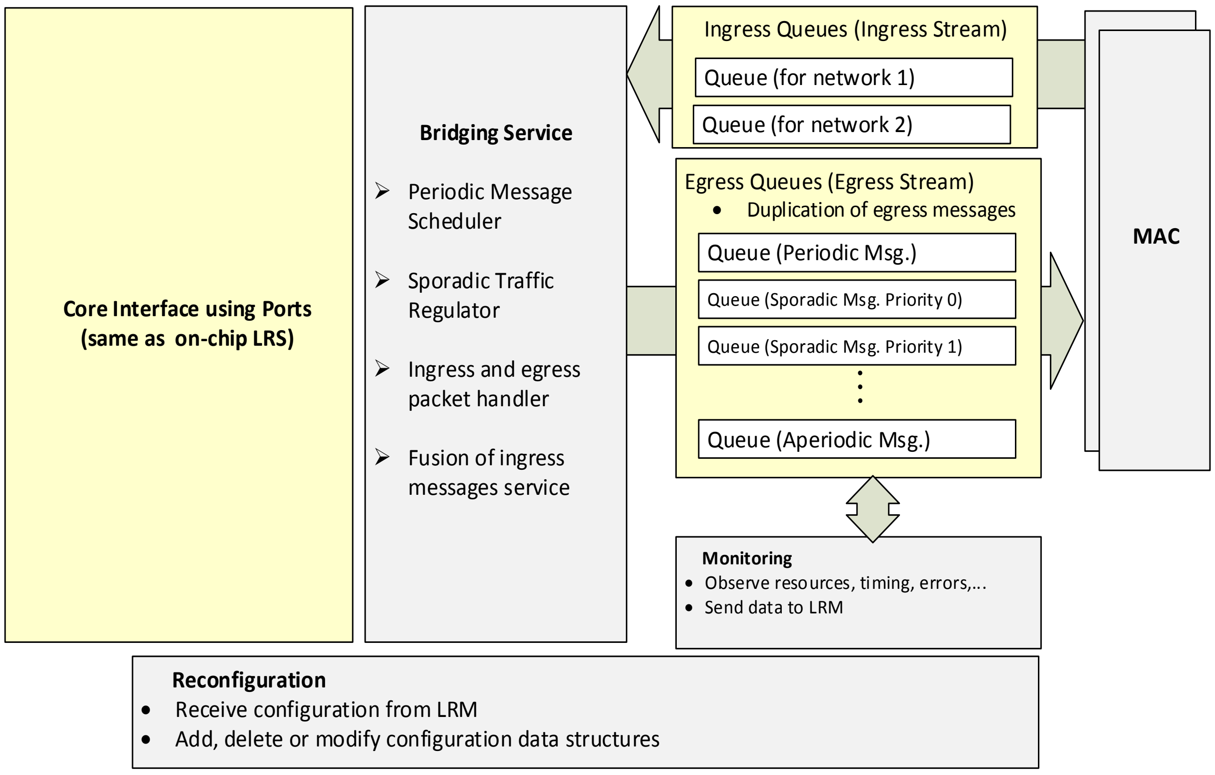

Core Interface using Ports (same as on-chip LRS)
Bridging Service
Periodic Message Scheduler
Sporadic Traffic Regulator
Ingress and egress packet handler
Fusion of ingress messages service
Ingress Queues (Ingress Stream)
Queue (for network 1)
Queue (for network 2)
Egress Queues (Egress Stream)
Duplication of egress messages
Queue (Periodic Msg.)
Queue (Sporadic Msg. Priority 0)
Queue (Sporadic Msg. Priority 1)
Queue (Aperiodic Msg.)
MAC
Monitoring
Observe resources, timing, errors,...
Send data to LRM
Reconfiguration
Receive configuration from LRM
Add, delete or modify configuration data structures

Figure 24: Off-Chip Network Interface

The off-chip network interface acts as the injection point for messages generated by a node for the off-chip network. Likewise, the network interface is a sink for messages from an off-chip network destined to the respective node.

Figure 24 shows a model of the network interface, which consists of a set of ports, a bridging service, egress queues, an ingress queue and a MAC.

1.3.1 Egress queuing service

The egress queues consist of one periodic egress queue, multiple sporadic queues and one aperiodic egress queue. Each sporadic queue has its own priority level.

The deterministic behaviour of the periodic messages is ensured by the "periodic message scheduler" (see section 1.3.4) in combination with the higher priority than sporadic messages. The deterministic behaviour guarantees that no conflict appears at the egress queue. Therefore one queue is sufficient, which needs to provide buffer capacity for a single periodic message of maximum size.

To control the resolving of contention between the sporadic messages, we distinguish multiple queues according to their priorities. These queues are used to multiplex the frame flow that comes from the internal message queues. The queues provide guaranteed buffer capacities, which can also be realized by dynamic memory allocation. The guaranteed buffer capacities allow to prevent message loss due to the bounded accumulation of sporadic messages determined by the rate-constraints.

1.3.2 Ingress queuing service

The ingress queue consists of one FIFO queue for each network. The incoming massages to the MAC from the network are queued into *the ingress queue,* then the *ingress queuing service* notifies the *message bridging service*.

1.3.3 Core interface service

This service allows the core to read and write to the ports in analogy to the on-chip network interface (cf. part II, section 1.1). The core interface is independent of whether the interaction between components occurs via an off-chip or an on-chip network.

1.3.4 Periodic message scheduler

The periodic message scheduler is responsible for forwarding the periodic messages from a corresponding virtual-link to the egress queue at the time specified in the static communication schedule.
The periodic message schedule uses the port configuration parameter to determine the point in time when the periodic message needs to be forwarded with respect to the global time base.

1.3.5 Sporadic traffic regulator

The sporadic traffic regulator guarantees the minimum interarrival time between two consecutive instances of sporadic messages on the respective virtual link. If this timing constraint is satisfied, then the sporadic traffic regulator relays these sporadic messages from its queue to one of the sporadic queues at the egress queue according to the message priority.

1.3.6 Ingress and egress packet handler

The packet handler is responsible for redirecting the incoming aperiodic messages from the off-chip network to the respective ports. In addition, the packet handler polls the aperiodic ports and redirects the respective messages to lowest priority egress queue.

1.3.7 Fusion of ingress messages

This service performs message deduplication using different mechanisms according to the traffic type.

- ***Periodic message***: In order to hide the paths and different latencies of the different networks, the fusion of ingress messages service requires a priori knowledge about the time-triggered schedule. This schedule includes information about the receiving time, the sending time and the corresponding buffer identification. The fusion of ingress messages service checks the corresponding virtual-link buffer before the sending time and takes the decision to send one of the redundant periodic messages accordingly. Moreover, the fusion of ingress messages service establishes deterministic arrival times of these messages.
- ***Sporadic message:*** For each incoming sporadic message the fusion of ingress messages service checks the sequence number and compares it with the sequence number that is listed in the configuration parameters. The "First Valid Wins" policy is used to take the decision on the forwarding of redundant messages. Upon the transmission of a message, the

fusion of ingress messages service updates the sequence number in the configuration parameters.

- ***Aperiodic message:*** NO redundant message support for the aperiodic message.

1.3.8 Duplication of egress messages

This service is responsible for creating copies of sporadic and periodic messages at the egress ports that are sent to the different MACs.

1.3.9 Reconfiguration and monitoring services

The off-chip network interface has two building blocks (i.e., reconfiguration and monitoring) that are responsible for rewriting the configuration parameters and for the observation of the communication resources, the message timing and for retrieving error detection information.

The monitoring block will monitor the time of the message arrival and transmission and compare it with its configuration parameters (i.e., period and phase of sporadic messages with tolerance windows, minimum interarrival times of sporadic messages). In addition, the monitor is responsible for monitoring the application behavior (e.g., monitoring deadlines, overload detection).

The LRM can check and analyse the monitored behavior and send a new configuration to the reconfiguration building block. The reconfiguration building block will interpret the messages from the GRM and adopt the modified configuration parameters of the ports. The following configuration parameters are supported:

- ***Timing configuration of ports***: This configuration parameters include the period, phase and tolerance windows of periodic messages, as well as the interarrival times and priorities of sporadic messages.
- ***Address information of ports***: The virtual link associated with a port can be changed.
- ***Change of guaranteed buffer capacities***: The queue size associated with ports, ingress and egress queues can be modified.
- ***Replication and fusion***: The redundancy degree of messages and the time for replication and fusion and be configured.
- ***Memory map of tile interface***: The address of the ports in the memory map of the tile interface can be changed.

1.3.10 MAC interfacing

The MAC interfacings sends and receives the message from the off-chip network by encapsulating the message in the frame or decapsulating the message from the frame. In case of incoming messages, the MAC layer filters messages that are not destined to this node based on the MAC address.

1.4 Off-Chip Communication Router

The model of the off-chip router is illustrated in Figure 25. The off-chip router architecture includes several building blocks to segregate messages from subsystems of different criticality, to ensure the deterministic behaviour of the periodic messages and the bounded end-to-end delay of sporadic messages.

The off-chip router provides multiple physical links and a bridge layer. Each physical link contains a physical layer and a MAC layer. The bridge layer is responsible for handling ingress messages and forwarding them to the egress ports depending on the traffic type (periodic, sporadic and aperiodic).

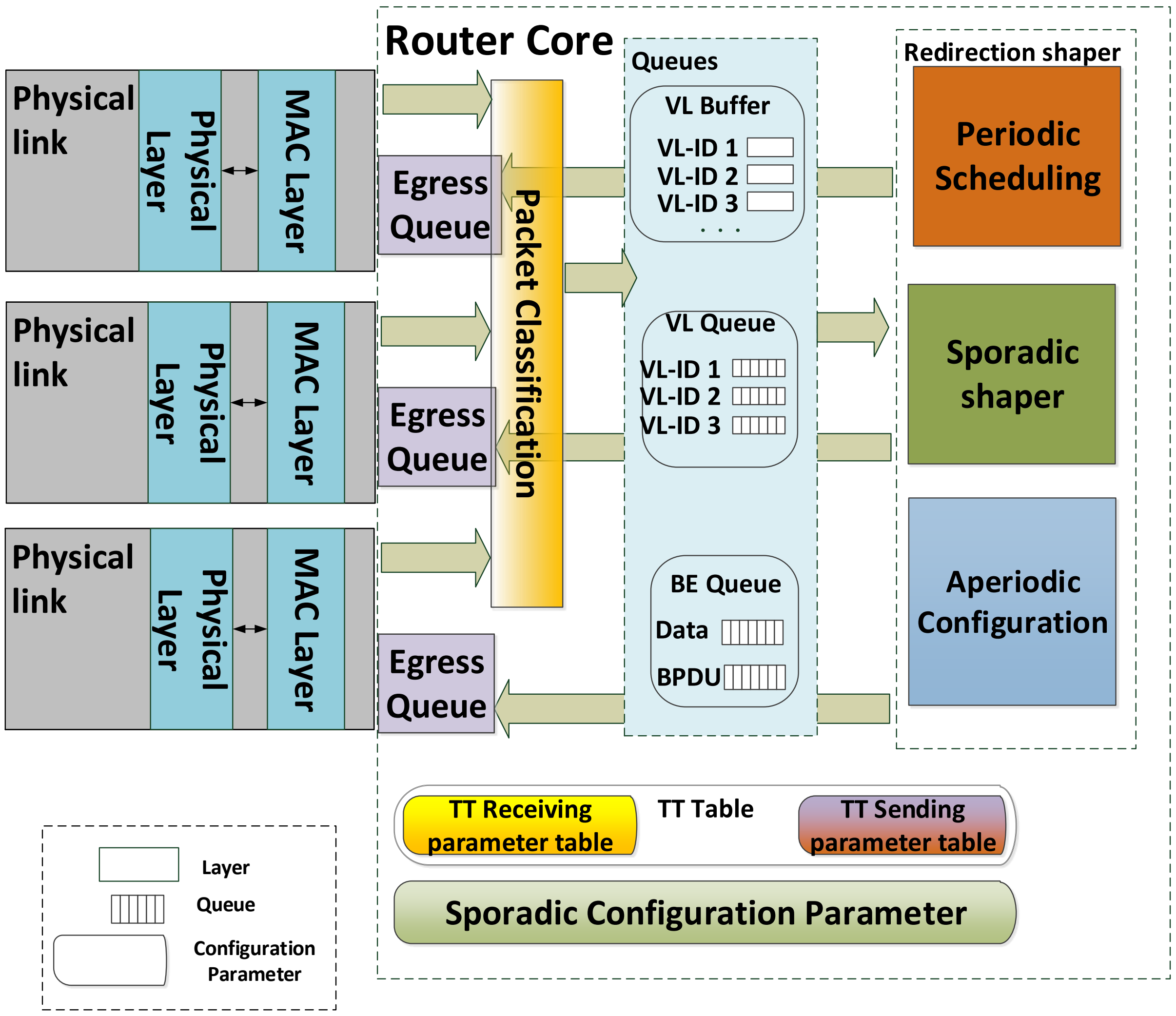


Figure 25: Block Diagram of Off-Chip Router

The router-core provides several services to redirect the messages and guarantee spatial and temporal partition of the critical traffic.

1.4.1 Internal message queuing

The internal message queues belong to three groups according to the message traffic type, as follow:

- ***Periodic VL Buffer***: Each periodic VL has one periodic VL buffer which provides buffer space for exactly one message. In case this buffer is full and another message arrives with the same VLID, the newer message replaces the old one.
- ***Sporadic VL queue***: Each sporadic VL has one queue. It is possible to store several messages of the respective VL in this queue.
- **Aperiodic Queue:** All aperiodic messages are stored in one queue since aperiodic messages have no timing constraints on successive message instances and no guarantees.

1.4.2 Egress queuing service

The egress queuing service is the same as for the off-chip network interface (cf. part II, section 1.3.1).

1.4.3 Packet classification service

The packet classification service distinguishes between traffic types based on connection-oriented and connectionless communication. The connection-oriented communication is used for the periodic and sporadic messages. Aperiodic messages use the connectionless communication. We regard a message as a tuple with the following elements:

- Message in connection-oriented communication : <type, VLID ,data>
- Message in connectionless communication: <type, destination address, data>.

For example, these traffic types can be realized in TTEthernet as follows. The packet classification service distinguishes between traffic types based on the destination address [7]. The destination address field is interpreted differently depending on the traffic type. In aperiodic traffic, the format for destination addresses consist of the mac address of the destination DREAMS chip. However, the destination address of the periodic and sporadic traffic is subdivided into a constant 32-bit field and a 16-bit field called the virtual link identifier (VL-ID). The constant field is extracted from the destination address using the bit mask 0xffffffff0000. In case the constant field has a predefined value, this message is either periodic or sporadic. Otherwise the message is considered as best-effort traffic. The bridge classification distinguishes between periodic or sporadic messages using the value of the VL-ID.

When a periodic message arrives at the router-core from the MAC layer, the packet classification service checks the integrity and validity of the message. The integrity checking verifies that the message has the correct size and arrives from the correct ingress physical link as defined by a time triggered (TT) table (cf. part II, section1.4.7) for the virtual link of the message. Valid messages are put into the corresponding virtual-link buffer, which provides buffer space for exactly one frame. In case this buffer is full and another message arrives with the same virtual-link identifier, the newer frame replaces the old one.

When a sporadic message arrives at the router-core, the message is checked in the filtering unit of the packet classification service. The size of the message must be below the maximum frame size and the ingress physical links must comply with the configuration parameters of the virtual link. Valid messages are enqueued into the corresponding virtual-link queue.

1.4.4 Periodic scheduling service

The periodic scheduling service is responsible for relaying the periodic message from the virtual-link buffer to the queue for periodic messages at the correct egress port according to a TT table. The TT table also determines the point in time when the periodic message is relayed, thereby ensuring the deterministic communication behaviour.

1.4.5 Sporadic shaper service

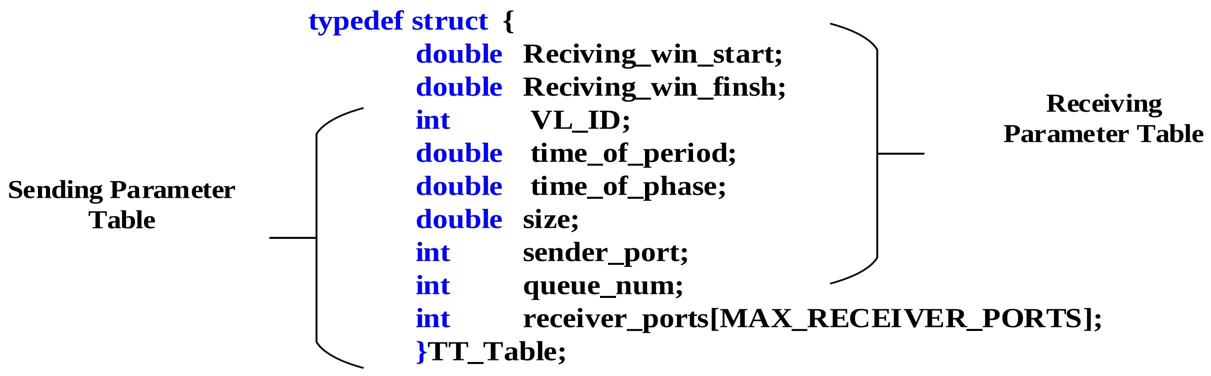
The sporadic shaper realizes the traffic policy for the sporadic messages by implementing an algorithm known as token bucket [8]. This service checks the time interval between consecutive frames on the same virtual link and moves sporadic messages from the virtual-link queue to one of the sporadic egress queues according to the message priority.

1.4.6 Aperiodic self-configuration service

For aperiodic message the spanning tree protocol is used to establish a loop-free topology for communication of aperiodic messages [9]. The supported aperiodic messages include Bridge Protocol Data Units (BPDU) and aperiodic data messages. BPDU messages are exchanged between off-chip routers to determine the network topology, e.g., after a topology change has been observed.

1.4.7 Configuration parameters and reconfiguration service

The time-triggered communication is based on a predefined schedule where there are two groups of parameters for each periodic message: a time-triggered receiving parameter table and a time-triggered sending parameter table providing the message period and phase with respect to a global time base (see Figure 26).



```
typedef struct {
        double   Reciving_win_start;
        double   Reciving_win_finsh;
        int       VL_ID;
        double    time_of_period;
        double    time_of_phase;
        double   size;
        int      sender_port;
        int      queue_num;
        int      receiver_ports[MAX_RECEIVER_PORTS];
        }TT_Table;
```

Figure 26: Time-Triggered Schedule

The sporadic communication is based on configuration parameters that define a minimum interarrival time and jitter for each virtual link. The minimum interarrival time is defined as the time interval between two consecutive messages that are transmitted on the same virtual link. The jitter is the maximum timing variability that can be introduced by multiplexing the virtual links into shared egress queues. A message that arrives within the jitter is considered as timely, otherwise a new minimum interarrival time is started. The structure of the configuration parameters is shown in Figure 27.

```
typedef struct {

                double          BAG;
                double           max_jitter; % jitter value
                double          max_size;   % Maximum message size
                int             sender_port;
                int             receiver_ports[MAX_RECEIVER_PORTS];
                int             priority;
                int             queue_num;
                int             VL-ID;
                int             SN; % sequence number
                Boolean         jitter;
} RC_msg;
```

Figure 27: Sporadic Configuration Parameters

The global resource management can switch time-triggered tables in case the system has multiple scenarios of the periodic messages. In addition, the global resource management can rewrite or

modify the configuration for one or several virtual links for the periodic and sporadic communication.
Moreover, the global resource management may modify the non-active time-triggered tables and later switch to this new table instead of the current one.

1.4.8 Monitoring service

The off-chip network offers a number of monitoring features that can provide the basis for reconfiguration decisions. These monitoring features cover the behaviour of switches themselves as well as the communication that is transferred by the switch.

Switch level monitoring: For each off-chip network switch, at least the following is monitored by the off-chip monitoring service.

- Invalid Switch configuration: Collects a number of flags (config not valid, wrong device ID, CRC error, etc…) that relate to the configuration that is loaded into the switch. If these are erroneous, typically the switch cannot operate in running mode and action is required.
- Not enough Switch memory: i.e. critical traffic dropped due to lack of memory (bVlPartitionDropError).

Network traffic monitoring: For network traffic, at least the following is monitored by the off-chip monitoring service on the level of each individual virtual link.

- Length error (nLengthError): a frame received exceeds the configured maximum length for the specific VL.
- Timing error (nTimingError): Frame received outside of the expected window (wrong timing).
- Unreleased error (nUnreleased): Frame received while previous frame was still unread.

1.4.9 Serialization service (timely block & shuffling)

The serialization service forwards the messages from the egress queues to the MAC layer according to the priority. The highest priority is assigned to periodic messages, whereas aperiodic messages have the lowest priority.

Also the serialization service uses one of the following mechanisms to solve the collision between different traffic types, the shuffling or timely block mechanisms. The timely block mechanism disables the sending of other messages in the router-core during a guarding window prior to the transmission of a periodic message. For the shuffling mechanism, no guarding window is needed. In the worst-case, the router-core delays a periodic message for the duration of maximum size message. In addition, the message serialization supports timely block and shuffling service as descried earlier in (part II, section 1.1.6).

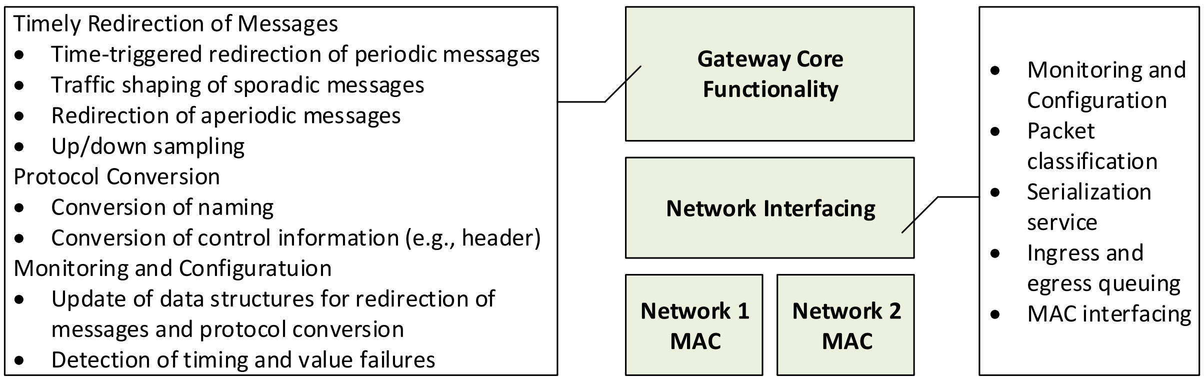
1.4.10 MAC interfacing

The MAC interfacing service is identical to the one of the off-chip network interface (cf. part II, section 1.3.10).

Group of Gateway Services

In order to establish the end-to-end communication over heterogeneous and mixed-criticality networks DREAMS gateways are used. The connection between off-chip networks, as well as between off-chip and on-chip networks is established through gateways as illustrated in Figure 28. The gateway consists of gateway core functionality, network interfacing and network MACs.

The gateway core is responsible for redirecting incoming messages based on timely redirection, protocol conversion, monitoring and configuration services. The network interfacing provides the interface between the MAC and the gateway core. Furthermore, classification and serialization of the packets is performed in the network interfacing. In order to realize fault-tolerance, the gateway can include multiple network MACs. Each network MAC connects the gateway to either an off-chip network (e.g., TTEthernet) or an on-chip network (e.g., STNoC). In case of network redundancy, multiple network MACs are required. Thus, the network interfacing is responsible for merging identical incoming messages and duplicating outgoing messages to be sent to different MACs.


Timely Redirection of Messages
Time-triggered redirection of periodic messages
Traffic shaping of sporadic messages
Redirection of aperiodic messages
Up/down sampling
Protocol Conversion
Conversion of naming
Conversion of control information (e.g., header)
Monitoring and Configuratuion
Update of data structures for redirection of messages and protocol conversion
Detection of timing and value failures
Gateway Core Functionality
Network Interfacing
Network 1 MAC
Network 2 MAC
Monitoring and Configuration
Packet classification
Serialization service
Ingress and egress queuing
MAC interfacing
Figure 28: Gateway

We can distinguish two types of DREAMS gateway: modular gateways and integrated gateways as shown in Figure 29. The integrated gateway combines the gateway core with the network interfacing. In the modular gateway type, the gateway core is realized on top of the network interfaces. The functionality of both types is similar but the required buffer capacity and the delays of the modular gateway will be higher than in case of an integrated gateway.

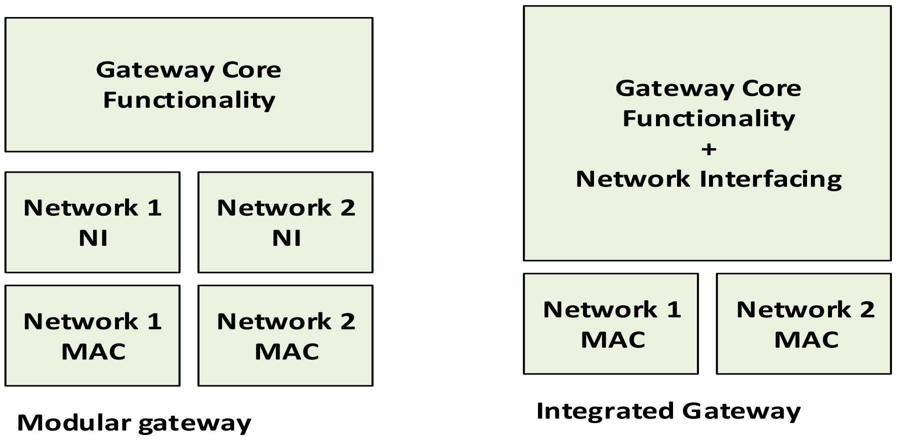


Figure 29: DREAMS Gateway Types

The network interface of the modular gateway can be instantiations of the ones presented in part II, sections 1.1 and 1.3. In case of the integrated gateway, the architecture is illustrated in Figure 30.

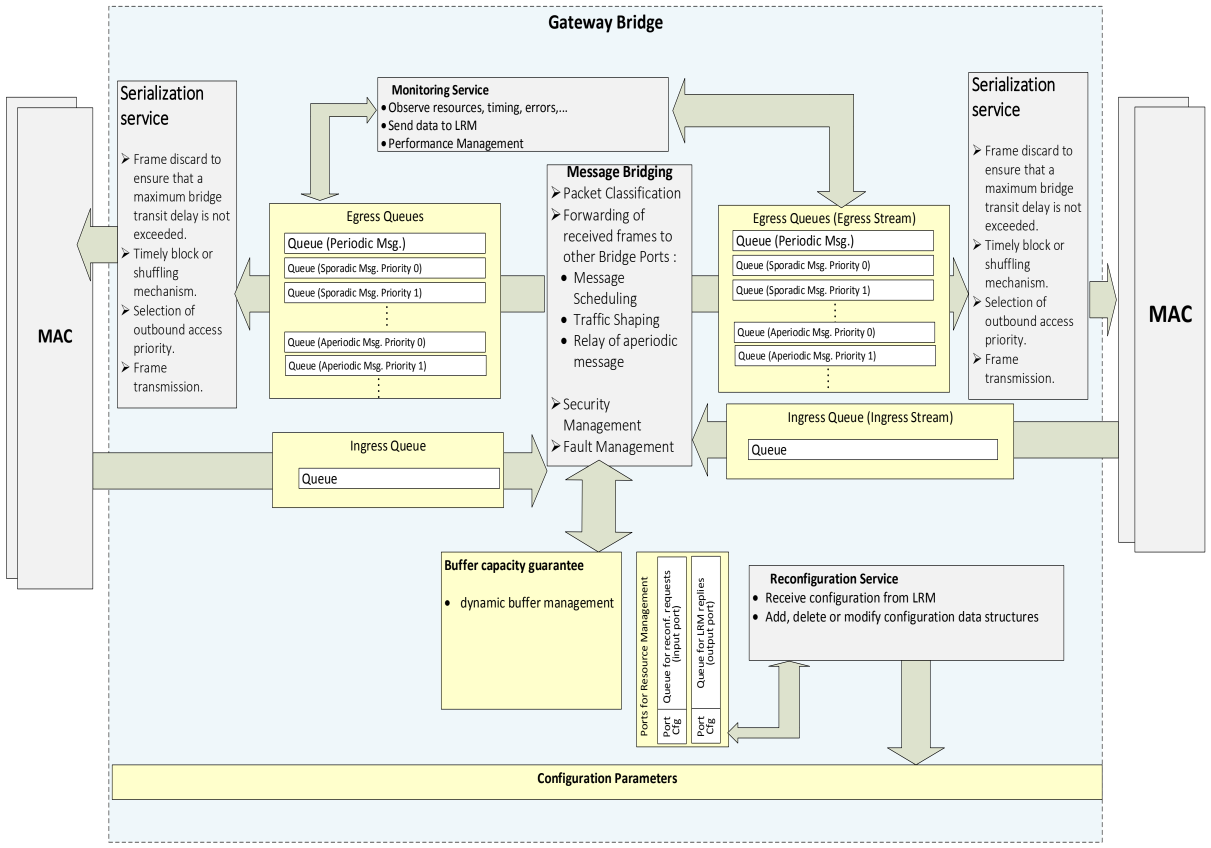

Gateway Bridge
MAC
Serialization service
Frame discard to ensure that a maximum bridge transit delay is not exceeded.
Timely block or shuffling mechanism.
Selection of outbound access priority.
Frame transmission.
Monitoring Service
Observe resources, timing, errors,…
Send data to LRM
Performance Management
Egress Queues
Queue (Periodic Msg.)
Queue (Sporadic Msg. Priority 0)
Queue (Sporadic Msg. Priority 1)
Queue (Aperiodic Msg. Priority 0)
Queue (Aperiodic Msg. Priority 1)
Message Bridging
Packet Classification
Forwarding of received frames to other Bridge Ports :
Message Scheduling
Traffic Shaping
Relay of aperiodic message
Security Management
Fault Management
Egress Queues (Egress Stream)
Ingress Queue (Ingress Stream)
Queue
Ingress Queue
Buffer capacity guarantee
dynamic buffer management
Ports for Resource Management
Queue for reconf. requests (input port)
Queue for LRM replies (output port)
Port Cfg
Reconfiguration Service
Receive configuration from LRM
Add, delete or modify configuration data structures
Configuration Parameters

Figure 30: Architecture of the Integrated Gateway

1.5 Gateway Core Functionality

The services of the gateway core functionality are as follows.

1.5.1 Configuration parameters

The configuration parameters of the gateway are as follows:

- ***Guaranteed buffer capacity***: Each ingress queue, egress queue and port is associated with a corresponding guaranteed minimum buffer capacity. The buffer capacity is determined by the maximum message and the message timing. This buffer capacity can avoid message omission of sporadic and aperiodic messages based on rate-constrains and message periods. The guaranteed buffer capacity can also be realized using dynamic memory management.
- ***Address information of ports***: The virtual link associated with a port and the data direction (from the off-chip network, to the off-chip network) are defined.
- ***Message type***: The message type is defined such as periodic, sporadic or aperiodic.
- ***Timing parameters***: In case of periodic messages, the parameters include the period and phase. For sporadic messages, the priority, the interarrival time and the jitter are specified. In case of aperiodic messages, no timing parameters are required.
-

1.5.2 Packet classification service

This service is responsible for classifying the incoming messages from the MAC in order to decide on the corresponding buffer (i.e., ingress and egress) according to message type and the configuration parameters. Additionally, the packet classification service will check the message format and its control information (e.g., VLID). In case the message has an invalid message frame, it will be discarded.

Moreover, the packet classification service uses the configuration parameters to check the integrity and validity of the periodic and sporadic messages. This includes the verification of the message size, checking whether messages arrive with correct VLID. In addition, it checks whether the periodic messages arrive within the specified receiving windows of the virtual link.

1.5.3 Message scheduling service

This service guarantees the determinism of the periodic message communication behaviour within the on-/off-chip gateway. Each periodic message has predefined parameters such as period and phase. According to the predefined configuration for the message scheduling, this service determines the point in time when the periodic message is relayed.

1.5.4 Traffic shaping service

This service is responsible for guaranteeing the minimum interarrival time between two consecutive sporadic messages on the respective virtual link. The minimum interarrival time and other parameters are available in the port configuration for each virtual link.

1.5.5 Relaying of aperiodic messages

This service is responsible to relay the aperiodic messages between ingress and egress queues based on the respective direction and the destination address.

1.5.6 Monitoring service

This service is responsible for observing the system resources, timing restrictions and unexpected system behaviour. The monitored data will be sent to the LRM to collect feedback and observe the

DREAMS chip. This collected data will possibly be used later in the GRM in order to reconfigure the network.

1.5.7 Down sampling

This service provides the message exchanges between networks with different periods of periodic messages or different rate-constraints of sporadic messages. The gateway has to redirect a subset of the incoming messages to satisfy the timing requirements of the target network. In addition, the redirection needs to be synchronized to ensure the forwarding of consistent data.

In the down sampling service, the gateway will send the most recent periodic message that arrived before the next sending time point. In case of the sporadic messages, the traffic shaper will drop all messages that arrive within the minimum interarrival time.

1.5.8 Protocol conversion

DREAMS supports virtual links over networks with different off-chip and on-chip communication protocol, e.g., time-triggered Ethernet, EtherCat and STNoC. Therefore, the gateway is responsible for adapting the message format according to the used communication protocol (e.g., header with address information, flow control, CRC). The conceptual logical and physical address space of DREAMS (cf. Part I) needs to be mapped to each network protocol.

The protocol conversion service is responsible of two major functions, encapsulation and decapsulation of the incoming and outgoing messages. The message format (described in part I, section 1.1.2) two styles according to the traffic types which need to be mapped to the respective network protocol:

- Periodic and sporadic <VLID, data>.
- Aperiodic <logical name sender, physical name receiver, data>.

In case of the periodic and sporadic messages, the VLID implicitly entails the source port, the destination ports, the path on the on-chip and off-chip networks and the message timing. For aperiodic messages, the information of the destination is encoded explicitly inside the message format as part of the logical/physical names.

1.6 Network Interfacing Services

The network interfacing services encompass several services that are also provided by the off-chip and on-chip network interfaces.

1.6.1 Buffer capacity guarantee

This service guarantees sufficient queue capacity for ingress and egress ports to avoid message loss based on the time behaviour of the periodic and sporadic messages. The buffer capacity guarantee can also be realized using dynamic buffer management mechanisms.

1.6.2 Egress queuing service

The egress queuing service is explained in part II, section 1.3.1.

1.6.3 Ingress queuing service

The ingress queuing service is explained in part II, section 1.3.2.

1.6.4 Configuration parameters

The configuration parameters are explained in part II, section 1.5.1.

1.6.5 Serialization services

The message serialization service is identical to the serialization service of the off chip communication router (cf. part II, section 1.4.9).

1.6.6 Monitoring service

The monitoring service is explained in part II, section 1.5.6.

1.6.7 Reconfiguration service

The reconfiguration service is explained in part II, section 1.4.7.

1.6.8 MAC interfacing

The MAC interfacing service is explained in part II, section 1.3.10.

Group of Shared Memory Services

1.7 Shared Memory Services

The shared memory model is supported on top of message-based networks. A shared address space is established for external memories and input/output devices. Thereby, application subsystems can exploit programming models based on shared memory in addition to message-based interactions, while exploiting the temporal and spatial partitioning of the message-based network infrastructure.

1.7.1 Address space/memory mapped accesses

Shared Memory communication allows efficient data exchanges between multiple programs running on the same processor of a tile. This can be further extended to the communication of multiple threads within a single program.

From the hardware perspective, shared memory consists in a typically large amount of RAM that can be accessed by means of read/write instructions issued by several CPUs in a multiprocessor computer system. This requires that all CPUs implicitly share a common application memory space, which classically consists of several on-chip DDR memory dies accessed through on-chip DDR memory controllers.

From the software perspective, two processes communicating through shared memory are using the same physical memory location as their regular working memory. This requires that the two processes are located on the same machine (running a given OS/hypervisor). While being very fast (the communication between the processes happens with a data rate in the order of a memory access), specific care must be taken with respect to memory inconsistency when the communicating processes are executed on two different CPUs. An underlying cache coherent architecture is necessary in this case. Cache coherency might be guaranteed using cache controllers coupled to OS services. In this case, part of the shared memory traffic will be constituted by read/write accesses generated by the cache controllers upon cache refill, cache miss (reads) or cache flush, clean (writes) events.

Considering the DREAMS targeted architecture and assuming a 32-bits address space, a proposed address space map model might be the one of Table 5.

The architectural style abstracts from the fine grain details of the on-chip/off-chip architecture as for example the size of the embedded RAM for each processor on chip as well as the size for the DDR shared memory. This will be evolving with the forthcoming definition/refinements of the DREAMS chip.

DDR controller 0
DDR controler 1
CPU0 dedicated space
...
CPUn dedicated space
Gateway dedicated space
Local on-chip periph, flash... dedicated space

Table 5: Memory Map example

1.7.2 Write access service

Write operations (also known as store operations) are used to write data in a shared memory location. Then any other process implied in a communication exchange may observe the data at the same memory mapped address. Writes are classical operations from the processor's instruction set for which IP protocols and underlying on-chip communication layers such as bus or NoC offer full support. Note that write operations might be used not only for writing data structure (such as strings, tables…) but also to access memory location that can be considered as flags. Furthermore, write operations reaching the shared memory might not always be generated from the processor itself but from an intermediate communication stage, such as a cache controller executing a flush/invalidate/write back operation.

1.7.3 Read access service

Read operations (also known as Load operations) are used to read data in a shared memory location. Reads are classical operations from the processors instruction set for which IP protocols and underlying on-chip communication layers such as bus or NoC offer full support. Note that read operations might be used not only for accessing data structure (such as strings, tables…) but also to access memory location that can be considered as flags. Furthermore, read operations reaching the shared memory might not always be generated from the processor itself but from an intermediate communication stage, such as a cache controller executing a speculative fectch/cache refill operation.

1.7.4 Shared memory coherency service

To avoid data inconsistency in shared memory, when in multiprocessor context, special care must be taken with respect to memory coherency. Furthermore, modern (=current) generations of processors embed L1/L2 caches memory which speed up access to memory at the cost of an higher processing for maintaining the cache/shared memory coherency.

Coupling Cache Controllers allowed operations to services offered by modern multiprocessor real-time OS, Cache coherency and shared memory consistency is a well-defined problem with standard solutions. We propose to rely on existing offered SW services for this aspect in DREAMS.

1.7.5 Monitoring and configuration services

For these resource management services, a shared memory controller must be assumed. Depending on the model chosen, different criteria might be monitored (internal queues status, number of page misses/hits) or reconfigured (internal queues allocation, power-off of part of the memory area, etc.).

However, the topic is too wide to be addressed at this point of the project without previous knowledge of the memory controller to be considered in DREAMS.

Group of IOMMU Services

1.8 System Services offered by IOMMU & NoC Firewall Components

This section provides a generic description of the I/O Memory Management Unit (IOMMU) and NoC Firewall services which relate to global shared physical memory and the off-chip gateway. These services will be further developed in DREAMS and are compliant with ARM v7 processor architecture and related virtualization extensions.

1.8.1 IOMMU address translation service

The I/O memory management unit (IOMMU) is a system module designed to translate addresses from the virtual space of a guest device to global shared physical address space, thereby managing how a DMA request originating from a device accesses external shared memory. This translation is similar to a processor's Memory Management Unit (MMU), except that the IOMMU translates memory accesses of fully virtualized devices rather than the CPU, as the MMU does.

1.8.2 Secure memory access services with page-level granularity

IOMMU functionality is not limited to translating device DMA addresses to physical addresses via virtual address translation. The IOMMU provides also secure memory access services by isolating the device accesses using page-level granularity. For instance, in a virtualization-aware environment, the hypervisor can configure (or remap) the I/O page tables of each device to safely map a device to a particular guest OS without risking integrity of other guests, i.e. a guest cannot break out of its address space with rogue DMA traffic. Additionally, the IOMMU is designed to provide an increased amount of security in scenarios without virtualization. In particular, the OS must be able to protect itself from buggy device drivers by limiting a device's memory accesses and managing the permissions of peripheral devices. Typically, upon an address translation request from a device, the IOMMU consults the I/O page table to find the physical page address. If a device tries to access memory without a valid entry in its I/O page table, then the IOMMU will access a default translation context and inform the hypervisor through an interrupt (or reject the access if configured to do so); notice that different types of system exceptions can occur, such as address translation requests from a device with uninitialized context, and even more critical security-related events, such as request access violations arising from malevolent or corrupt devices, such as DMA controllers.

1.8.3 IOMMU monitoring service

The IOMMU can also provide **monitoring services** focusing on page-level access granularity through a specialized hardware monitoring unit (HMU). This IOMMU module is able to monitor particular events related to: (i) internal IOMMU activity (counter statistics and error logs) and (ii) interface transactions (AMBA AXI bus). These events can be used to perform access pattern analysis, estimate key performance metrics, e.g. latency, throughput and resource utilization, and optimize the architecture by introducing novel decision control mechanisms, including system-wide services for

- **Dynamic management**, such as I/O remapping,
- **Performance-oriented system adaptation**, including pre-fetching and/or pinning of certain pages (e.g. this in particular is related to hard real-time processing at process- or VM-level), and
- **Fault tolerant services**, such as dynamic reconfiguration of the page entries in order to recover from hardware faults.

1.8.4 Virtualization-aware hardware NoC Firewall service

In addition to IOMMU services, a virtualization-aware hardware NoC Firewall unit at the on/off-chip network interface can support **VM isolation services** throughout the multicore SoC by tagging NoC transactions, establishing access rules for virtual components on physical address regions and ensuring that rules are obeyed at each network interface. One possible solution envisioned in DREAMS targets fine grain rule-checking at memory page-level by invoking a rules table walk to an external memory which stores the rules defined with page-level granularity.

The NoC Firewall concept supports multi-compartment philosophy [Fiorin2010, Porquet2011], extending existing protection mechanisms available in virtualization-aware technologies, such as ARM v7 Trustzone architecture (and related IOMMU support) [ARM2010]. More specifically, ARM v7 Trustzone architecture defines only two security domains (secure and non-secure) identified using an NS bit available within the memory page descriptors. Notice that alike our rules, the NS bit can be statically set (e.g. for a secure or non-secure peripheral), or dynamically modified either at boot time or by a system security thread.

Group of Communication Security Services

In the following, the security services regarding communication are described. There are two different types of communication in the DREAMS architecture: the on-chip communication and the off-chip communication. Based on the threat model for communication services (section 2.3.2) the on-chip communication is proceeded in a trusted zone. Hence, security services like en-/decryption or authentication are only needed for off-chip communication.

1.9 On-Chip Communication Services Security

1.9.1 Access Control Service

The access control service verifies if a system resource is allowed to access the requested resource. The on-chip communication access control service verifies the permission for components connected to the NoC, e.g., on-chip/off-chip gateway and memory-controller or the related network interfaces respectively

1.10 Off-Chip Communication Services Security

1.10.1 Encryption Service

The encryption service encrypts data with a given cryptographic key. It transforms a plain text into a cipher text. The encryption service for off-chip communication is used for confidential communication between two on-chip/off-chip gateways. No component of the off-chip network as well as other gateways can interpret the content of the communication even if they can read it.

1.10.2 Decryption Service

The decryption service decrypts data with a given cryptographic key. It transforms a cipher text into a plaintext. The plaintext is correctly recovered only if the key is correct and there was no transmission error. The decryption service for off-chip communication is used for confidential communication between two on-chip/off-chip gateways. No component of the off-chip network as well as other gateways can interpret the content of the communication because they do not possess the right key for decryption. Only the on-chip/off-chip gateways owning the cryptographic key can decrypt the data.

1.10.3 Integrity Service

The integrity service generates a cryptographic checksum for a message, which is transmitted together with the message. With this checksum, any modification in the message is detectable. The integrity check service for off-chip communication ensures that changes during the off-chip communication are noticeable.

1.10.4 Integrity Check Service

The integrity check service verifies the integrity of a message by re-calculating the cryptographic checksum on the received message and comparing it with the received checksum. The integrity check service for off-chip communication ensures that changes during the off-chip communication are noticeable.

1.10.5 Authentication Code Generation Service

The authentication code generation service generates a message authentication code (MAC) tag or digital signatures on the message for ensuring the data origin as well as to verify the communication partner. This service generates the authentication code at the on-chip/off-chip gateways for authenticating the off-chip communication.

1.10.6 Authentication Code Verification Service

The authentication code verification service verifies the data origin or the communication partner by verifying the received MAC tag or digital signatures along with the message. This service checks the authentication code at the on-chip/off-chip gateways for authenticating the off-chip communication.

1.10.7 Access Control Service

The access control service verifies, if a system resource is allowed to access the requested object. For off-chip communication the access control service checks, if a component has the permission to communicate through the on-chip/off-chip gateway. The gateway checks both directions, the on-chip/off-chip communication and the off-chip/on-chip communication.

2 Core Platform Services – Global Time

Synchronization Problem Formulation

From a generic point of view the clock synchronization problem appears in systems that consist of entities connected to each other by using a network, where some, or all of, the entities are equipped with local clocks. In such systems, the aim of the clock synchronization services is then to establish a concept of "global time" between those entities that have local clocks. Typically, global time is defined as follows:

Definition 1 – Global Time: Global time is established, when the distributed local clocks in a system, which are usually implemented as counters (e.g., as SW variables or HW registers) have "about the same value" at "about the same points in real-time"

Clock synchronization services establish exactly that. Consequently, as the presented definition of a global time is vague it makes sense to discuss the clock synchronization services in a generic sense first and specialize them hand in hand with concretizing the definition of a global time, as we will do as an example for off-chip and on-chip networks. In this section we give a general overview of the clock synchronization problem.

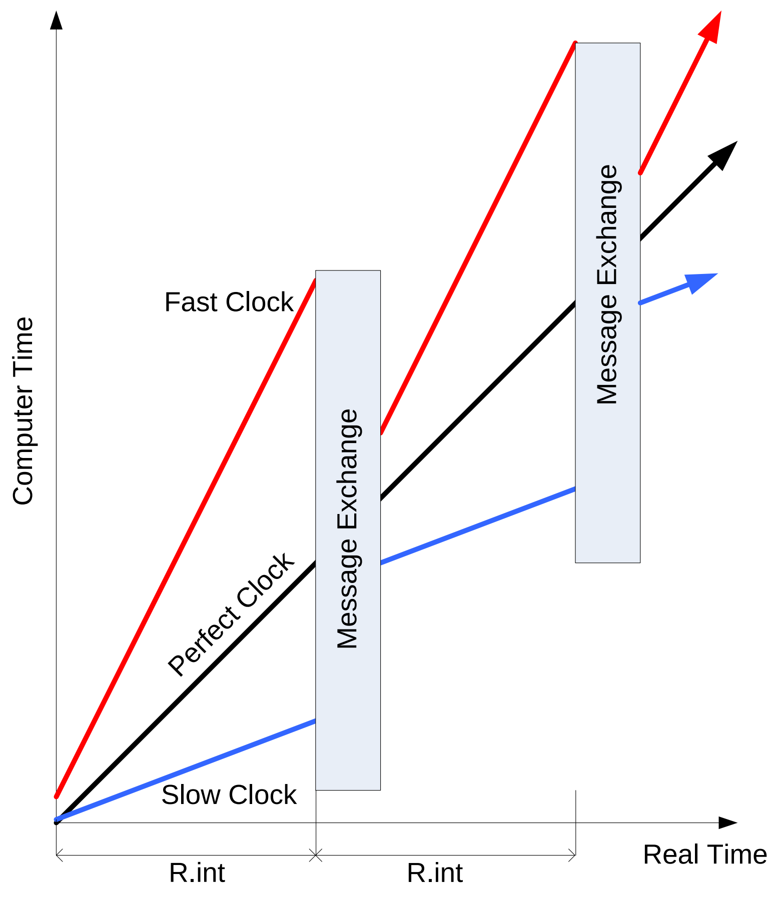


Figure 31: Computer Time vs. Real Time

In the following we will use Figure 31 in the discussion of the generic clock synchronization services. The diagram plots real time on the x-axis vs. computer time on the y-axis, where the computer time is the simulation of real time by the local clocks. The diagram depicts the traces of three local clocks, a slow clock, a fast clock, and a clock that perfectly resembles the progress of real time – the perfect clock. Applying Definition 1 to Figure 31, we get: at any point (or small interval) in real time, the difference in computer time of the local clocks in the system (in this case the system of the three clocks) has an upper bound. The key challenge in clock synchronization is to design specialized services that ensure that a given upper bound can be guaranteed in system-specific settings.

We will use the following quality aspects of synchronization in the discussion of the synchronization services:

- precision: worst-case difference of any two non-faulty clocks in the system
- accuracy: worst-case difference of the clocks in the system to an external time reference
- startup time: worst-case time after startup of the time sources until the system is synchronized (with given precision and/or accuracy)
- integration time: worst-case time for a non-synchronized component in the system to become synchronized
- changeover time: worst-case time for the components in the system to change from one time source to another one (e.g., in the case that the original time source fails)
- recovery time: worst-case time for the synchronized timebase to recover after global synchronization loss

Before discussing the generic services in detail, we should note that there is frequent ambiguity by what "time" actually means and we have found that the following differentiations help in the discussion.

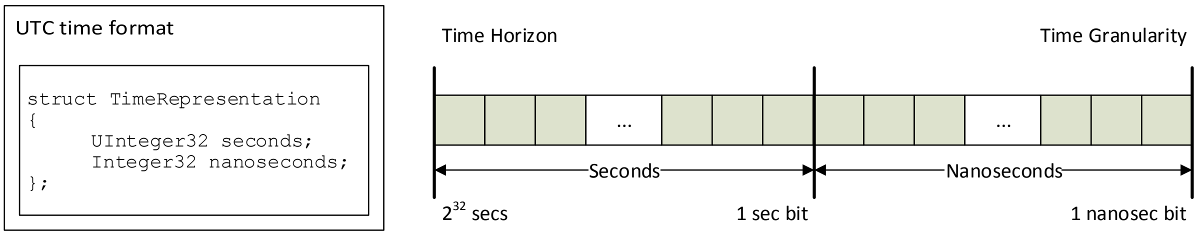
- Phase synchronization vs. TAI synchronization:
  - Phase synchronization refers to clock synchronization in a way that the time represented by the local clocks is a circular counter, e.g., starting with 0, and counting up to a maximum value (usually referred to as the "epoch" of time), once the epoch is reached the counter wraps around and starts counting at 0 again. Definition 1 holds as it stands above.
  - TAI synchronization in contrast to phase synchronization means that the local clocks are not only synchronized to each other in conformance to Definition 1, but TAI synchronization also requires the local clocks to represent Time Atomique International (TAI time).
- State synchronization vs. Rate synchronization
  - State synchronization refers to the process of the distributed local clocks instantaneously changing the current value of the counters that are used to implement the clocks.
  - Rate synchronization refers to the process of the distributed local clocks changing the rate according which the counters are updated. Rate correction can be done post-factum or into the future, or both: post-factum means that a local clock that found that it is currently deviating from other local clocks in the system applies rate correction for some time to gradually reach alignment with the other local clocks again, while into the future means that the local clock updates its rate with the aim not to generate a deviation to other local clocks in the first place.

Figure 31 depicts phase and state synchronization.

Time Representation

The representation of the global time base within the DREAMS architecture is based on a uniform time format for all configurations, which has been standardized by the IEEE Standard 1588 [IEEE1588]. A digital time format can be characterized by three parameters: granularity, horizon and epoch. The granularity determines the minimum interval between two adjacent ticks of a clock, i.e., the smallest interval that can be measured with this time format. The reasonable granularity can be derived from the achieved precision of the clock synchronization. The horizon determines the instant when the time will wrap around. The epoch determines the instant when the measuring of the time starts.

The unified time format (see Figure 32) is a binary time-format that is based on the physical second and nanoseconds. According to this time format, the highest possible granularity of the global time base is in nanoseconds.



```
struct TimeRepresentation
{
     UInteger32 seconds;
     Integer32 nanoseconds;
};
```

Figure 32 IEEE 1588 time format [IEEE1588]

The range of the absolute value of the nanoseconds member shall be restricted to:

$0 \leq |\text{nanoseconds}| < 10^9$

The sign of the nanoseconds member shall be interpreted as the sign of the entire representation and a negative timestamp shall indicate time prior to the epoch.

Note that the time horizon of the off-chip network and the on-chip network may differ and require conversion. In particular, the off-chip network provides global time only on a granularity of major and minor cycles of the communication as shown in Figure 33.

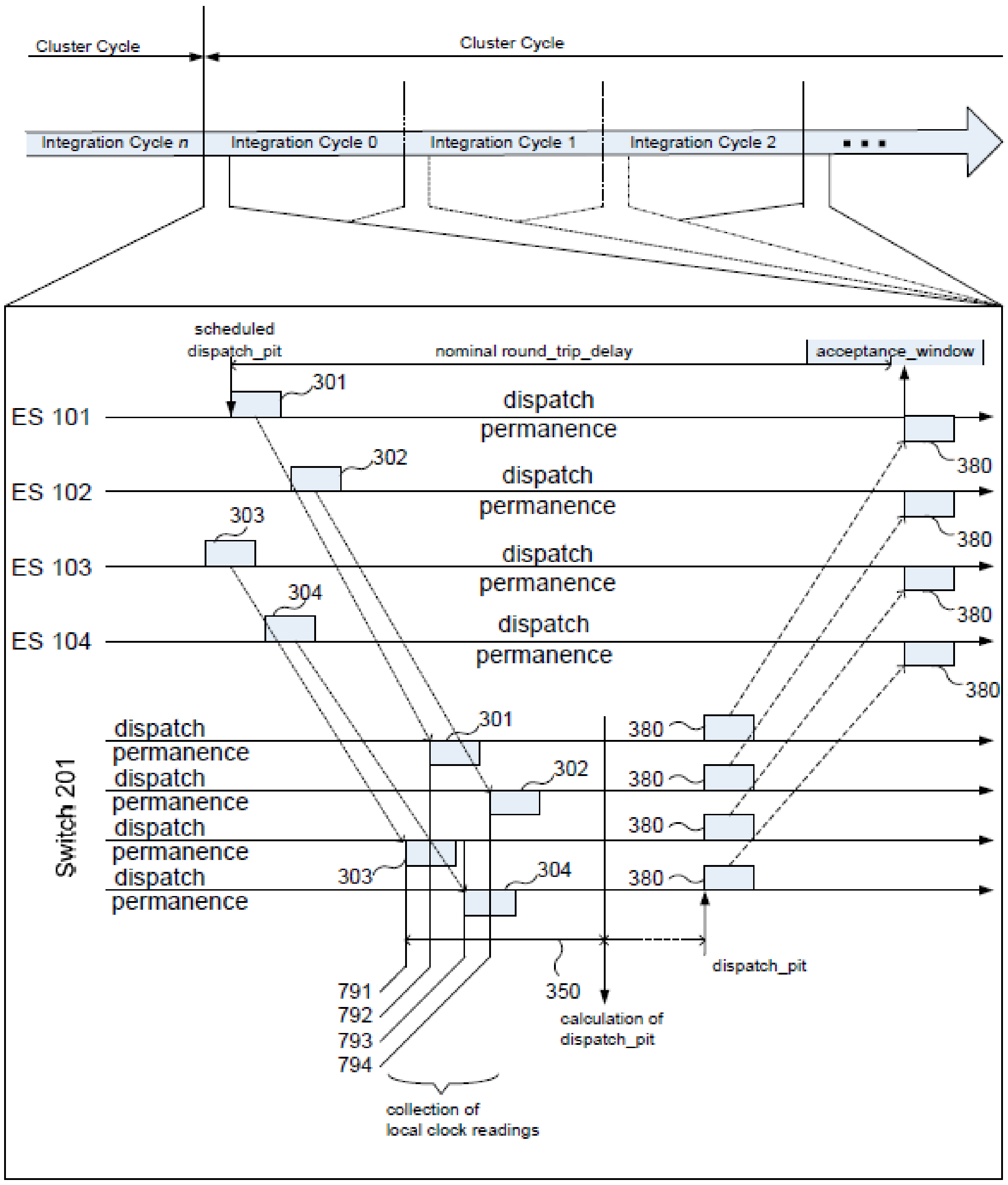


Figure 33: Off-Chip Global Time Granularity

2.1 On-Chip Clock Synchronization Service

In general, a multi-core chip cannot be assumed to provide a single clock signal for the entire chip. The reasons why designers introduce multiple clock domains include the handling of clock skew, the clocking down of individual IP blocks as part of power management, or the support for heterogeneous IP blocks with different speeds (e.g., high-clocked special purpose hardware and a slower general purpose CPU).

Despite the existence of multiple clock domains, the DREAMS architecture will support a global time base at chip-level that is also externally synchronized with respect to a chip-external reference time (i.e., the cluster-level global time base). Figure 34 shows the global time at chip-level and the provision of multiple clock domains by providing different clocks to different components.

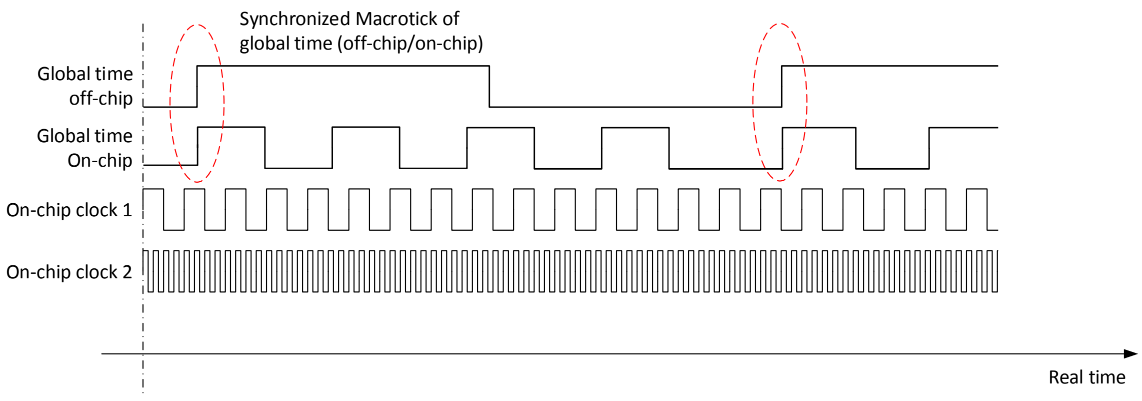


Figure 34: Example of different clock domains in DREAMS architecture

2.1.1 Different Clock Domains

The DREAMS architecture supports different clock domains by design. As shown in Figure 35, different parts of the system can operate at different clock speeds and components can include an arbitrary number of local clock domains, which are not visible outside of the tiles. For instance, a tile can be assembled by processor cores, memories, and network interface, which operate at their own frequencies.

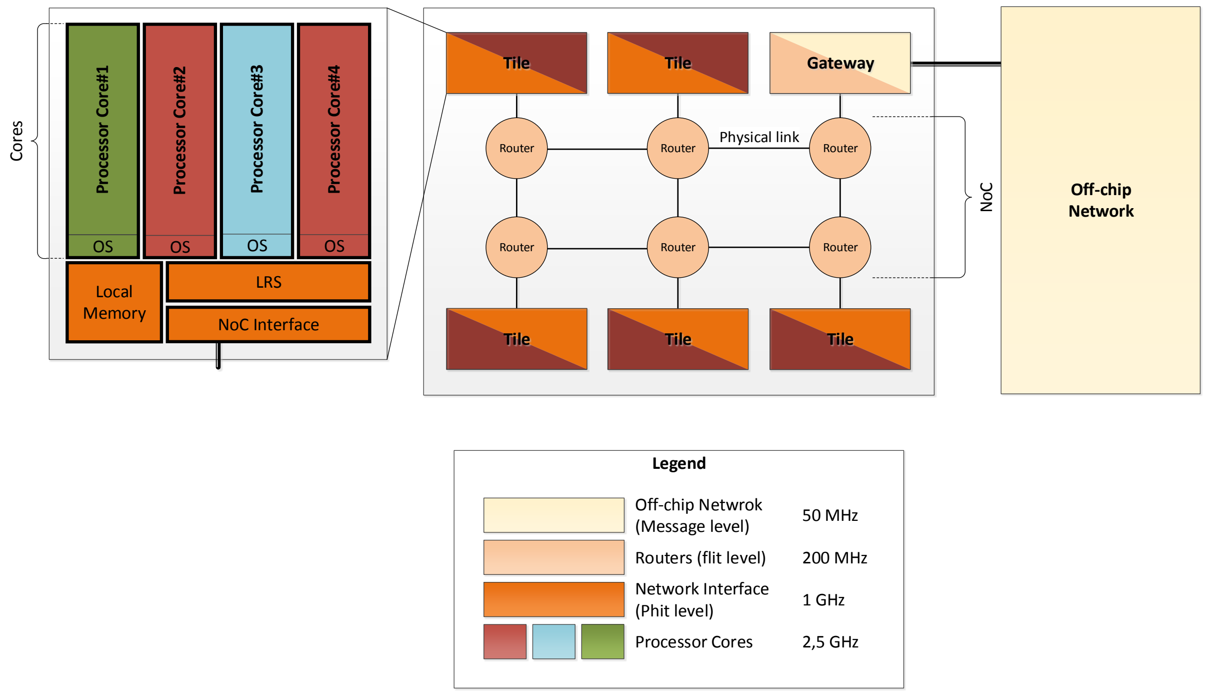


Figure 35: Example of different clock speeds at different parts of the system

On the other hand, the aim of DREAMS is to introduce an architecture which provides a system-wide synchronized global time base. This global time base allows the temporal coordination of actions on the distributed components (e.g., avoidance of contention at resources based on TDMA). In addition, timestamps assigned at different components can be related to each other. Timestamps become also meaningful outside the component where the event has been observed.

The global time base at chip-level embodies an independent clock domain, which typically has a lower frequency than the rest of the chip. This clock can be provided by a low-frequency global clock signal, thereby avoiding the problems that would be incurred by a high frequency global clock signal on the chip (e.g., clock skew). Alternatively, the global clock signal can be generated through internal clock synchronization (i.e., within the chip).

- **Global clock line**: as shown in Figure 36, a dedicated clock line will be available at each component (e.g., routers, processing cores, network interface, etc.) and each of them synchronizes itself with the provided clock reference.

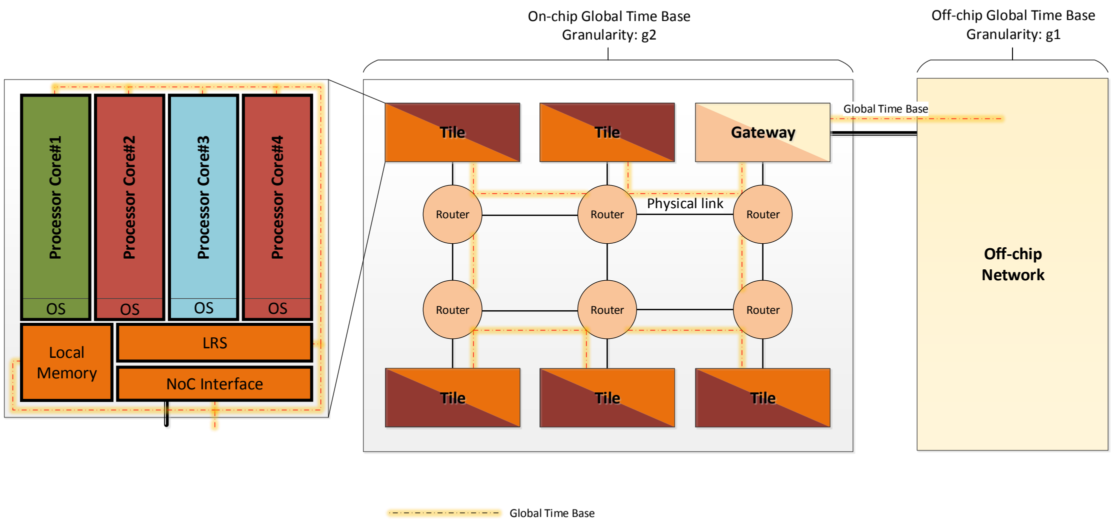


Figure 36: Global time base clock line

- **Message-based synchronization**: Alternatively, the value of the global time base can be provided to each component via a message based synchronization protocol. In this method, the value of global time base will be sent to the components in defined unified time format and they will update the local clock by either of mentioned synchronization methods.

For example, individual clock domains can operate in the range of GHz, whereas the global on-chip clock signal can have a lower frequency by several orders of magnitude.

The choice of the frequency determines the precision of the temporal coordination and the meaningful granularity of timestamps. In particular, the frequency of the global time base determines how densely a sequence of mutually exclusive distributed actions with time-triggered execution can be packed together while still avoiding collisions at the respective resources. (An example is given later for the on-chip communication.)

The existence of multiple clock domains, particularly of a global time base, entails the decoupling of synchronization of actions within the system and the operation of local entities. The global time base is allowed to maintain a relatively slow clock domain compared to the remainder of the system and the frequency associated with this clock domain determines the global granularity, to which actions in the system are synchronized. More precisely, the activities are not driven by the global time base, but they are synchronized by the global time base.

For instance, the on-chip communication of flits and phits can take place at a frequency that is higher than the rate of the global time base while operating in a synchronized manner with the global time base. The frequency at which the LRS at on-chip NI operates, is higher than the frequency of the global time base (as shown in Figure 34), but fully synchronized with it. In the example in figure 4, after every 16 clock cycles of the LRS there must be a single clock cycle of the global time base. This synchronization is necessary for the transmission of periodic messages. The global time is used at the LRS to align the start of the transmission of a periodic messages with other NIs, in order to guarantee bounded delay and minimum jitter for periodic messages (cf. Figure 37). In contrast, the global time base will not be necessary for sporadic and aperiodic transmission of messages.

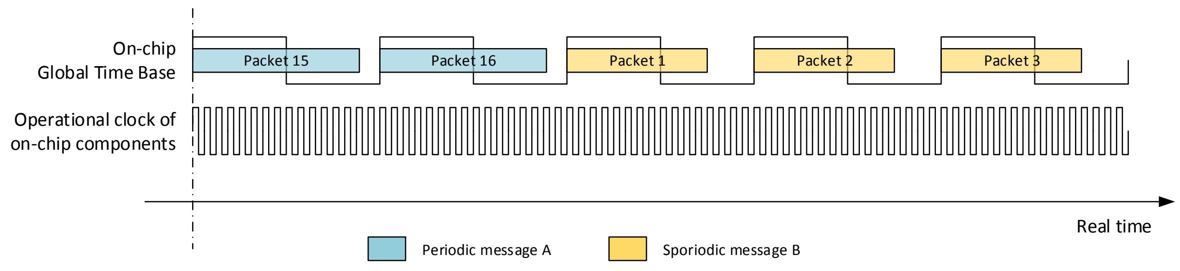


Figure 37 Global time base vs. transmission of packets and flits

2.1.2 On-Chip Synchronization

The synchronization between the on-chip global time base and the off-chip global time base is based on rate correction in combination with overflow time intervals. Figure 38 shows an example, where the on-chip global time base is four time faster than the off-chip global time base, but supposed to be synchronized, in a sense that each fourth rising edge of the on-chip global time is associated with a rising edge of the off-chip global time base. However, the on-chip global time base runs faster and as shown in the figure, after the fourth occurrence, the next rising edge waits until the rising edge of the reference clock, i.e., the off-chip global time base. The reflow interval determines the tolerable deviation between the rates of the off-chip and on-chip global time base.

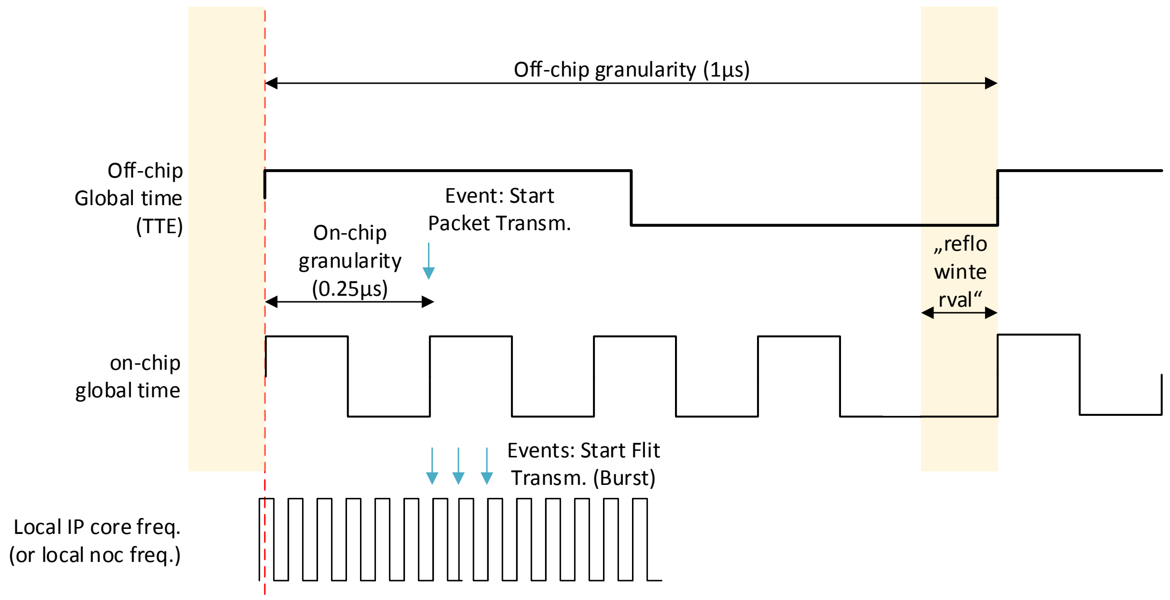


Figure 38 State synchronization for on-chip global time base

In addition, one can adjust the rate of the on-chip global time base in a way that in coming cycles the drift becomes smaller.

2.1.3 Loss of synchronization

We can consider a system as clock synchronization perspective in one of the following statuses:

- System wide synchronization: in this case, the synchronization between multiple clock domains is operating without any problem and all entities are well synchronized.
- Loss of off-chip synchronization (on-chip only): in case of a loss of off-chip clock synchronization, the on-chip transmission of periodic messages is still possible, since the NoC is still able to correct the on-chip clock with the global time base.
- Loss of global time base: if the synchronization with the global time base fails, the NoC will no longer be able to support the transmission of periodic messages in order to avoid

contention. In this case the subsystem which is unable to be synchronized with the global time base shall enter the safe state.

-

2.1.4 Monitoring and Reconfiguration

As mentioned in the previous sections, in some cases there is a need to reconfigure the clock system. For instance, in case of loss of the global clock line, the monitoring interface shall report the failure to the LRM in order to provide the new configuration. Furthermore, local modifications, for instance tuning frequencies in components and the communication subsystem clock parameters (e.g., horizon, epoch, etc.) can be established using the reconfiguration services.

2.2 Off-Chip Clock Synchronization Service

Assumptions:

- Distributed local clocks are being driven by independent oscillators.
- Non-negligible transport delays in the communication of the local clock values between nodes.
- Off-chip network implements the SAE AS6802 standard.

There are two different modes of operation in an off-chip network: normal operation and startup/restart. During normal operation the synchronization strategy assumes initial synchronization is established and maintains this synchrony. It is the task of the startup/restart to establish initial synchrony. The difficulty in designing a synchronization strategy for fault-tolerant systems is the transition from startup/restart to normal operation and vice versa.

Considering the mission time of a system, the number of synchronization processes executed under normal operation mode will by far outnumber the number of startup/restart processes which ideally occurs only once per mission time. Let's give a representative example: during normal operation mode re-synchronization may be scheduled with a period of 50 ms. Given a 10-hour flight, this means that the synchronization actions in normal operation mode will be executed 720,000 times, while the startup/restart occurs only once. These numbers are a solid basis that underlines our preference to keep normal operation mode and startup/restart separated over a combined synchronization approach.

Nevertheless, it must be guaranteed under a defined fault hypothesis that the startup/restart will be successful. The mere fact that startup/restart is an infrequent event does not relieve the algorithms from proper operation under failure conditions. A sound startup/restart is essential when the system is exposed to failure conditions that are at the limits of the failure hypothesis or even beyond.

For safety-critical systems SAE AS6802 specifies a fault-tolerant Multi-Master synchronization strategy, in which each component is configured either as Synchronization Master (SM), Synchronization Client (SM), or as Compression Master (CM). An example configuration is depicted in Figure 39. Typically the end systems would be configured as SM, while the central role of the CM suggests its realization in the switch in the computer network, though this is not mandatory. All other components in the network are configured as SCs and only react passively to the synchronization strategy. The synchronization information is exchanged in Protocol Control Frames (PCFs). There are three types of PCFs: integration (IN) frames are communicated in normal operation mode, coldstart (CS) and coldstart acknowledgement (CA) frames are communicated during startup/restart.

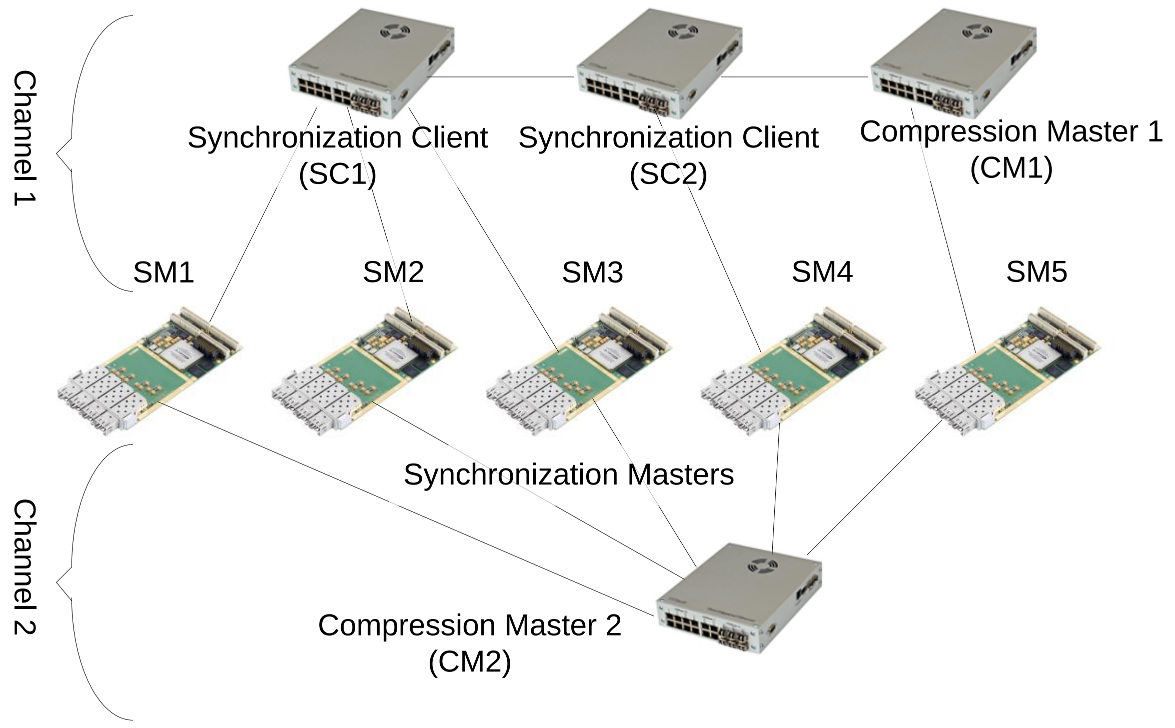

Channel 1
Synchronization Client (SC1)
Synchronization Client (SC2)
Compression Master 1 (CM1)
SM1
SM2
SM3
SM4
SM5
Synchronization Masters
Channel 2
Compression Master 2 (CM2)

Figure 39: Example configuration of the synchronization services for an off-chip network

2.2.1 Time-preserving transmission service

As discussed, in general entities use a network to exchange the current values of their local clocks. In order to allow synchronization at all, the network must provide a time-preserving transmission service with known timing error. For example if the local clock values are exchanged by using a message-based transmission service, the transmission latency and transmission jitter need to be predictable. The quality of the transmission latency and jitter of the service typically also directly influence the quality of the synchronization, i.e., the smaller the latency and jitter the better the local clocks can be synchronized to each other.

The off-chip network implements a one-step transparent clock mechanism – a mechanism implemented in the nodes and switches in the off-chip network to measure the delay of Ethernet frames used for the synchronization services. In particular the transparent clock mechanism operates as follows:

Ethernet frames used for the synchronization services, called Protocol Control Frames (PCFs) contain a field in their payload called “transparent clock”

The off-chip nodes and switches modify this transparent clock field in the following way

- Nodes will measure the duration it takes from the internal trigger to send a PCF until the first bit of the PCF will be transmitted on the Ethernet network and add this delay into the transparent clock field
    - Switches will measure the duration it takes from reception of a PCF until the forwarding of the PCF and add this delay into to the transparent clock field
    - Additionally the nodes and switches may add delays to the transparent clock field that reflect the transmission delays imposed by the wiring itself
    - A receiver of a PCF will thus be able to learn from the value of the transparent clock field inside the PCF, for how long the PCF has been in transmission.

2.2.2 Synchronization Startup Service

The synchronization startup service refers to the process of initially synchronization the local clocks to each other, e.g., after initial power up of the system.

During startup/restart in a multiple-failures hypothesis the SMs execute an interactive consistency agreement algorithm in which they negotiate the initial point in time. For this the SMs transmit\ dedicated PCFs, the coldstart (CS) and coldstart acknowledge (CA) frames. The CMs will only interfere minor in this negotiation process and synchronize to the SMs once startup/restart is finished. Once, the CMs see a sufficiently high number of operational SMs they will block coldstart frames and so prevent startup/restart initiated by a faulty SM (only relevant in a failure scenario with two faulty SMs).

For two-fault tolerance we assume an inconsistent omission failure mode for both, SMs and CMs. For single-fault tolerance there is also an option to configure the CMs to operate as central guardians. In the role of a central guardian the CM will then interfere more tightly with PCFs sent from the SMs which allows an arbitrary failure mode of the SMs.

2.2.3 Resynchronization Service (Clock Synchronization – state/rate)

The resynchronization service (typically referred to in literate as clock synchronization) refers to the process of periodically aligning the local clocks to each other. As discussed earlier re-synchronization can be done state-based or rate-based.

The local clocks in the off-chip network are resynchronized in two steps. In the first step, the SMs send PCFs to the CMs. The CMs extract from the arrival points in time of the PCFs the current state of their local clocks and execute a first convergence function, the so-called compression function. The result of the convergence function is then delivered to the SMs in form of new PCFs (the compressed PCFs). In the second step the SMs/SCs collect the compressed PCFs from the CMs and execute a second convergence function.

2.2.4 Integration Service

The integration service refers to the process of entities joining an already synchronized system, e.g., in case the entity is powered-on late or after a transient failure of the entity.

The nodes and switches in the off-chip network use information in the payload (the membership field) of the IN frames for the integration service:

- The membership field in the PCFs is a bit vector that statically associates each bit with a specific SM in the network.
- When the CMs generate the compressed PCFs, they will set the bit of a respective SM in the membership vector of the compressed PCF if the SM has provided a PCF and clear the bit otherwise.
- Thus, the compressed PCFs carry in the membership field a current view on how many (and also which) SMs are currently supporting a given timeline.
- A node and/or switch that is powered-up (or re-integrates) waits for at least one synchronization interval to receive an IN frame.
- If the number of bits set in the membership field of a received IN frame are equal or higher than an offline configured threshold, then the node/switch will adopt its local clock to the time associated with the received IN frame.

2.2.5 Clique Detection Service

The clique detection service refers to the process of detecting global synchronization failure, in particular the identification of situations in which several subsets of local clocks have been formed where the local clocks within the subsets are synchronized to each other but not over subset boundaries.

The nodes and switches use the membership information carried in IN frames also for clique detection. In particular the devices execute three clique detection mechanisms: synchronous clique detection, asynchronous clique detection, and relative clique detection.

2.2.6 Clique Resolution Service

The clique resolution service refers to the process of resolving clique scenarios once they have been formed. Clique resolution typically follows clique detection as discussed above. Once a device has detected a clique scenario it will resolve it by either executing the Synchronization Startup Service or the Integration Service.

2.2.7 Synchronization Restart Service

The synchronization restart service refers to the process of globally restarting the global time within a system. Synchronization restart can be a means for clique resolution.

See Synchronization Startup Service.

2.2.8 External Clock Synchronization Service

The external clock synchronization service refers to the process of synchronizing the local clocks to a system-external time source. Such a system-external time source may be for example a GPS receiver.

The nodes in the off-chip network can be configured to apply a configuration-specific value in addition to the value as calculated by the Resynchronization Service when resetting their local clocks. This mechanism can be used to synchronize the nodes to an external time source.

2.2.9 Time-Hierarchy Service (Up, Down)

The time-hierarchy service refers to the processes of translating global time between the different layers in the hierarchy of networks in the DREAMS architecture.

2.3 Using the clock synchronization services

At the application level these services should be transparent. To achieve this transparency, the Operating system or the virtualization layer should take into account the mechanisms proposed and offer the time services in a transparent way.

In DREAMS, the main component to provide the time services is the virtualization layer which should support the selected mechanisms. The main property for clock management in real-time applications is to deal with a monotonic increasing clock and timers based on it.

As stated above, the clock synchronization at node level can introduce some problems:

1. if the local clock is faster than the global clock, at synchronization time, the local clock can be set "before". This situation implies that the clock is not monotonic increasing.
2. if the local clock is slower than the global clock, at synchronization time, the local clock can be set "after" This situation can generate that some timers set on this clock can be past at time synchronization. The property is preserved but a jump in the clock can generate that several timers can expire at the same time. It can generate punctual overloads that should have to be considered in the real-time schedulability analysis.

The virtualization (operating system) layer has to provide methods and techniques to deal with the possible problems and to guarantee the correct system behavior. One of the solutions is to use the local clock for all the application services and to identify synchronization points that should not affect to the applications.

2.3.1 Synchronization time points (interval)

The virtualization layer or operating system has to define secure synchronization points to perform the clock synchronization. In order to consider the adjustment instead a point it is considered an interval. This secure interval have to fulfill two basic requirements:

- It shall set the local clock before or later without affecting the application behavior
- No application timers shall be pendent

This activity at virtualization layer or operating system has to be scheduled properly to guarantee the requirements. In the case of a virtualization layer with cyclic schedule, the secure interval to perform this synchronization is at the end of the major frame (MAF) that correspond to the hyperperiod. At the end of the MAF all periodic activities have been completed and no pending application timers should be set. The following figure shows the scheme for the MAF synchronization.

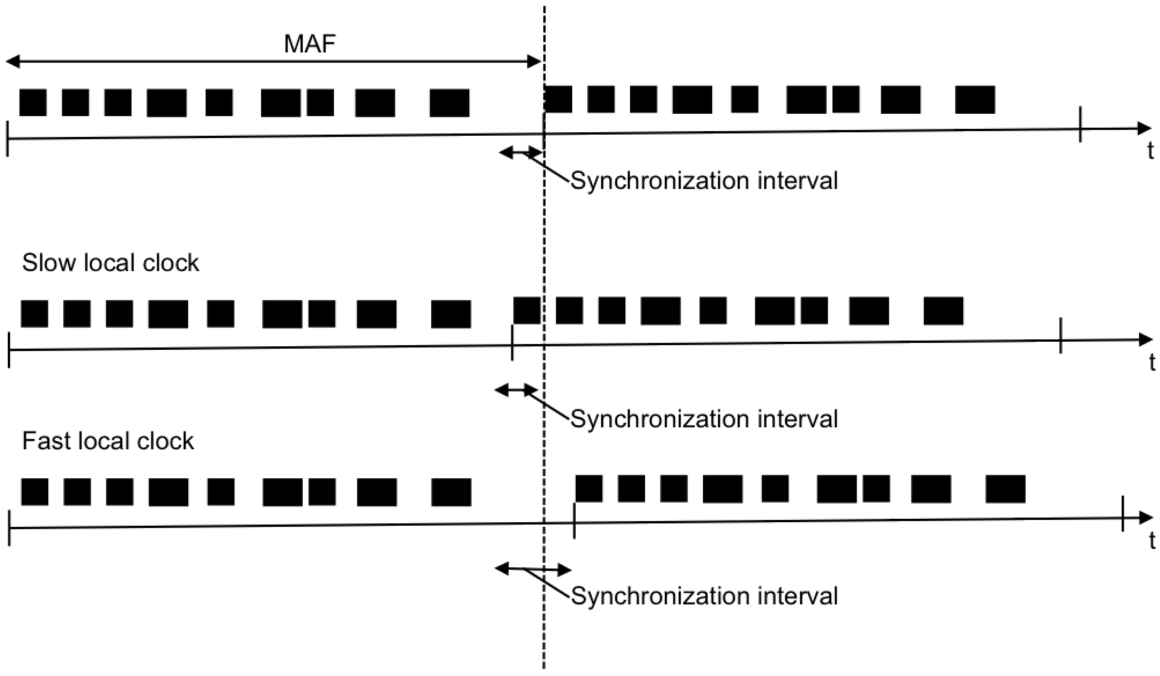


Figure 40: Synchronization interval at the end of the MAF

The figure shows how the clock synchronization is achieved. Next to the end of the MAF, the virtualization layer waits for the synchronization periodic message from the network. When the synchronization message arrives, the local clock is updated and the next MAF is executed.

Group of Global Time Security Services

As described in the threat model for global time services, there are two main attack targets on the global time services. On the one hand there are attacks against the clocks or the time values in the components itself, and on the other hand there are attacks against the time synchronization.

The attacks against the clock values itself can be prevented by using access control and authentication services. Only users or applications that are allowed to change the clock values can change these values.

The attacks focusing on the time synchronization can be prevented by using secure communication services. Depending on the type of the synchronization process, it could use either the services on the network level or the services on the application level. On the network level, the secure communication services from the DREAMS Communication Services will be used and on the application level, the secure communication services from the DREAMS execution services will be used.

3 Core Platform Services: Execution

An important part of the DREAMS architectural core services are the execution services that provide basic operations to run the system.

Software Architecture

This section describes the software architecture supported by DREAMS, that involve several applications with different levels of criticality. It details the execution environment and the services provided to support the application execution.

The software architecture is built on top of a DREAMS node that manages the entire tile including one or more processor cores.

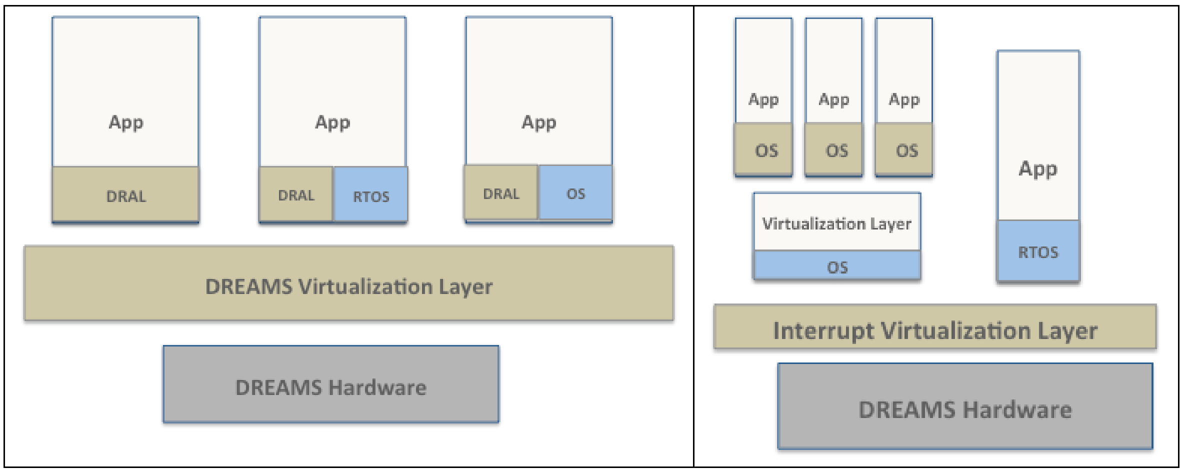


Figure 41: Software architectures

In order to support mixed-criticality applications, the DREAMS software architecture is composed by:

- Virtualization layer: It is a software layer that provides hardware virtualization to the applications. Two different approaches are considered in DREAMS depending on the application constraints.
    - Partitioning kernel: It provides virtualization of the hardware resources by defining a set of services that are used by the partitions to access the virtualized resources. The partitioning kernel provides spatial and temporal isolation to the partitions.
    - Interrupt Virtualization layer: This layer virtualizes the Host OS interrupts and is only introduced when KVM hypervisor is used. The main objective is to take hardware interrupts control away from Host OS and handle them in a thin layer, so as to preserve timing guarantees for the RTOS. Thus, an interrupt virtualization layer (ADEOS or similar) is introduced below the Host OS and real-time partition to prioritize the RTOS.
- Partitions: A partition is the execution unit in the DREAMS architecture. It provides the basic infrastructure to execute an application. Different partitions are supported in the DREAMS architecture.
    - Basic single-thread application to be executed near a native hardware
    - Multi-thread real-time applications to be executed on top of a real-time operating system
    - Multi process applications to be executed on top of a full featured operating system
    - Multi-partition applications to be executed on top of a operating system that provides the ability to build virtualized multiple process applications.

Figure 41 sketches the two proposed software architectures.

DREAMS Virtualization Layer

The virtualization layer is the software layer that abstracts the underlying hardware and provides virtualization of the CPUs. This virtualization layer is a hypervisor that permits to execute multiple isolated virtual machines. Each virtual machine is a partition.

As the virtualization layer is a common layer for all the partitions, in order to support mixed criticality applications, it has to achieve the highest level of criticality in the system.

The basic properties that the virtualization layer shall accomplish are:

- Spatial isolation: A partition is completely allocated in a unique address space (code, data, stack). This address space is not accessible by other partitions. The hypervisor has to guarantee the spatial isolation of the partitions. The system architect can relax this property by defining specific shared memory areas between partitions.
- Temporal isolation: A partition is executed independently of the execution of other partitions. In other words, the execution of a partition cannot be disturbed by the execution of other partitions. It influences directly on the scheduling policies at hypervisor level. The hypervisor has to schedule partitions under a scheduling policy that guarantees the partition execution.
- Fault isolation and management: A fundamental issue in critical systems is the fault management. Faults, when occur, have to be detected and handled properly in order to isolate them and avoid the propagation. A fault model to deal with the different types of errors is to be designed. The hypervisor has to implement the fault management model and permits to the partitions to manage those errors that involve the partition execution.
- Predictability: A partition with real-time constraints has to execute its code in a predictable way. It can be influenced by the underlying layers of software (guest-OS and hypervisor) and by the hardware. From the hypervisor point of view, the predictability applies to the provided services, the operations involved in the partition execution and the interruption management of the partitions.
- Security: All the information in a system (partitioned system) has to be protected against access and modification from unauthorized partitions or unplanned actions. Security implies the definition of a set of elements and mechanisms that permit to establish the system security functions. This property is strongly related with the static resource allocation and a fault model to identify and confine the vulnerabilities of the system.
- Confidentiality: Partitions cannot access to the space of other partitions neither to see how the system is working. From its point of view, they only can see its own partition. This property can be relaxed to some specific partitions in order to see the status of other partitions or control their execution.

Key properties for certification

From the point of view of certification/qualification the next properties are considered key elements:

1. Spatial and temporal isolation: It will allow that applications could be independently developed, analyzed and, consequently, certified/qualified. With respect to the virtualization layer, it has to provide the mechanisms to guarantee them. The temporal isolation involves two key aspects: temporal allocation of resources to the applications/partitions and interferences due to the parallel execution on other cores. The interference of other cores

due to shared resources could be removed by using appropriated hardware mechanisms or modeling the interferences and dimensioning appropriately the applications to take into account them and generating partition schedules to deal with it.
From the virtualization layer, under previous premises, the implication of the temporal allocation of partitions permits to offer the basic mechanisms to guarantee the temporal isolation.

2. Prevent the fault propagation: faults have to be detected and handled in the way that they do not influence the execution of the rest of the system. Health monitoring techniques at virtualization layer have to deal with the detection and management of faults providing the mechanisms to avoid fault propagation and monitor the generated faults to implement additional mechanisms for global fault management.

Static resource allocation: The system architect is the responsible of the system definition and resource allocation. This system definition is detailed in the configuration file of the system specifying all system resources, namely, number of CPUs, memory layout, peripherals, partitions, the execution plan of each CPU, etc. Each partition has to specify the memory regions, communication ports, temporal requirements and other resources that are needed to execute the partition code. Static resource allocation is the basis of predictability and security of the system. The hypervisor has to guarantee that a partition can access to the allocated resources and deny the requests to other not allocated resources.

Interrupt Virtualization Layer

KVM converts Host (Linux) Processes into virtual machines, and re-uses most of the common features provided by Host OS such as Process Scheduling, Memory Management, Interrupt Handling etc. In order to support a hard real-time partition, we can either introduce a thin interrupt virtualization layer below the Host kernel or modify most of the Host kernel sub-systems. The former approach is considered a better option, such as using ADEOS (Adaptive Domain Environment for Operating Systems) or a similar one than modifications to the Host kernel, thanks to its smaller TCB (Trusted Computing Base). For example, ADEOS “nanokernel” is composed of a few KLOC for ARM processors as opposed to a fully featured Host OS such as Linux, which has a very large TCB. Thus, an interrupt virtualization layer along with the KVM hypervisor is necessary for realizing the RTOS-GPOS co-existence use-case.

Previous real-time efforts for KVM hypervisor ([5], [6]) have focused on either semi-automatic virtual machine prioritization/shielding techniques or modification of guest system to realize a paravirtual interface. All of these techniques have failed to produce a hard real-time virtualization solution, so we consider them in-adequate for DREAMS project. Moreover, maintaining a paravirtualized solution is difficult as it requires modifications to the guest operating systems.

The interrupt virtualization layer schedules multiple operating system instances running above it, and allows for the co-existence of multiple prioritized domains (real-time and non real-time). This layer implements an interrupt management scheme, which allocates specialized interrupt handlers for the Host OS and RTOS. The RTOS-specific interrupts are given higher priority to ensure real-time behavior. KVM will run on the Host OS (within the non-RT domain) and create multiple virtual machines.

Execution Units: Partitions

The execution unit in partitioning is a partition. A partition is basically a program in a single application environment. It can comprise: the application code, the partition runtime and the configuration file. A partition runtime can have a minimal layer to facilitate the application execution and a guest Operating System adapted to be executed on top of the virtualization layer.

The software that resides in a DREAMS partition can be:

– Application software: It refers to the code designed to deal with the specific application requirements

– Runtime support. It provides the services to execute the application code.

Different types of partitions can be built:

- Bare Partitions: Partitions that are executed as they were on top of the hardware. The application code can be a single thread executed in one core or several single-threads.
- Real-Time Partitions: These partitions shall contain a real-time operating system adapted to be executed on top of the DREAMS virtualization layer. Additionally, it can include the DRAL layer that complement the RTOS services with specific services for partitioning. The partition boot is managed by the RTOS.
    - Real-Time Partitions (XtratuM Case): The real-time partitions for XtratuM will be similar to non real-time partitions, as XtratuM is a baremetal hypervisor and can fully control scheduling of these partitions.
    - Real-Time Partition (KVM Case): The real-time partition for KVM will be based on a minimal interrupt virtualization layer, in order to ensure hard real-time behavior for a given RTOS. This design change is necessary as KVM uses Linux kernel for scheduling its virtual machines, which is soft real-time at best. Figure 42 shows the two types of RT partitions.
- General purpose Partitions: These partitions shall contain a full featured operating system (e.g. Linux) that offers the OS services to the partitions. Additionally, it can include the DRAL layer that complements the OS services with specific services for partitioning. The partition boot is managed by the OS.

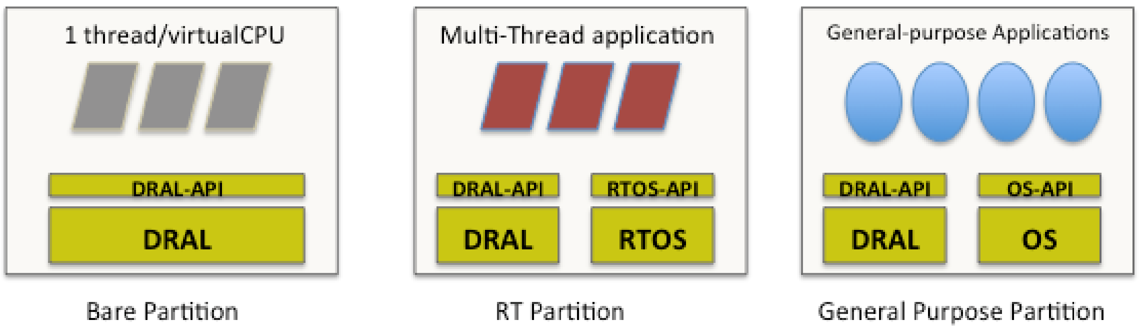


Figure 42: Partition Classes

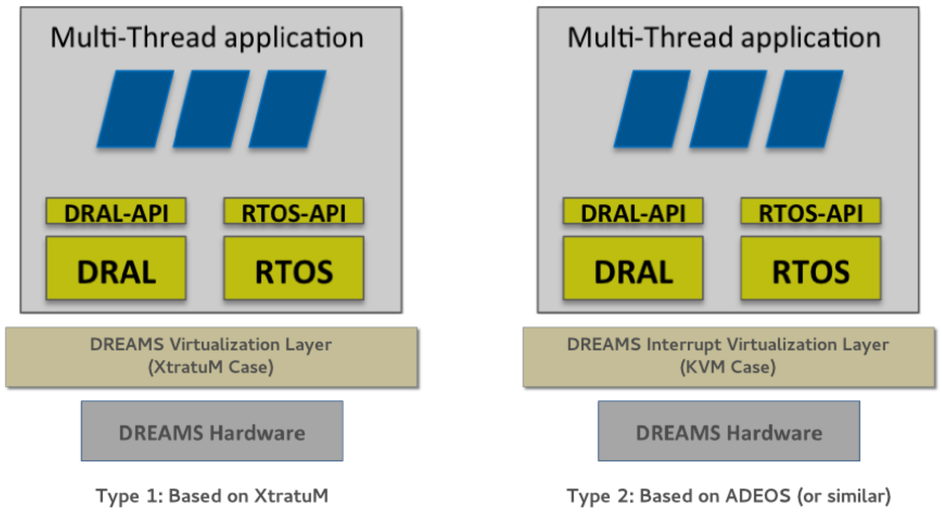


Figure 43: RT-partition types

Figure 43 shows the different types of real-time partitions. Independently of the partition class, a partition is seen by the virtualization layer as a piece of code with an entry point and a set of access points (communication ports) that allow to communicate it with other partitions. Figure 44 sketches the partition view.

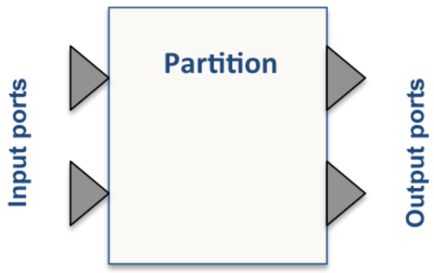


Figure 44: Partition view

Input and output ports permit to a partition to send/receive messages to/from other partitions. Services to deal with these inter-partition communications shall be defined. A message is a variable block of data that is sent from a source partition to one or more destination partitions. The data of a message is transparent to the message passing system.

The message transport mechanism is a communication channel that is the logical path between one source and one or more destinations. Partitions send/receive messages through ports. The virtualization layer is responsible of the message transport from the memory area of a source partition to a memory area of the destination(s) partition(s).

Two basic inter-partition communication ports are supported: sampling and queuing.

- Sampling port: It provides support for broadcast, multicast and unicast messages. No queuing is supported in this mode. A message remains in the source port until it is transmitted through the channel or it is overwritten by a new occurrence of the message, whatever occurs first. Each new instance of a message overwrites the current message when it reaches a destination port, and remains there until it is overwritten. This allows the destination partitions to access the latest message.
- Queuing port: It provides support for buffered unicast communication between partitions. Each port has associated a queue where messages are buffered until they are delivered to the destination partition. Messages are delivered in FIFO order.

Channels, ports, maximum message sizes and maximum number of messages (queuing ports) are entirely defined and allocated off-line.

Partition Types

Depending on the partition rights, a partition can be defined as:

- System partition: System partitions are allowed to manage and monitor the state of the system and other partitions. A subset of services of DRAL dealing with the change of the state of the system or another partition only can be invoked if the partition is defined as system partition.
- Real-time system partition: A real-time system partition only exists when an interrupt virtualization layer is used (KVM case). This partition is similar to a system partition, except that it is dedicated for real-time OS and will have a unique instance on a given DREAMS chip.
- Normal partition: It corresponds to the partitions that have not the system attributes.

Considering the virtual cores that a partition uses, it can be:

1. Mono-core partition: This partition only uses a virtual core.
2. Multi-core partition: This partition is associated with several virtual CPUs. The virtualization layer only boots the virtual CPU0, it is responsibility of the partition, to boot the rest of the virtual CPUs.

Partition states and transitions

The virtualization layer is not aware about the nature of a partition. Partitions can be based on bare applications or OS dependent applications.

From the virtualization layer point of view, a partition has the states and transitions as shown in Figure 45.

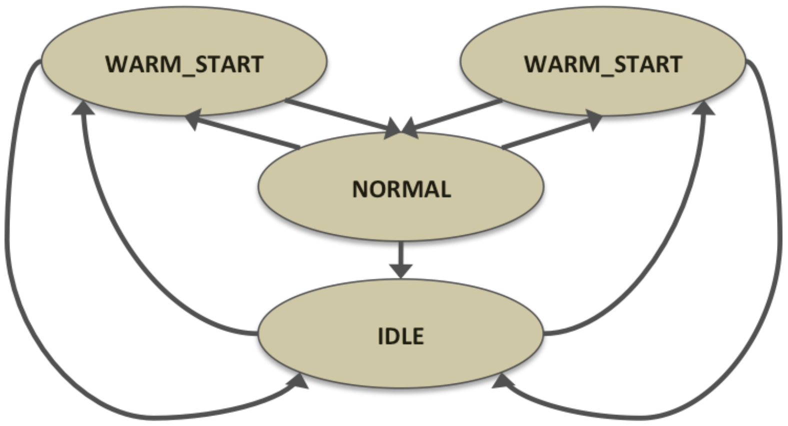


Figure 45: Partition states

After the virtualization layer initialization, each partition is loaded in memory and ready to be booted. When the resources are allocated to a partition, it boots (that is, initializes a correct stack and sets up the virtual processor control registers). From the virtualization layer, the partition is in NORMAL state.

From the virtualization layer point of view, there is no difference between the BOOT state and the NORMAL state.

The NORMAL state is subdivided in three sub-states:

- Ready : The partition is ready to execute code, but it is not scheduled because it is not in its time slot.
- Running : The partition is being executed by the processor.
- Idle : If the partition does not need to use the processor during its allocated time slot, it can yield the processor and wait for an interrupt or for the next time slot.

A partition can halt itself or be halted by a system partition. In the HALT state, the partition is not selected by the scheduler and the time slot allocated to it is left idle (it is not allocated to other partitions). All resources allocated to the partition are released. It is not possible to return to normal state.

In SUSPENDED state, a partition will not be scheduled and interrupts are not delivered. Interrupts raised while in suspended state are left pending. If the partition returns to NORMAL state, then pending interrupts are delivered to the partition.

Partition schedule

The virtualization layer schedules partitions in a fixed, cyclic basis (ARINC-653 scheduling policy). This policy ensures that one partition cannot use the processor for longer than scheduled to the detriment of the other partitions. The set of time slots allocated to each partition is defined in the configuration file during the design phase by means of a cyclic plan in a temporal interval referred as Major Frame (MAF).

Each partition is scheduled for a time slot defined as a start time and a duration. If there are several concurrent activities in the partition, the partition shall implement its own scheduling algorithm. This two-level scheduling scheme is known as hierarchical scheduling.

Multi-core schedule

The virtualization layer provides different policies that can be attached to any of the CPU. Two basic policies are defined:

1. Cyclic scheduling: Pairs <*partition, vcpu*> are scheduled in a fixed, cyclic basis (ARINC-653 scheduling policy). This policy ensures that one partition cannot use the processor for longer than scheduled to the detriment of the other partitions. The set of time slots allocated to each <*partition, vcpu*> is defined in the configuration file. Each <*partition, vcpu*> is scheduled for a time slot defined as a start time and a duration. Within a time slot, the virtualization layer allocates the system resources to the partition and virtual CPU specified.
2. Priority scheduling: Under this scheduling policy, pairs <*partition, vcpu*> are scheduled based on the partition priority. The partition priority is specified in the configuration file. Priority 0 corresponds to the highest priority. All pairs <*partition, vcpu*> in normal state (ready) allocated in the configuration file to a processor attached to this policy are executed taking into account its priority.

Multiple scheduling plans

The system can define several scheduling plans or modes. A system partition can request the change from one plan to another. Once the change is accepted, it is effective at the end of the Major Frame (MAF).

Partition offering a Virtualization Layer (TBD)

The general purpose partition running on top of the interrupt virtualization layer will offer additional virtualization features for the creation of generic partitions using KVM, which will then co-exist with the real-time partition. These new sub-partitions will be in control of the KVM hypervisor, always in accordance with the RTOS system partition, and without affecting its operation. Scheduling of the real-time partition will always take precedence over non real-time domains, as a consequence of its higher priority for interrupt processing. Scheduling of non real-time partitions will be similar to the hierarchical scheduling, as described above, except for its dynamic nature which will depend on the Host scheduling algorithm in use.

3.1 DRAL

3.1.1 System Management Services

System Management Services refer to the services that a partition can invoke to get the status of the virtualization layer or perform actions on it.

Services are:

Name	Description	Constraints
DRAL_GET_SYSTEM_STATUS	Returns the status of the virtualization layer. The result is a data structure that provides some information related to the current status. In the case of interrupt virtualization, this service will set the configuration details of such a layer, for instance, interrupt masking, peripheral binding/unbdinding, etc.	System
DRAL_SET_SYSTEM_MODE	Provides to a partition the ability to change the status of the virtualization layer. Actions to be invoked are: - Perform a cold reset on the system. As result of this invokation, the system is reset and boots. A counter informs about the number of consecutive warm resets have been produced. This counter is zeroed when the cold reset is invoked. - Perform a warm reset on the system. As result of this invokation, the system is reset and boots. The reset counter is increased. - Perform a system halt. As result of this invokation, the system is halted. A physical reset is required to restart the system.	System

3.1.2 Partition Management Services

Partition Management Services refer to the services that a partition can invoke to get its own status or other partition status or perform actions on them.

Services are:

Name	Description	Constraints
DRAL_GET_PARTITION_ID	Access to the partition identifier.	Normal
DRAL_GET_PARTITION_ID_BY_NAME	Access to the partition identifier from the partition name.	System /Normal
DRAL_GET_PARTITION_STATUS	Returns the status of a partition. The result is a data structure that provides some information related to the current partition status.	System /Normal
DRAL_SET_PARTITION_MODE	It provides to a partition the ability to change its own status or the status of other partition. Actions to be invoked are: - Perform a cold reset on a partition. As result of this invokation, the partition is reset and boots. A counter informs about the number of consecutive warm resets have been produced. This counter is zeroed when the cold reset is invoked. - Perform a warm reset on a partition. As result of this invokation, the partition is reset and boots. The reset counter is increased. - Perform a partition halt. As result of this invokation, the partition is halted. - Perform a partition suspend. As result of this invokation, the partition is suspended. - Perform a partition resume. As result of this invokation, the partition is resumed. In the case of interrupt virtualization, this service will set the configuration details of such a layer, for instance, interrupt masking, peripheral binding/unbdinding, etc.	System /Normal

3.1.3 Process Management

These services are provided by the GuestOS.

3.1.4 Time Management Services

Time Management Services refer to the services that a partition can invoke to get time information or set timers.

Time can be global or local. Global time is referred to a monotonic clock of the system. Local time is referred to a partition clock that runs when the partition is executed. Timers can be set taking as reference the global or the local time.

Services are:

Name	Description	Constraints
DRAL_GET_TIME	Get the current time (global or local).	Normal
DRAL_SET_TIMER	Set a timer referred to the global or local clock.	Normal

3.1.5 Inter-Partition Communication Services

A partition can send/receive messages to/from other partitions using sampling or queuing ports.

Services are:

Name	Description	Constraints
DRAL_CREATE_SAMPLING_PORT	Creates a sampling port.	Normal
DRAL_WRITE_SAMPLING_MESSAGE	Writes a message in a sampling port.	Normal
DRAL_READ_SAMPLING_MESSAGE	Reads a message in a sampling port.	Normal
DRAL_CREATE_QUEUING_PORT	Creates a sampling port.	Normal
DRAL_SEND_QUEUING_MESSAGE	Sends a message in a queuing port.	Normal
DRAL_RECEIVE_QUEUING_MESSAGE	Receives a message in a queuing port.	Normal
DRAL_GET_QUEUING_PORT_STATUS	Gets the status of a queuing port.	Normal
DRAL_CLEAR_QUEUING_PORT	Removes all messages in a queuing port.	Normal

3.1.6 Intra-Partition Communication

These services are provided by the GuestOS.

3.1.7 Scheduling Services

A partition is scheduled under the virtualization layer policy. It is relevant for the partition to get the information related to its own schedule. On the other hand, a partition can be interested in define local schedules for other partitions in spare slots. How to deal with spare slots and dynamic allocation of resources will be discussed in WP4.

GPOS sub-partitions created by KVM will also use these services to get scheduling policy details. In this use case the RTOS system partition will be able to force a scheduling policy on partitions that offer virtualization features (Linux/KVM partition).

Services are:

Name	Description	Constraints
DRAL_GET_PARTITION_SCHEDULE	Gets the information of the partition schedule in a MAF.	Normal
DRAL_GET_PARTITION_SCHEDULE_STATUS	Gets the information related to the current execution slot.	Normal
DRAL_SET_MODULE_SCHEDULE	Requests for a schedule plan change.	System
DRAL_GET_MODULE_SCHEDULE_STATUS	Gets the current schedule plan status. DRAL SET SPARE SCHEDULE : To be discussed	Normal
DRAL_GET_SPARE_SCHEDULE	To be discussed	Normal

3.1.8 Monitoring Services (Health Monitor)

A partition can raise health monitor (HM) events to the virtualization layer. These HM events are detected and generated by the application or the partition runtime. The events that the partition can raise are:

- APPLICATION ERROR: An error in the application.
- DEADLINE MISSED: A deadline miss has been detected.
- NUMERIC ERROR: The application has detected a numeric error.
- STACK OVERFLOW: The partition detects a stack overflow.
- MEMORY VIOLATION: The partition detects an illegal memory access.

Services are:

Name	Description	Constraints
DRAL_GET_ERROR_STATUS	Permits to the partition to access to the reported errors.	Normal
DRAL_RAISE_APPLICATION_ERROR	The partition raises an HM event that will be handled by the virtualization layer	Normal

3.1.9 Configuration services

The following table summarizes what constitutes configurations services, i.e. all services that allow for reconfiguration of the system:

Name	Description	Constraints
DRAL_SET_MODULE_SCHEDULE	Requests for a schedule plan change.	System
DRAL_SET_PARTITION_MODE	It provides to a partition the ability to change its own status or the status of other partition. Actions to be invoked are: - Perform a cold reset on a partition. As result of this invokation, the partition is reset and boots. A counter informs about the number of consecutive warm resets have been produced. This counter is zeroed when the cold reset is invoked. - Perform a warm reset on a partition. As result of this invokation, the partition is reset and boots. The reset counter is increased. - Perform a partition halt. As result of this invokation, the partition is halted. - Perform a partition suspend. As result of this invokation, the partition is suspended. - Perform a partition resume. As result of this invokation, the partition is resumed.	System /Normal
DRAL_SET_SYSTEM_MODE	Provides to a partition the ability to change the status of the virtualization layer. Actions to be invoked are: - Perform a cold reset on the system. As result of this invokation, the system is reset and boots. A counter informs about the number of consecutive warm resets have been produced. This counter is zeroed when the cold reset is invoked. - Perform a warm reset on the system. As result of this invokation, the system is reset and boots. The reset counter is increased. - Perform a system halt. As result of this invokation, the system is halted. A physical reset is required to restart the system.	System

Group of Execution Security Services

In the following, the security services for the end-to-end communication on application level are described. Hence, there is a secure communication from one application to another application. The secure communication from one application to another application includes all parts in the communication between the application like on-chip communication as well as off-chip communication.

3.2 Security Services for End-to-End Communication

3.2.1 Encryption Service

The encryption service encrypts data with a given cryptographic key. It transforms a plaintext into a cipher text so that the un-intended recipients cannot understand the messages exchanged between two legitimate communication partners. The encryption service for end-to-end communication is used for a confidential communication between two applications. Even the system components between the two applications, e.g., gateways and routers, cannot interpret the content of the communication.

3.2.2 Decryption Service

The decryption service decrypts data with a given cryptographic key. It transforms a cipher text into plain text, if the key is correct and there was no transmission error. The decryption service for end-to-end communication is used for a confidential communication between two applications. The adversaries and the unintended recipients, such as the gateways and the routers cannot interpret the exchanged messages because they do not possess the key to decrypt the exchanged messages. Only the legitimate communication partners, owning the cryptographic key, can decrypt the exchanged data.

3.2.3 Integrity Service

The integrity service generates a cryptographic hash (or secure checksum) for a message, which is transmitted together with the message. With this checksum, any modifications in the message are detectable. The integrity service for end-to-end communication ensures that all changes are noticeable and that not only the changes during the off-chip communication are detectable. For example, this service can be used by the monitoring and resource scheduling components (GRM, LRM, LRS and MON) to ensure the integrity of the communication.

3.2.4 Integrity Check Service

The integrity check service verifies the integrity of a message by re-calculating the cryptographic hash (or secure checksum) on the received message and comparing it with the received checksum. With this checksum, even a single bit modification is detectable. The integrity check service for end-to-end communication ensures that all changes are noticeable and that not only the changes during the off-chip communication are detectable. For example, this service can be used by the monitoring and resource scheduling components (GRM, LRM, LRS and MON) to check the integrity of the communication.

3.2.5 Authentication Code Generation Service

The authentication code generation service generates a message authentication code (MAC) tag or digital signatures for ensuring the data origin respectively to verify the communication partner. This service generates the MAC tag or the digital signatures on the application layer. This implies that the service can be used by the monitoring and resource scheduling components (GRM, LRM, LRS and MON) to ensure the authenticity of the communication.

3.2.6 Authentication Code Verification Service

The authentication code verification service verifies the data origin or the communication partner by verifying the message authentication code (MAC) tag or the digital signatures received with the message. This service verifies the authentication tag or the digital signatures on the application layer. This implies that this Service can be used by the monitoring and resource scheduling components (GRM, LRM, LRS and MON) to verify the authenticity of the communication.

3.2.7 Access Control Service

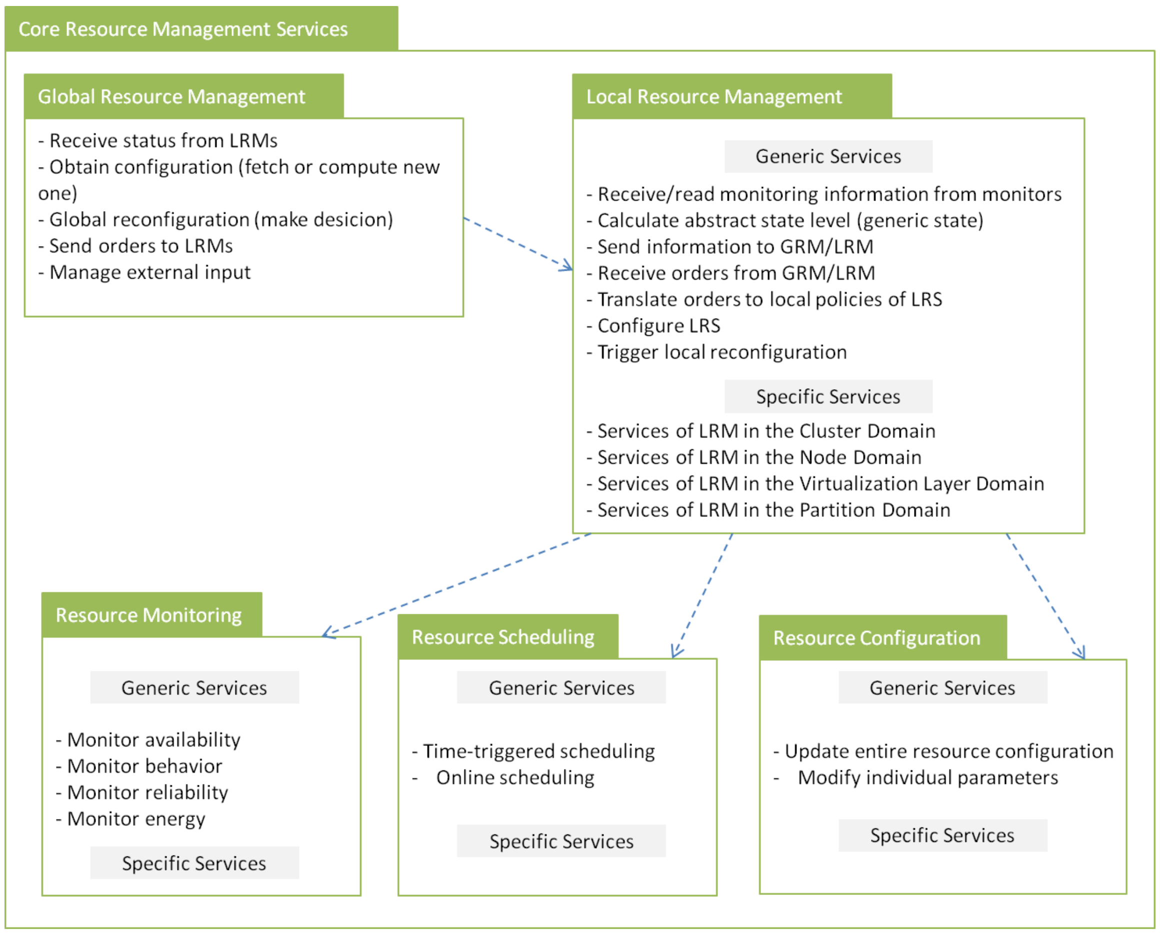
The access control service verifies if a system resource is allowed to access the requested object. For end-to-end communication it checks the permission on application layer for access to secure memory. Either the access control service or secure storage service (or both of them together) will ensure the concept of secure memory storage.

4 Core Platform Services - Resource Management

This section provides information on the Resource Management category of the Core Platform Services in DREAMS. The next two unnumbered sections present, first, the classification of the resource management services and the building blocks that provide them, and second, the resource management architecture adopted in DREAMS. The following numbered sections correspond to the sub-categories of resource management services, and within them, the specific services are described.

Groups of services and building blocks

The integrated resource management architecture in DREAMS provides the Core Platform Resource Management Services. This category of Core Platform Services is divided into four groups of services: *Global resource management services*, *Local resource management services*, *Monitoring services*, *Scheduling services* and *Configuration services*.



Core Resource Management Services
Global Resource Management
- Receive status from LRMs
- Obtain configuration (fetch or compute new one)
- Global reconfiguration (make desicion)
- Send orders to LRMs
- Manage external input
Local Resource Management
Generic Services
- Receive/read monitoring information from monitors
- Calculate abstract state level (generic state)
- Send information to GRM/LRM
- Receive orders from GRM/LRM
- Translate orders to local policies of LRS
- Configure LRS
- Trigger local reconfiguration
Specific Services
- Services of LRM in the Cluster Domain
- Services of LRM in the Node Domain
- Services of LRM in the Virtualization Layer Domain
- Services of LRM in the Partition Domain
Resource Monitoring
Generic Services
- Monitor availability
- Monitor behavior
- Monitor reliability
- Monitor energy
Specific Services
Resource Scheduling
Generic Services
- Time-triggered scheduling
- Online scheduling
Specific Services
Resource Configuration
Generic Services
- Update entire resource configuration
- Modify individual parameters
Specific Services

Figure 46: Groups of core platform resource management services

There are four types of building blocks that interact with each other, and together they provide the aforementioned services in DREAMS. Those building blocks are: the Global Resource Manager (GRM), the Local Resource Managers (LRM), the Resource Monitors (MON) and the Local Resource Schedulers (LRS). Each of the building blocks provides different services depending on the type of resource to which they correspond:

- **Resource Monitors (MON)** provide *monitoring services* (e.g. monitor availability, energy, detect errors).
- **Local Resource Schedulers (LRS)** provide *scheduling services* and *configuration services*. Examples of *scheduling services* are dispatching of time-triggered messages, schedule tasks according to offline tables or online scheduling parameters, etc. The *configuration services* refer to the ability of an LRS to accept requests for executing changes and updates on its own configuration.
- **Local Resource Managers (LRM)** provide *local resource management services*, which comprise activities such as translating monitored information into abstract state levels (e.g. error counts may be associated to a certain reliability level), sending abstract state of resources to the GRM, receiving reconfiguration orders from GRM, adapting orders from GRM into specific scheduling parameters for LRS and initiating local reconfigurations on its own.
- **Global Resource Manager (GRM)** provides *global resource management services*. It performs global decisions based on the information received from LRMs, it obtains new configurations by selecting them from an offline-computed set of configurations or by computing new ones online and it sends reconfiguration orders to the LRMs. It also manages an external input to manually trigger a system-wide reconfiguration.

In order to guarantee that the resource management components have a correct view of the system, these services are not intended to be used at the application level, or by any component that is alien to the resource management architecture. **This means that only resource management building blocks can communicate with each other.** MONs will track changes in the resources, of receive status updates from them, but will only accept status requests from LRMs. Likewise, LRSs will schedule resources, and they can only receive orders from LRMs.

In some cases, the local *monitoring* and *scheduling services* for a specific resource may actually be implemented within a single component, e.g. the LRS of the On-Chip Network Interface (Section1.1). However, from the resource management perspective, those services are provided by two different types of entities: Resource Monitors (MON) and Local Resource Schedulers (LRS).

This chapter presents resource management services for resources that provide the DREAMS core services. It leaves out resource management services that may be implemented for specific application components (e.g. local schedulers of a Guest OS, local monitor of an application hardware accelerator). Furthermore, in this section, a set of **generic** *monitoring* and *configuration services* (provided by MON and LRS) is presented. The *monitoring*, *scheduling* and *configuration services* of specific type of resources are covered in the corresponding section inside the other core services to which they belong. For example, the reconfiguration and monitoring services of the Off-Chip Communication Network Interface are presented in section 1.3, under the Communication Core Services. The *local resource management services* (LRM) at all levels and the *global resource management services* (GRM) are also covered in the following sub-sections.

Resource management architecture

The four types of resource management building blocks (GRM, LRM, MON, LRS) can be arranged across the DREAMS platform in many different configurations. We take a look at two main classes: a flat architecture, shown in Figure 47, and a hierarchical architecture, shown in Figure 48.

The flat architecture in Figure 47 consists of a GRM at the top of the hierarchy, which directly supervises and controls a set of LRMs and has a complete view of the system. All LRMs stand all at the same level. Each of them manages one resource, together with a pair of MON and LRS. The resources in Figure 47 are hardware resources and they can be processor cores or clusters, memories, I/O components, hardware accelerators, among others. In this scheme, each LRM directly communicates with the GRM, with disregard for where the resource is located in the system, i.e. inside which node (chip) or off-chip cluster.

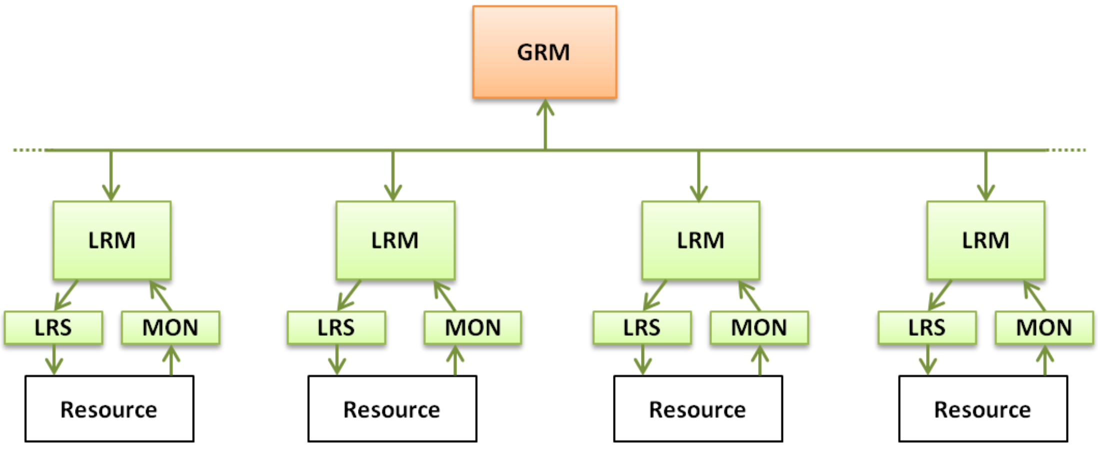


Figure 47: Flat resource management architecture

An important disadvantage of this structure is that it cannot cope with granularity issues, especially from a timing perspective. Different resources realize their activities at considerable different speeds. When all LRMs are treated equally by the GRM, it is not possible to take that fact into account. Furthermore, there can be faults that require a reconfiguration of only a subset of resources, e.g. all resources inside a single node (chip). In a flat architecture, such faults and the subsequent reconfiguration can only be addressed by the GRM, the only component with a system-wide view.

Figure 48 presents a hierarchical resource management architecture. It consists of a GRM at the top of the hierarchy and a set of LRMs, some of them standing at different levels or domains (represented by the horizontal lines). The GRM directly communicates with the LRMs at the second to highest level, while those communicate with LRMs at a lower level and LRS+MON pairs. Each LRM communicating to another LRM introduces a new level in the architecture. This structure allows the LRMs to act as a granularity interface, which hides fine-grained activities of a sub-system from the GRM view, only to communicate relevant information when global reconfiguration may be necessary, e.g. when local reconfiguration is not enough to deal with the fault. From the temporal perspective, local reconfiguration of a sub-system can be initiated by an LRM much sooner without the need to wait for the communication with the GRM. Faults could be temporarily mitigated while waiting for instructions by the GRM to implement a more permanent solution.

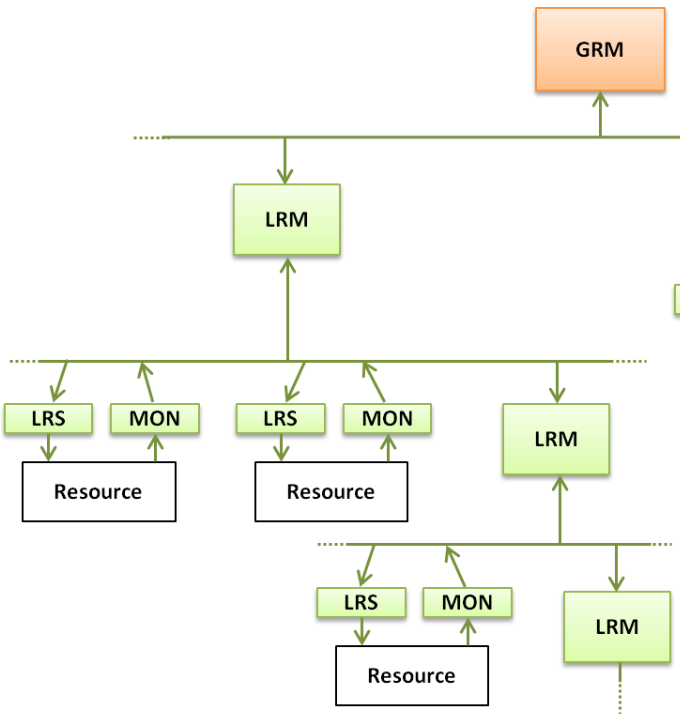


Figure 48: Hierarchical resource management architecture

On the one hand, a non-hierarchical architecture for resource management in DREAMS is possible and it is a simple solution. It is inflexible and not scalable, as having the LRMs of each resource communication with the GRM will become infeasible soon, as the number of resources increases. On the other hand, a hierarchical architecture is more complex and provides a lot more flexibility. For some implementations of the platform, the conceptually hierarchical structure could be limited to certain specific levels, whenever required.

In this deliverable, we aim at describing a generic hierarchical resource management architecture that is flexible enough to accommodate the heterogeneous resources of the DREAMS platform, while providing the basic services for adaptation and reconfiguration.

In order to establish the hierarchy in the resource management architecture in the DREAMS platform, we introduce the concept of the ***resource management domains***. We consider five different domains in a DREAMS platform, for the purposes of performing resource management: System Domain, Cluster Domain, Node Domain, Virtualization Layer Domain and Partition Domain. The domains represent the composition of the system from the resource management perspective.

Conceptually, the GRM controls the resources in the System domain, and corresponds to the highest level of hierarchy of resource management components. **For each of the other domains there shall be an LRM block in charge of supervising and controlling the resources of its corresponding domain**. Such resources could be controlled indirectly, through communication with a lower-level LRM, or directly by communication with monitors and LRS of the individual resource itself. Figure 49 presents the composition of the system in terms of domains. It also depicts the GRM and LRMs in the system and the scope of their actions. We can differentiate between four types of LRMs: *LRM in the Cluster domain*, *LRM in the Node domain*, *LRM in the Virtualization Layer domain* and *LRM in the Partition domain*. The services provided by each of them are detailed in the following subsections. It is important to note that Figure 49 does not intent to show where each GRM/LRM is physically implemented or where it executes. Instead, it presents an abstract view of the resource management hierarchy.

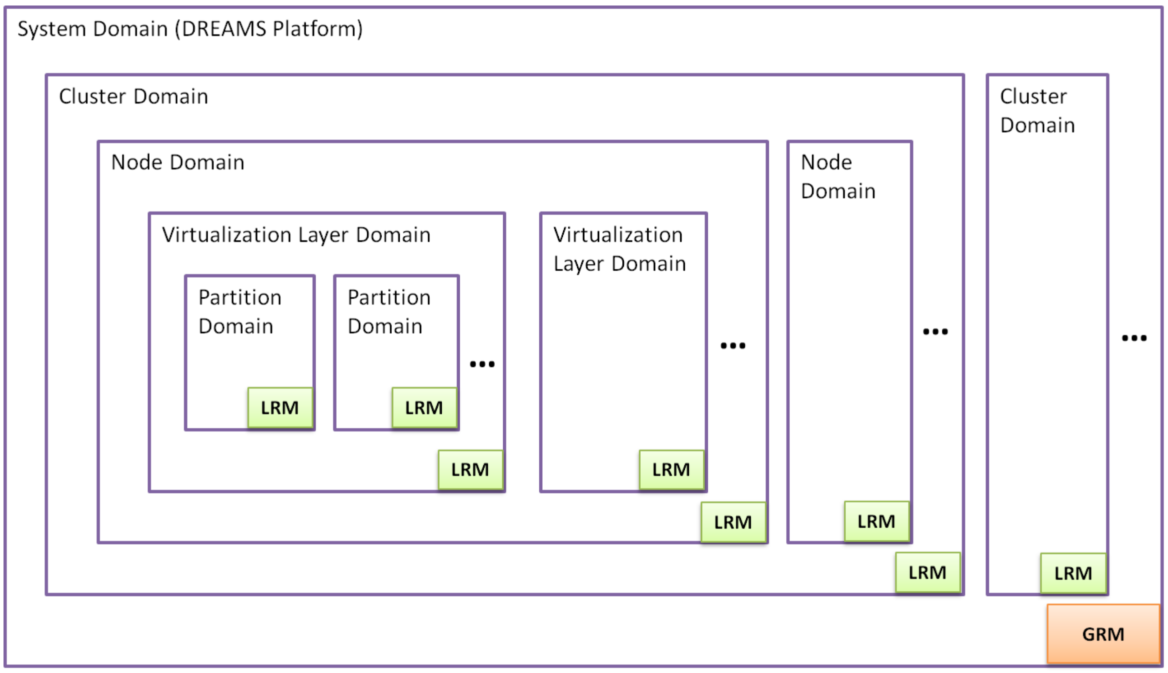


Figure 49: Resource management domains

In general, all resource management building blocks can have hardware or software implementations, or a combination of both, depending on the domain and type of resource. The actual physical implementation of the local building blocks will be discussed in WP2 and presented in deliverable D2.2.2 (Report on monitoring, local resource scheduling and reconfiguration services for mixed-criticality and security with implementation of low- and high-level monitors, scheduling, security and reconfiguration services supporting mixed criticality and adaptation), while the actual physical implementation of the GRM is to be discussed in WP3 and covered progressively in D3.2.1, D3.2.2 and D3.2.3 (High-level design, first implementation and final implementation of Global resource management services, respectively).

Figure 50 presents an example of where the resource management building blocks can be physically implemented in the DREAMS platform, in the node and virtualization layer domains.

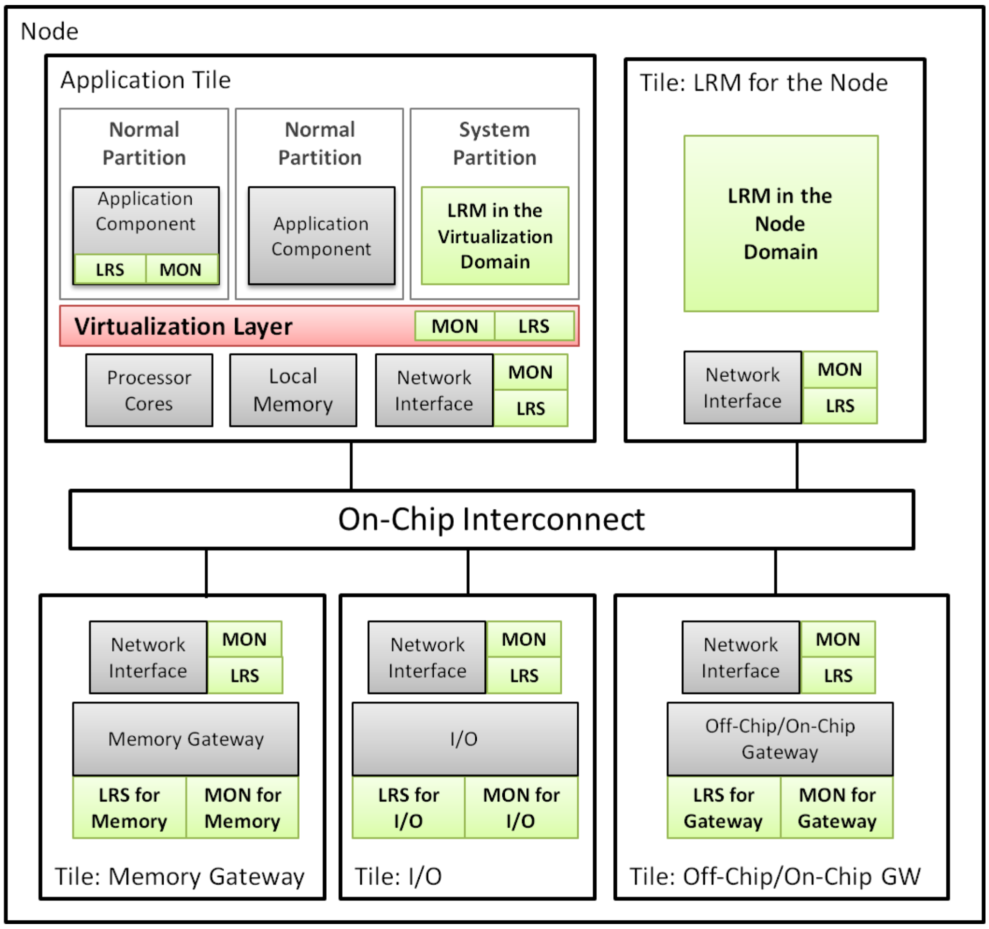


Figure 50: Example of resource management building blocks (in green) layout in the node and virtualization layer domain (resources in grey)

The LRM in the Node domain can be implemented in hardware as an IP core, or in software in a dedicated processor core. This LRM is in charge of supervising and controlling the LRMs in the virtualization layer(s) in the node and all other **non-virtualized resources**. That includes the LRS+MON of the network interfaces, the LRS+MON of the memory gateways or on-chip memories, the LRS+MON of the I/O components and the LRS+MON of the off-chip/on-chip gateway.

The LRM in the Virtualization Layer domain can be implemented in a system partition (see part II, section 3). This LRM would be in charge of supervising and controlling the LRSs and MONs in the virtualization layer domain. That includes the LRS+MON inside the hypervisor (Partition Scheduler and Health Monitor, see Part II, section 3), LRS+MON of the network interface and LRS+MON of application components (e.g. scheduler of guest OS, application monitors).

Figure 51 presents an example of where the resource management building blocks can be physically implemented in the DREAMS platform, in the system and cluster domains.

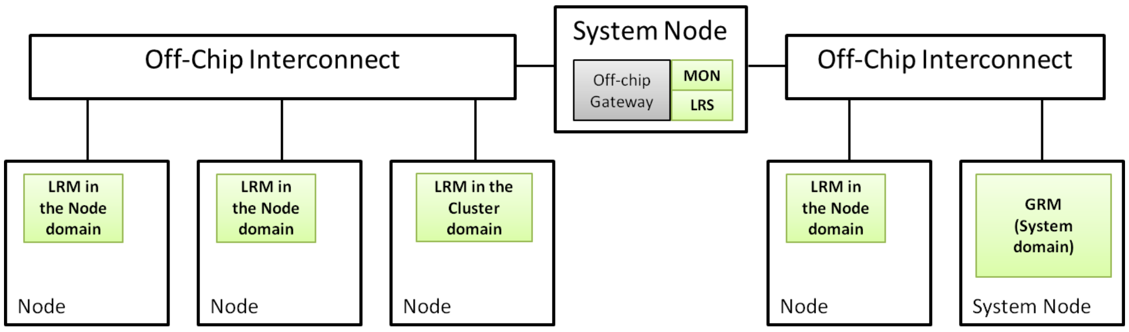


Figure 51: Example of resource management building blocks (in green) layout in the system and cluster domain (resources in grey)

The LRMs in the cluster domain can be implemented in a dedicated node or it can be positioned in a regular node. These LRMs are in charge of supervision and controlling the LRMs in the node domain, the LRS+MON of the off-chip gateways and the LRS+MON of the off-chip switches (not depicted in the figure). The GRM in the system domain can be implemented in a system node. The GRM is in charge of supervising and controlling the LRMs in the cluster domain and the LRMs in the node domain that do not have a cluster-level LRM on top in the hierarchy.

Group of Global Resource Management Services

In the following, the group of global resource management services is described.

4.1 Global Resource Management Services

In this section, we present the services that will be provided by the Global Resource Manager (GRM).

4.1.1 Gather status from LRMs

The GRM collects all monitored information from the LRMs with whom it communicates. The main parameters to be transferred are the physical name of the resource, the name of the monitored variable and its value.

4.1.2 Obtain configuration (fetch or compute new one)

The GRM is in charge of the data base of all off-line precomputed configuration of resources. Such configurations can be stored in a distributed or in a centralized way. Alternatively, the GRM could also compute new configurations (i.e. determine new scheduling tables or scheduling parameters) at runtime.

4.1.3 Global reconfiguration (make decision)

The GRM will analyse the monitored information at the system level and take reconfiguration decisisons that allow the system to adapt to different modes or conditions. The GRM takes into account information from all types of resources to make a decision that considers system-wide constraints.

4.1.4 Send orders to LRMs

Once a reconfiguration decision has been taken, the GRM will communicate it to the LRMs involved in the reconfiguration, via network and middleware, taking into account all types of resources. For example, if a change in the scheduling plans of an application tile is required, the GRM will provide a new scheduling table for the virtualization layer (or processor cores), as well as for the network interfaces of the application tiles involved, because reconfiguration of the network is expected. Orders could be given in the form of a simple reference to the actual configuration, or the GRM could transmit the complete configuration via the network.

4.1.5 Manage external input

The GRM will manage an external input that can trigger a global reconfiguration. Such input could be given locally (I/O peripheral directly connected to the GRM node) or remotely (via off-system Ethernet). This input could be a new constraint to the system, and the GRM would obtain a new configuration that satisfies the constraint, or it could be an absolute reconfiguration decision which the GRM could simply communication to the LRMs.

Group of Local Resource Management Services

In the following, the group of local resource management services is described. Inside this group, there are two main sub-categories: generics services (section 4.2) and specific services (sections 4.3, 4.4, 4.5 and 4.6).

4.2 Generic LRM Services

4.2.1 Receive/read monitoring information from monitors

Two communication paradigms are possible: interrupt and polling. In the first case, the MONs send information periodically to the LRM. In the second case, the LRM requests information from the MONs. A combination of the two approached is also possible.

4.2.2 Calculate abstract state level (generic state)

The LRMs are in charge of calculating an abstract level of the state variables of the resource, based on monitored information from the MONs. Abstract state variables can be energy, availability, reliability, behavior, among others. The level of abstraction depends on the specific resources and available monitors. This approach is based on providing a resource view on an abstract level, to reduce the overhead of disseminating the low-level monitor variables and only provide information requiring a system-wide reconfiguration.

4.2.3 Send information to GRM/LRM

Each LRM will transmit the abstract state of the resources in its domain, via the network and middleware, to the next LRM in the hierarchy, or to the GRM if it stands at the second to last level.

4.2.4 Receive orders from GRM/LRM

The LRMs can only receive orders from other LRMs or the GRM, never from application components or other system components.

4.2.5 Translate orders to local policies of LRS

This service is provided by the LRMs. After receiving an order from the GRM, the LRM maps it to the local scheduling policies of the LRS of the resource. In the case of LRSs that implement online scheduling of the resource, the LRM can provide the scheduling parameters to the corresponding LRMs. In the case of table-based scheduling policies, the LRM can provide the table itself, or a reference to it. This approach is based on the conceptual separation between implementation details of the scheduler of a resource, and the abstract view of the component that is keept by the GRM.

4.2.6 Configure LRS

The LRM configures the LRSs in its domain, and it gives orders to the LRMs at a lower level in the hierarchy.

4.2.7 Trigger local reconfiguration

Small changes in the state of a resource can be handled locally by the LRM in its domain. For example, suspension of a low-criticality partition that only communicates with other partitions in its own application tile, can be requested by the LRM of that application tile, as no reconfiguration of resources outside of the tile is necessary. In that case, the LRM will report the new state of the resource (the application tile) to the GRM to maintain coherence in the state of the system.

4.3 LRM Services in the Cluster Domain

The LRM in the cluster domain serves as a granularity interface between the GRM and two different types of off-chip networks. It would typically not handle individual resources, but would only communicate to configure and monitor LRMs in the smaller domain, i.e. LRMs in the Node domain. An LRM in the cluster domain could be physically implemented in one of the existing nodes in the system. For reasons of practicality and reduction of communication and costs overhead, we could exclude the LRM in the Node domain for that specific node, given that there would already be a cluster level LRM in place. In that case, the LRM in the Cluster domain would indeed handle individual resources, e.g. memory gateways IOMMU, cores or LRMs in the Virtualization Layer domain.

These are specific services provided by the LRM in the cluster domain.

4.3.1 Monitor lower-level LRM

This services refers to the capability of each LRM to receive or read monitoring information previously gathered by a lower-level LRM, i.e. by the LRM in a smaller domain.

4.3.2 Configure lower-level LRM

This services refers to the capability of each LRM to configure a lower-level LRM, i.e. an LRM in a smaller domain, given the directions and orders received from the GRM, or higher-level LRM.

4.4 LRM Services in the Node Domain

An LRM in the Node domain have to deal with LRMs in the immediately smaller domain, i.e. the virtualization Layer domain, as well as with individual resources, like for example, an IOMMU or a non-virtualized memory gateway.

Typically, the LRM in the Virtualization Layer domain would have the following specific service:

4.4.1 Monitor lower-level LRM

This services refers to the capability of each LRM to receive or read monitoring information previously gathered by a lower-level LRM, i.e. by the LRM in a smaller domain.

4.4.2 Configure lower-level LRM

This services refers to the capability of each LRM to configure a lower-level LRM, i.e. an LRM in a smaller domain, given the directions and orders received from the GRM, or higher-level LRM.

4.5 LRM Services in the Virtualization Layer Domain

As explained before, the LRM in the Virtualization Layer Domain, could be implemented in a system partition, which would grant the permissions necessary to configure the system (perform a reset, halt another partition), and change the scheduling plan at runtime. We regard this scheduling plan to be managed by the LRS in the Virtualization Layer domain, i.e. the partitions scheduler. We distinguish this from the intra-partition schedulers, i.e. schedulers of a guest OS inside a partition, which we regard as the LRS in the Partition domain.

Typically, the LRM in the Virtualization Layer domain would have the following specific service:

4.5.1 Monitor lower-level LRM

This services refers to the capability of each LRM to receive or read monitoring information previously gathered by a lower-level LRM, i.e. by the LRM in a smaller domain.

4.5.2 Configure lower-level LRM

This service refers to the capability of each LRM to configure a lower-level LRM, i.e. an LRM in a smaller domain, given the directions and orders received from the GRM, or higher-level LRM.

4.6 LRM Services in the Partition Domain

The LRMs in the Partition domain are to be provided by the partition developer. An example of this type of LRM is a management component inside a partition with a guest OS. The guest OS would typically have an inside scheduler, which would require parameters to be configured, i.e. a DREAMS LRS. It would also have monitoring capabilities that would allow the LRM in the partition to gather the health state of the guest OS and its tasks.

Group of Resource Monitoring Services

Each resource that should be managed by the integrated DREAMS resource management architecture is paired with a set of LRS and MON. Each MON monitors the status variables that are pertinent for that type of resource. In the following, the group of resource monitoring services is described. Inside this group, we present one sub-category: generics services (section 4.7). Specific services for resource monitoring are presented in section 1 for communication resources (network interfaces, routers, gateways, memories) and section 4 for execution resources (partitions).

4.7 Generic Resource Monitoring Services

4.7.1 Monitor availability

These services are provided by MONs. Several distinct state variables of a resource can be associated with resource availability. Some examples are: processor core or memory utilization, operational status of the resource (correct/faulty).

4.7.2 Monitor behavior

These services are provided by MONs. Several distinct state variables of a resource can be used to monitor behavior. Some examples are: status of the resource (e.g. number of waiting messages in a queue, number of aperiodic tasks in the ready queue) and application-level monitoring (e.g. QoS of applications, stability of control).

4.7.3 Monitor reliability

These services are provided by MONs. Several distinct state variables of a resource can be used to monitor reliably. In particular, errors are closely linked with the reliability of the operation of a resource. We can distinguish between temporal errors (e.g. violations of period and arrival time of messages, deadline miss of a task) and errors in value (e.g. errors in the body of a message, errors in results of computation at the application-level, corrupt memory space).

4.7.4 Monitor energy

All resources in the system in all resource management domains are subject be monitored with respect to energy. If the energy consumption of a resource can be measured at runtime, or the resource is characterized by power (i.e. power models are available) offline for different modes of operation, the MON can use this information to let the LRM know about the energy status of the resource. This service depends on the availability of such power models.

4.8 Specific Resource Monitoring Services

See section 1 for monitoring services offered by communication resources and section 4 for monitoring services offered by the virtualization layer in DREAMS.

Group of Resource Scheduling Services

4.9 Specific Resource Scheduling Services

Each resource that should be managed by the integrated DREAMS resource management architecture is paired with a set of LRS and MON. Each LRS schedules the use of the resource. The services that it provides and its own implementation are very specific and particular to each resource, and are presented in Section 1 for communication resources (network interfaces, routers, gateways, memories) and Section 3 for execution resources (partitions).

In general, LRSs perform the runtime scheduling of resource requests, for example, execution of partitions on top of the virtualization layer, execution of tasks inside a partition, processing of queued memory and I/O requests, dispatching of time-triggered and aperiodic messages at the network interfaces. In general, the LRSs in DREAMS support different scheduling policies, which can be classified at a high level as offline scheduling and online scheduling.

Group of Resource Configuration Services

In the following, the group of resource configuration services is described. Inside this group, we present one sub-category: generics services. Specific services for resource configuration are presented in Section 1 for communication resources (network interfaces, routers, gateways, memories) and Section 3 for execution resources (partitions).

4.10 Generic Resource Configuration Services

4.10.1 Update entire resource configuration

This service is provided by the LRS of the resource. It refers to the capability of the LRS to change its own configuration, whenever required by an LRM. Depending on the type of resource, this could mean to load a new off-line precomputed schedule, or accept changes to the online scheduling parameters.

In the case of execution resources, like virtualized processor cores, the scheduler of the virtualization layer can make a change in the scheduling plan of the partitions, as explained in section 3.1.8. Conceptually, such change can only be requested by an LRM, and practically, this change can be requested from a system partition, inside which the roll of an LRM would be implemented.

4.10.2 Modify individual parameters

Similarly to service 4.10.1, this service is provided by the LRS of the resource. It refers to the capability of the LRS to change specific parameter of its own configuration, whenever required by an LRM.

4.11 Specific Resource Configuration Services

See **Section 1** for configuration services offered by communication resources and **Section 3** for configuration services offered by the virtualization layer in DREAMS.

Group of Resource Management Security Services

4.12 Security Services

As described in the threat model for resource management services, there are two main attack targets on the resource management services. On the one hand there are attacks on the resource management components itself, and on the other hand there are attacks against the communication process of the resource management services.

The attacks against the GRM, LRM, LRS and MON components can be prevented by using access control and authentication. Only authorized users or the other authorized resource management components are allowed to change scheduling tables, configurations, etc.

The attacks focusing on the communication process of the resource management services can be prevented by using secure communication services. Depending on the type of the communication, it could use either the services on the network level or the services on the application level. On the network level, the secure communication services from the DREAMS communication services will be used and on the application level, the secure communication services from the DREAMS execution services will be used.

In addition mechanisms to provide trustworthy communication between the GRM, LRMs and MONs, the resource management services serves the key management in the DREAMS architecture. This includes key generation, destruction and exchange. To store confidential data securely, a secure storage service is provided.

4.12.1 Key Generation and Destruction Service

The key generation and destruction service generates cryptographic keys needed for secure communication and destructs (securely removes) the keys that are note longer needed. The service can generate both symmetric keys and asymmetric key pairs. Symmetric keys are used for encrypted communication. Asymmetric keys are used for the sharing of the symmetric keys or with some additional effort, they can be used to authenticate a communication partner or the origin of the data. If a cryptographic key is no longer needed by the application for which it was created, the service destructs the key which is usually stored in the secure storage.

4.12.2 Key Exchange Service

The key exchange service exchanges cryptographic keys between the communication partners. Considering the threat assumptions, this service is mainly used for the off-chip communications. The key exchange is performed in a secure way so that an adversary cannot get hold of the keys transferred through the network.

4.12.3 Secure Storage Service

The secure storage service saves important data, such as cryptographic keys, in a secured part of the memory. Applications can save confidential data in the storage and no other application can interpret the confidential data. The access to the storage is controlled by an access control list. The secure storage service can be used by the key generation and destruction service for managing the cryptographic keys of an application.

5 Optional Services

As pointed out in the definition of the DREAMS architectural style (see Part I of the document), core services are required for all domains and applications. Optional services are built on top of these core services and provide higher-level capabilities for certain domains.

Since they can be integrated only when needed, they allow for a more efficient system design. Optional services can be implemented in hardware (IP core) or software.

While the focus of this deliverable is to define the core services constituting the DREAMS architectural style, in this section, the definition of an exemplary optional service will be provided. In the following, it will be briefly motivated why a *software-based voting service* has been selected as example:

- One reason to prefer a software-based design over a dedicated hardware IP core is certainly to reduce the required effort. Additionally, the proposed design can be used to validate the interaction of a number of DREAMS core services located at different levels of the service stack (i.e., core services implemented in hardware and software). This is because the voting service depends on the execution services (see Section 3) which in turn depend on a number of further DREAMS core services (e.g., communication).
- The voting service demonstrates how the guarantees provided by the DREAMS core services can be employed to enhance the robustness of a system, i.e. its capability to withstand certain perturbations. The core services already provide assurances to prevent or contain the effects of malicious perturbations (e.g., caused by malign actions such as denial-of-service attacks, manipulation of data integrity, or side-effects of intrusion attempts) and accidental perturbations (e.g., caused by internal or external physical faults, or design faults) covered by the DREAMS fault hypothesis (see Section 2). Here, the voting services demonstrates how based on the containment property for accidental faults a building block for fault-tolerant systems (i.e., systems that to continue to operate while faults have occurred) can be designed.

5.1.1 Voting Service

Software-fault tolerance mechanisms typically require the application of adjudicators, i.e., decision mechanisms that employ redundancy to determine if the result computed by an application component is correct.

A voter is an adjudicator that compares the input of two or more replicas (possibly variants) of an application component and decides the correct result, if it exists. As such, the concept of voting can be applied at different levels of the system, and therefore can be implemented in hardware and/or software.

Figure 52 provides a general overview how a number of (non-fault-tolerant) replicas of an application component and a voter can be combined into a fault-tolerant unit (FTU).

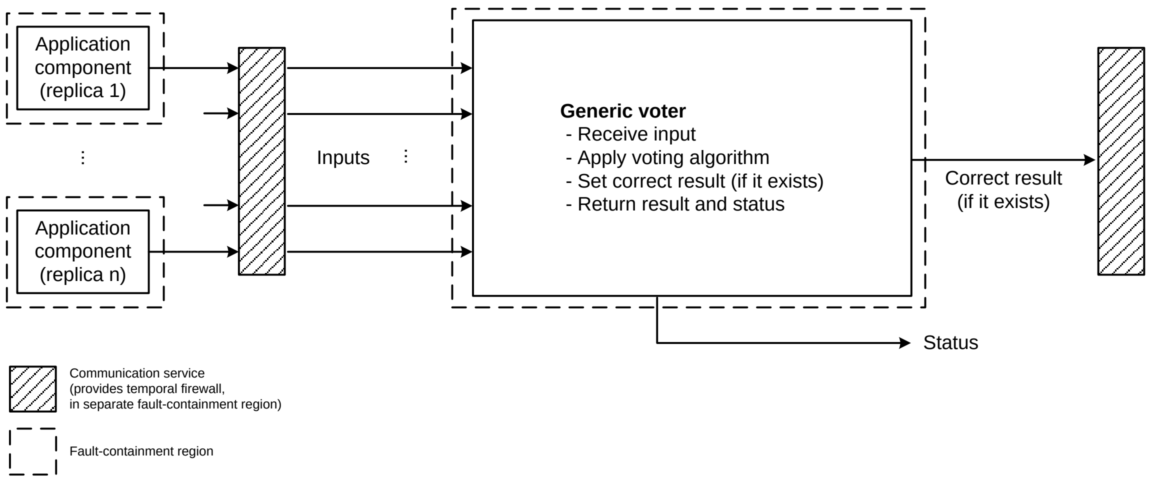

Application component (replica 1)
⋮
Application component (replica n)
Inputs
⋮
Generic voter
- Receive input
- Apply voting algorithm
- Set correct result (if it exists)
- Return result and status
Correct result (if it exists)
Status
Communication service (provides temporal firewall, in separate fault-containment region)
Fault-containment region

Figure 52: Fault-tolerant unit (FTU): replicas, communication service (with temporal firewall), and voter

In the following, assumptions and dependencies of the voting service will be discussed. In order to prevent a single fault to compromise an arbitrary number of replicas (and possibly the voter itself), each replica and the voter must reside in different *fault-containment regions* (FCRs).

FTUs exclusively based on voters can only be used to handle errors in the value domain of the expected service of the underlying application component. Since for real-time systems also temporal correctness of the FTUs *output* (i.e., its observable behaviour) must be guaranteed, the application of temporal firewalls that enforce this behaviour is mandatory. Likewise, voters in real-time systems do not only have to cope with input that is erroneous in the value domain, but also with input that violates its “timing contract”. In this case, the temporal firewall mechanism prevents the delivery of belated / early input, which makes them from the voter’s point-of-view appear as missing input. Missing input might also occur because of a fault (e.g., design fault, or lockup due to a physical fault), or a deliberate decision at originating components (e.g., failed admittance test on inputs, failed acceptance test on computed results). Hence, the DREAMS voting service implements the *dynamic versions of* the particular voting strategies (which are also defined on partial input).

Comparison function

The *comparison* of two inputs is the basic building block of any implementation of the voting service. Here, two general approaches can be distinguished:

Exact voting refers to voting strategies that are based on the procedure of performing a bit-by-bit comparison of the inputs. The advantages of this approach are that it is an efficient, scalable, strikingly simple and generic method. It maintains a strict separation of concerns between the voter and the application components that enables the universal usability of the comparison function. Since the bit-wise comparison induces a number of equivalence classes on the input data, it is typically used in the majority or the consensus voting strategies. Exact voting is based on the assumption of *replica determinism* where – in the absence of faults – all replicated components are guaranteed to produce exactly the same output messages with a bounded temporal deviation. Exact voting is not compatible with FTUs where *multiple correct results* (MCR) of the different replica are possible. Some causes for MCR such as non-deterministic algorithms may not be prevalent in the domain of safety-critical systems. However, typical design patterns such as replicated sensors can provoke the same problem. In order to ensure the required replica determinism, it necessary to reduce the redundant sensor input to one harmonized value using an agreement protocol (e.g., data fusion). As a consequence, extra care must be taken in case the replicated components use floating-point arithmetic (FPA), e.g., because of inconsistent implementations on heterogeneous architectures.

Inexact voting relies on the application and/or data-type specific comparison functions. Since this approach can be used to implement comparison functions that define orderings on the input domain, it can be used to compute the “most appropriate” voting result from varying input sets (e.g., input from replicated sensors). Inexact voters do in general not guarantee that their output is a member of the original input set (depending on the selected voting strategy). Because of the use of comparison tolerances, inexact voting is generally better suited to handle FPA. However, it might also cause additional problems. E.g., the result computed by an average voter on a set of identical floating-point values might exhibit a slight deviation from the original value.

Voting strategy

Since the DREAMS architecture ensures replica determinism for safety-critical subsystems, the DREAMS voting service implements the dynamic exact majority voting (DEMV) strategy. DEMV masks a fault if and only if a majority exists among the non-faulty inputs with respect to the size of the entire input vector. If the voter detects an agreement of all inputs, it returns the correct result and indicates a successful return status. Otherwise, i.e. if a (correct) majority exists but there is at least one deviating input source, the voter returns the correct result, and determines which of the originating application component replicas are erroneous. If no majority exists, DEMV has detected faults in the application component replicas whose detailed origin cannot be determined. Here, it will not produce a return value and assume that a potential error in all of the originating application component replicas exists. In particular, this case also occurs when less than the half of the input vector has been received on time. DEMV is defeated if a “tainted” majority exists, in which case it will deliver an incorrect result. The dimensioning of the system, i.e. the degree of redundancy and the deployment of the individual components of the FTU to separate FCRs, must ensure that this case can only occur due to a rare fault that is not covered by the systems fault hypothesis.

The modular architecture of the voting service enables the integration of further voting strategies, such as dynamic consensus voter (relaxation of majority voter that does not require an absolute majority), and inexact voters such as the dynamic average or the dynamic median voter).

Fault Assumptions and Behavior in Presence of Faults

In the presence of faults in application component replica(s), the situations described below can arise:

- If the FTU contains “enough redundancy” to tolerate all current simultaneous faults, it is able to *mask the fault(s*) in the *value domain*, and therefore exhibits *fail-operational* behaviour. Additionally, it is possible to identify the erroneous replica(s) and to possibly initiate appropriate counter-measures (*normal fault*).
- The FTU is able to detect the presence of value errors in the application component replicas, but it is not able to decide the correct result (and therefore also not to identify erroneous replicas). In this case, the FTU will not output any result at all in order to contain (value) errors. In combination with aforementioned temporal firewalls, voters can be used to guarantee *fail-silent* behaviour in this case.
- If the voter is not able to detect/correct the presence of faults in one more replicas, it is “defeated” and will forward an erroneous result. The design and dimensioning of the system must guarantee the probability of this case is sufficiently low (*rare fault).*
- In the case of a fault in the voter itself, the voter might either be defeated due to a computation error in the voter logic or it might not return any value at all (e.g., lockup because of a corruption of the program counter). In the latter case, the voter is also likely to fail to provide information about its status.

Dependency on other services

Depending on how the voter is integrated into the system (see Appendix for suggested design-patterns), the voting service is either implemented a pure software library, or as a system component in a partition (based on the voting library).

The software library is almost completely self-contained and only depends on a reporting facility in case faults have been detected (e.g., **DRAL_RAISE_APPLICATION_ERROR).**

In case the second option is seleted, the voting service is embedded into a dedicated (real-time) partition provided by the DRAL and relies on the communication service in order to send and receive periodic (multi-cast) messages (e.g., **DRAL_CREATE_SAMPLING_PORT, DRAL_WRITE_SAMPLING_MESSAGE, DRAL_READ_SAMPLING_MESSAGE**).

6 Certification Strategy

6.1 Introduction

The certification strategy builds on top of previous successful milestones (FP7 MULTIPARTES), such as the positive assessment of a safety concept for a wind power mixed-criticality embedded system ([26], [27], [28], [29]) based on a multicore partitioning solution that met IEC-61508 and ISO-13849 industrial standards. Different lessons were learnt and have been used to define current certification strategy:

- It is technically feasible to develop and certify IEC-61508 based mixed-criticality systems based on multicore and partitioning, but the effort is high
- In order to enable cost efficient certification, rules and strategies are required, e.g. diagnosis strategy.
- If modularity and variability of product families is not considered from the beginning, minor variations of the system require a complete revision and update of the safety-concept
- The usage of non-compliant items dramatically increases the safety-concept definition effort, e.g. COTS multicore processors designed for average performance

The certification strategy aims to pave the way towards the competitive development and certification of mixed-criticality solutions. Competitive development and certification emphasizes the need for solutions to manage previously described learnt lessons. For this purpose, modular safety-cases, patterns and product families are considered. IEC-61508 is considered to be the reference safety standard.

6.2 Modular Certification

The generic concept of "modular certification" is explicitly defined in different standards using different names and scopes: 'compliant item' (IEC-61508), 'Safety Element out of Context' (ISO-26262), 'Generic application' (EN-5012X), etc. In all cases, the final objective is to enable the cost competitive development of products and reduce the probability of systematic faults by means of reusability.

In DREAMS architecture we find both HW and SW components (building blocks), from which only a subset of them provides safety critical properties. Modular certification aims to reuse in an structured manner, evidences-arguments--claims of building blocks already certified, qualified or developed in compliance with safety-standards (even if they are not certified). This reuse of modular safety cases of building blocks, should reduce the effort and complexity management in the development of the mixed-criticality system, reusing.

Modular safety cases use arguments and evidences to support a given claim (e.g. the compliant item is safe for its purpose) using a graphical representation that can be documented in detail. Some arguments and evidences might need to be provided by a third party, e.g. system integrator. For this purpose the concept "export" is defined. For example, the assurance that temporal interferences that could be dangerous are either controlled or that there is diagnosis measures to go to a safe state, which shall be provided by the system integrator. An export is an assumption in the safety concept that must be provided by the system integrator. So, a sub-claim can be denominated as export item and therefore, must be provided by the system integrator.

- A claim is defined as a statement asserted within the argument that can be assessed to be true or false (e.g., X System is adequately safe). Each claim is supported by a number of sub-claims and may contain additional contextual material (i.e., explanation of used terms).
- An argument is a description of the argument approach presented in the support of a claim (e.g., deterministic, probabilistic or qualitative arguments). An argument links the evidences to the claims.
- An evidence is a reference to the evidence being presented in support of the claim or argument. This can be either facts (e.g., based on a research), assumptions, or sub-claims, derived from lower-level sub-arguments.

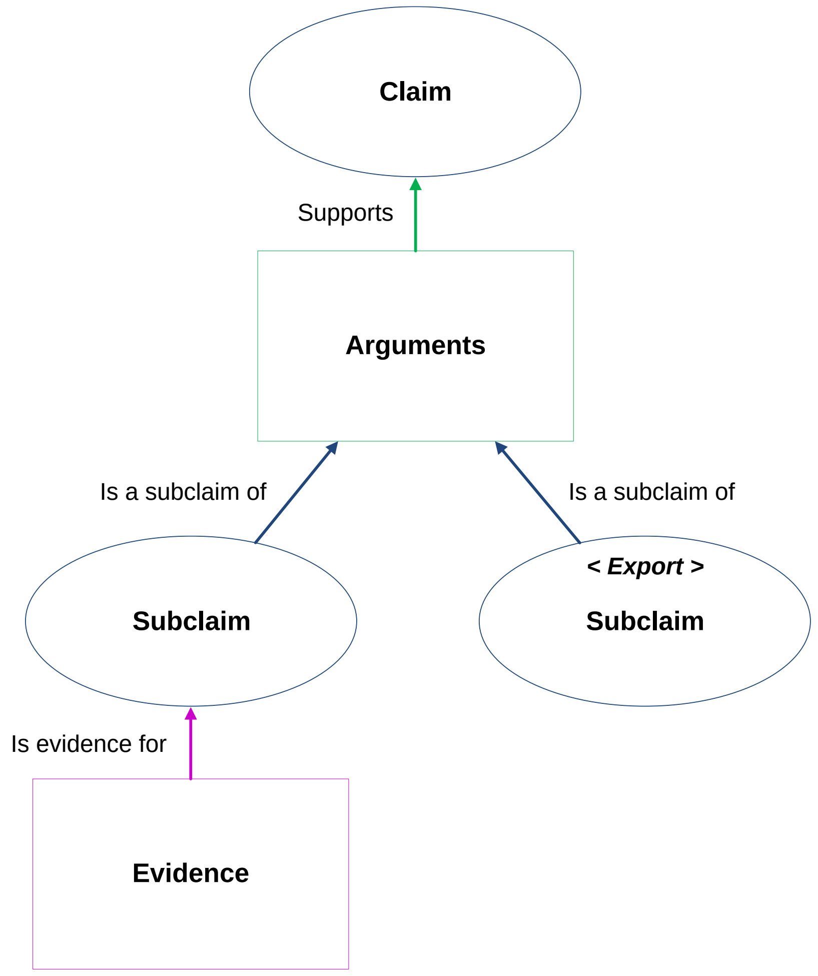


Figure 53: Claims, Arguments and Evidence (CAE) notation.

Modular Safety Cases cover the following aspects:

- Analysis of the system regarding safety needs.
- Strategies adopted to achieve the desirable SIL (Safety Integrity Level).
- Techniques and Measures to control random faults.
- Demonstration that selected techniques are sufficient to fulfil safety needs.

The sources of information to identify techniques available are:

- IEC 61508-2 Annex A.
- Other modular Safety Cases.

The architecture of the system must be well defined and preferably divided into subsystems (some of the subsystems can form Modular Safety Cases). Some of the subsystems can be:

- Input/Output Modules
- Memory Units
- On/Off chip communications
- Safety/Non-Safety interactions
- Etc.

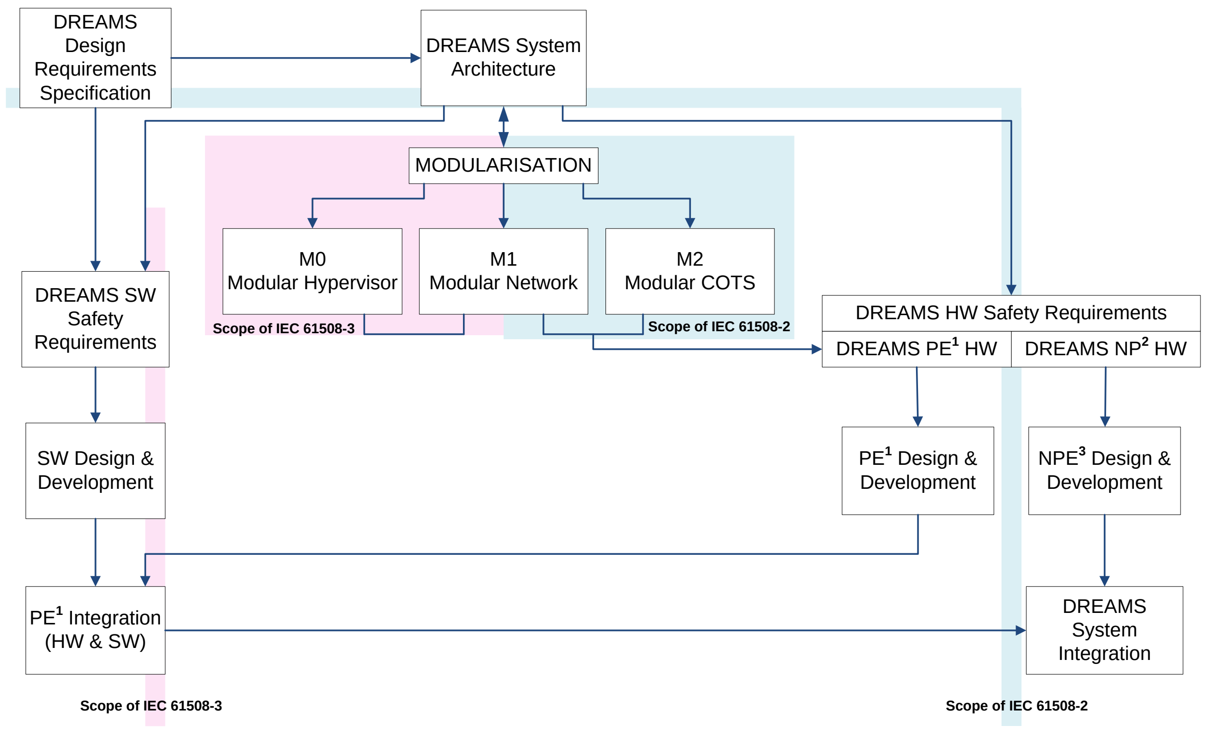


Figure 54 Relationship and scope for IEC 61508-2 / IEC 61508-3 and Modular Safety Cases

[1] **PE:** Programmable Electronic.
[2] **NP:** No Programmable.
[3] **NPE:** No Programmable Electronic.

6.2.1 Examples of Modular Certification

For example, a modular certification of the hypervisor means that the hypervisor itself must be compliant item and the safety partitions generated by the hypervisor can be considered by IEC 61508 compliant items (Annex D IEC 61508-3). In case of partitions that are not related to safety functions, they do not need to be a compliant item. The hypervisor shall provide the non-interference between partitions following IEC 61508-3 Annex F for assure interference freeness of partitions.

The hypervisor as a modular certified unit must ensure the independence of execution between the software elements hosted in the DREAMS chip. DREAMS can host elements of different systematic capability, or software contributing to safety and non-safety functions, etc. Those software elements will not adversely interfere with each other´s behaviour such that a dangerous failure can occur. The hypervisor should demonstrate the independence of execution in both the temporal and spatial domains.

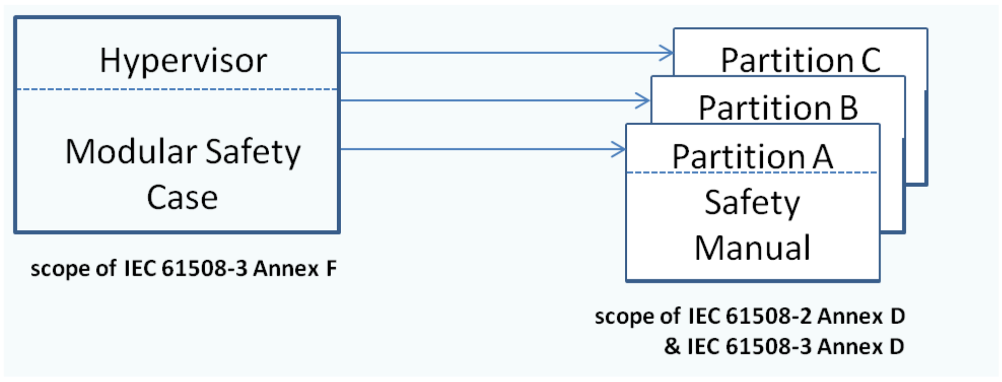

Hypervisor
Modular Safety Case
scope of IEC 61508-3 Annex F
Partition C
Partition B
Partition A
Safety Manual
scope of IEC 61508-2 Annex D & IEC 61508-3 Annex D

Figure 55: Relationship among hypervisor, partitions and IEC 61508.

In case that one safety partition, which is considered as compliant item, is intended to be reused in one or more instantiations of the DREAMS chip, it will need to follow IEC 61508-2 Annex D and IEC 61508-3 Annex D. These annexes define that each safety partition shall provide a Safety Manual that contains all the information relating to compliant item, which is required to enable the integration of the compliant item into a safety-related system, or a subsystem or element, in compliance with the requirements of IEC 61508. The safety manual should describe attributes, functions, constraints, etc. to be taken into account by the integrator and on top of all the evidence for the future assessment of the instantiation.

For the case of mixed criticality networks, DREAMS architecture system involves the transfer of information between different locations. The transmission system forms an integral part of the safety related system, which must be protected to guarantee the end-to-end communication integrity. The integrity of end-to-end channel can be ensured by checking correctness of messages between applications, so the end-to-end communication can be considered as safe.

According to IEC 61508-2, there exist two transmission systems:

- **White Channel:** The entire channel is designed, implemented and validated according to IEC 61508 and IEC 61784-3 or IEC 62280 series.
- **Black Channel:** The channel is not designed, implemented or validated according to IEC 61508, and measures shall be implemented in the E/E/PE safety-related subsystems or elements that interface with the communication channel in accordance with IEC 61784-3 or IEC 62280. Measures are typically a safety related protocol which is put on top of the existing non-safety related communications and which includes measures against fault such as describes in IEC 61784-3 and IEC 62280.

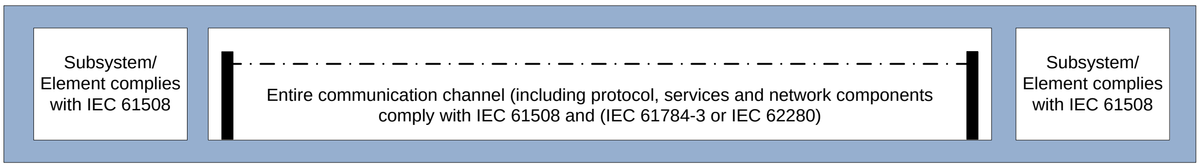

Subsystem/ Element complies with IEC 61508
Entire communication channel (including protocol, services and network components comply with IEC 61508 and (IEC 61784-3 or IEC 62280)
Subsystem/ Element complies with IEC 61508

Figure 56: White Channel Architecture for Mixed-Criticality Data Communication.

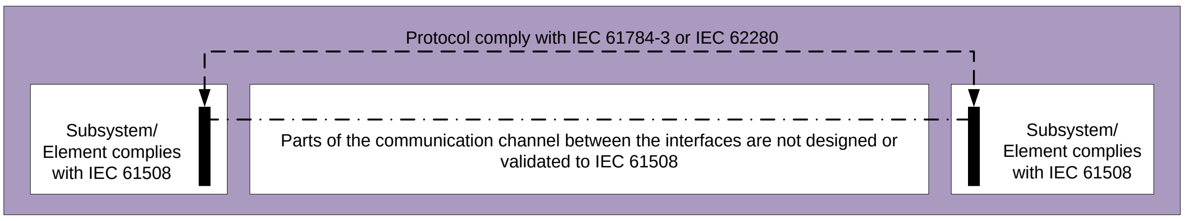

Protocol comply with IEC 61784-3 or IEC 62280
Subsystem/ Element complies with IEC 61508
Parts of the communication channel between the interfaces are not designed or validated to IEC 61508
Subsystem/ Element complies with IEC 61508

Figure 57: Black Channel Architecture for Mixed Criticality Data Communication.

The modularity approach makes potentially possible that a modular safety case for mixed criticality networks can contain a Safety Communication Layer that can be reused as a compliant item for other developments.

The instantiations of Safety Communication Layer can vary depending on the selection of transmission system type (open or close). In consequence, as Figure 36 shows, it is potentially possible to achieve a modular SCL which can be used as independent and compliant for easier and more flexible development of new systems.

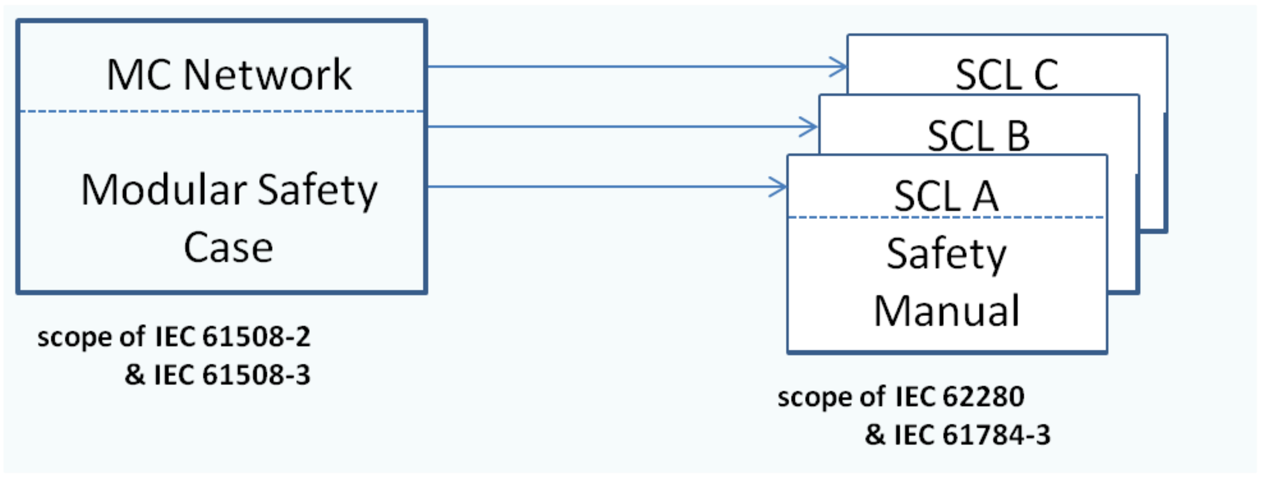

MC Network
Modular Safety Case
scope of IEC 61508-2 & IEC 61508-3
SCL C
SCL B
SCL A
Safety Manual
scope of IEC 62280 & IEC 61784-3

Figure 58: Relationship between Network Modular Safety Cases and the Compliant Items

6.3 Mixed-Criticality Patterns & Product Families

A Mixed criticality system is a system that contains applications of different criticality levels that interacts and coexists on the same computational level.

Design patterns provide solutions to well known and repetitive problems, e.g. how to share a memory region that commands digital outputs (safety and non-safety related) among a set of partitions of mixed-criticality. The reusability of design patterns also enables the cost competitive development of products and reduces the probability of systematic faults by means of reusability. Design patterns can also reuse previously defined modular safety cases.

A software design pattern can be defined as a description of communicating objects and classes that are customized to solve a general design problem in a particular context. Like in software development, design patterns have been also adapted for hardware design to provide implementation-independent and abstract views for recurring hardware design solutions. In general, a design pattern can be defined as an abstract representation of a general design problem that occurs in many applications.

Dependability is defined as the ability of a system to avoid frequent failures. Therefore, dependability as a system property is related to the safety-critical applications. The notion of dependability covers the following attributes:

- **Availability:** readiness for correct service,
- **Reliability:** Continuity of correct service,
- **Safety:** Absence of catastrophic failures,
- **Integrity:** Absence of improper system alterations,
- **Maintainability:** Ability to undergo modifications and repairs.

Dependable patterns describe what the safety engineer should consider and how it could be done. These patterns usually provides definitions of:

- Functionality to be provided to the system integrator.
- Information to be provided (arguments) from the system integrator.
- Proof of a certain realization (evidences) similar like information on the certification of a compliant item.

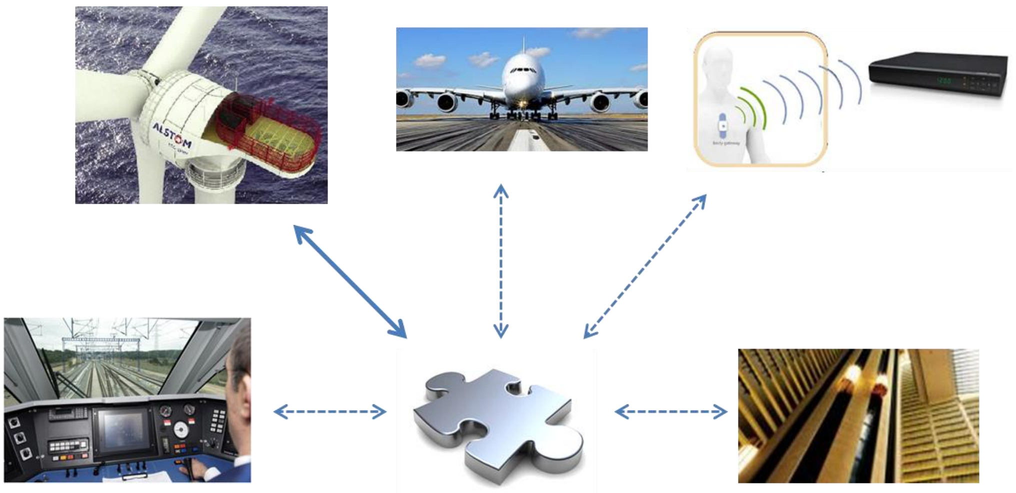


Figure 59: Design Patterns Re-usability

Product families / Product lines take into consideration the variability of products (e.g. low end and high end version) and the continuous trend towards the integration of more functionality and components in products. Product families could be efficiently constructed using design patterns that already consider variability and modular safety cases.

Product family certification indeed can take advantage of mentioned above approaches (cross-domain patterns and modular safety case). The use of modularity and cross-domain pattern based approach provides a way to reduce the development time and cost of new developments and pre-existing product actualization in a product line. In the development process of a product-line, there are two possible development scenarios. A scenario where a product-line pre-exist and a second one, where there is no product line at all. In each case, the development process will follow the IEC 61508 standard. Likewise, approaches of cross-domain patterns and modularity will be applied at different manner.

In the first case, when there is a pre-existent product line and certain modularity then cross-domain patterns will be based on the components and sub-components which are obtained from the functional decomposition of selected product-line. This way, a generic core of a product-family is obtained. After certification process of the core, in case of modifications or new developments, the same certified generic core can be re-used and the only re-certification needed will be the one of the modified part of product.

In the second approach, there is not a product line to decompose, so, in this case, the development process of a product and therefore the development process of a product line must be started from the beginning: from specifying requirements until first product development, across component and sub-component design/development/certification, to achieve one generic certified core of a product-line. This generic certified core will be the central element from where the development of new products will be started.

Bibliography

[1] A. Addisu, L. George, V. Sciandra, and M. Agueh. Mixed criticality scheduling applied to jpeg2000 video streaming over wireless multimedia sensor networks. In Proc. WMC, RTSS , pages 55–60, 2013.

[2] Kopetz, Hermann. *Real-Time Systems: Design Principles for Distributed Embedded Applications*, 2nd Edition. In J. A. Stankovic (Editor): Real-Time Systems Series. Springer, New York, Dordrecht, Heidelberg, London, 2011.

[3] Butler, Ricky W. *A primer on architectural level fault tolerance.* Tech. Rep. NASA/TM-2008-215108, NASA Langley Research Center, Hampton, VA, USA, Feb. 1, 2008.

[4] Pullum, Laura L. *Software Fault-Tolerance - Techniques and Implementation*. Artech House Boston, London, Sept. 2001.

[5] Boyer, Robert S. and Moore, J. Strother. *MJRTY—A Fast Majority Vote Algorithm.* In Boyer, Robert S. and Pase, William (Editors): Automated Reasoning, Springer Netherlands, 1991, 1, 105-117.

[6] J. Zhang, K. Chen, B. Zuo, R. Ma, Y. Dong, and H. Guan. Performance Analysis Towards a KVM-Based Embedded Real-Time Virtualization Architecture. In 5th International Conference on Computer Sciences and Convergence Information Technology (ICCIT ’10), pages 421–426, 2010.

[7] J. Kiszka. Towards Linux as a Real-Time Hypervisor. In 11th Real-Time Linux Workshop (RTLW ’09), pages 205–214, Sep 2009. Dresden, Germany.

[8] M. Abuteir and R. Obermaisser, "Simulation Environment for Time-Triggered Ethernet," in Industrial Informatics (INDIN), 2013 11th IEEE International Conference on, 2013.

[9] “Aircraft Data Network Part 7 Avionics Full Duplex Switched Ethernet AFDX) Network”, Airlines Electronic Engineering Committee Std., 2005

[10] "IEEE standard for local and metropolitan area networks: Media Access Control (MAC) bridges," IEEE Std 802.1D-2004 (Revision of IEEE Std 802.1D-1998).

[11] P. Peti, R. Obermaisser, " A Fault Hypothesis for Integrated Architectures”, proceedings of the 4th International Workshop on Intelligent Solutions in Embedded Systems (WISES'06), pp. 47-64, Vienna, Austria. June 2006

[12] R. Obermaisser, Time-triggered communication, Boca Raton: Taylor & Francis, 2012.

[13] IEC, 61508 functional safety of electrical/electronic/programmable electronic safety-related systems, International electrotechnical commission, 2010.

[14] J. W. Dally and B. Towles, Principles and practices of interconnection networks, San Francisco, CA: Morgan Kaufmann Publishers, 2003.

[15] ARM. Trustzone: security foundation, 2010. http://www.arm.com/products/processors/technologies/trustzone.php.

[16] L. Fiorin, G. Palermo, and C. Silvano. "A security monitoring service for NoCs", in Proceedings ACM Conf. Hardware/Software co-design and system synthesis, pages 197–202, 2008.

[17] J. Porquet, A. Greiner, and C. Schwarz, "NoC-MPU: a secure architecture for flexible co-hosting on shared memory MPSoCs", in Proceedings Conf. Design Automation and Test in Europe, pages 591–594, 2011.

[18] Mizrahi, T., "Time synchronization security using IPsec and MACsec," *Precision Clock Synchronization for Measurement Control and Communication (ISPCS), 2011 International IEEE Symposium on* , vol., no., pp.38,43, 12-16 Sept. 2011
doi: 10.1109/ISPCS.2011.6070153

[19] Shirey, R., "Internet Security Glossary", RFC 2828, May 2000.

[20] H. Kopetz. Fault Containment and Error Detection in the Time-Triggered Architecture.2002.

[21] A. B. Campbell, O. Musseau, V. Ferlet-Cavrois, W. J. Stapor, and P. T. McDonald. Analysis of single event effects at grazing angle. IEEE Transactions on Nuclear Science, 45:1603–1611, 1998.

[22] Reference: B. Pauli, A. Meyna, and P. Heitmann. Reliability of electronic components and control units in motor vehicle applications. VDI Berichte, pages 1009–1024, 1998.

[23] J. Karlsson, P. Folkesson, J. Arlat, Y. Crouzet, and G. Leber. Integration and comparison of three physical fault injection techniques. In B. Randell, J. Laprie, H. Kopetz, and B. Littlewood, editors, Predictably Dependable Computing Systems, pages 309–327. Springer Verlag, heidelberg edition, 1995.

[24] R. Obermaisser and P. Peti. The Fault Assumptions in Distributed Integrated Architectures. SAE AeroTech Congress & Exhibition. 2007.

[25] F. Swiderski und W. Snyder, Threat modeling, Redmond, Wash: Microsoft Press, 2004.

[26] Perez, Jon and Anton Trapman. Deliverable D7.2 (Annex) - Wind Power Case-Study Safety Concept - V03.00. FP7 MULTIPARTES, 2014.

[27] Häb, Stephan and Gebhard Bouwer. Statement on the "Multipartes Wind Power Case-Study Safety Concept". TÜV Rheinland, 2014.

[28] Perez, Jon, David Gonzalez, Salvador Trujillo, Anton Trapman and Jose Miguel Garate. "A Safety Concept for a Wind Power Mixed-Critically Embedded System Based on Multicore Partitioning." In 11th International Symposium - Functional Safety in Industrial Applications (TÜV Rheinland). Cologne (Germany), 2014.

[29] Perez, Jon, David Gonzalez, Carlos Fernando Nicolas, Ton Trapman and Jose Miguel Garate. "A Safety Certification Strategy for Iec-61508 Compliant Industrial Mixed-Criticality Systems Based on Multicore Partitioning." Euromicro DSD/SEAA Verona, Italy, (2014).

Part III
Appendix