Distributed REal-time Architecture for Mixed criticality Systems

Initial Collection of Offline Adaptation Strategies for Mixed Criticality

D 4.1.1

Project Acronym	DREAMS	**Grant Agreement Number**		FP7-ICT-2013.3.4-610640	
Document Version	1.0	**Date**	2014-7-31	**Deliverable No.**	4.1.1
Contact Person	Ankit Agrawal	**Organisation**		Technische Universität Kaiserslautern	
Phone	+49 (0)631 205 3674	**E-Mail**		agrawal@eit.uni-kl.de	

Contributors

Name	Partner
Ankit Agrawal	TUKL
Gerhard Fohler	TUKL
Ramon Serna Oliver	TTT
Silviu S. Craciunas	TTT
Øystein Haugen	SINTEF
Franck Chauvel	SINTEF
Simon Barner	FORTISS
Claire Pagetti	ONERA
Patricia Balbastre	UPV
Fernando Eizaguirre	IKL
Jörn Migge	RTaW
Lionel Havet	RTaW
Zaher Owda	USIEGEN
Mohammad Abuteir	USIEGEN
Lionel Havet	RTaW
Alexander Mark Diewald	FORTISS

Table of Contents

List of Abbreviations

ASIL	Automotive Safety Integrity Level
AFDX	Avionics Full DupleX switched ethernet
ARP	Aerospace Recommended Practice
ASAAC	Allied Standard Avionics Architecture Council
BTA	Binary Tree Analysis
CFG	Control Flow Graph
CMF	Common-Model-Failures
CMS	Centralized Maintenance System
COM/MON	COMmand/MONitoring
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CSP	Constraint Satisfaction Problem
DAG	Directed Acyclic Graph
DAL	Design Assurance Level
DDR	Double Data Rate (Synchronous Dynamic Random Access Memory)
DHRTS	Distributed Hard Real-Time Systems
DM	Deadline Monotonic
DSE	Design Space Exploration
DUF	Detectable Unrecoverable Faults
EA	Evolutionary Algorithm
ECFG	Extended Control Flow Graph
EDF	Earliest Deadline First
FDIR	Failure Detection Isolation and Recovery
FIT	Failure-In-Time
FMS	Flight Management System
FPPS	Fixed Priority Pre-emptive Scheduling
HAZ	Hazardous
IEC	International Electrotechnical Commission
IMA	Integrated Modular Avionics
LCM	Least Common Multiple
MAJ	MAJor
MARS	MAintainable Real-Time System

MC	Mixed-Criticality
MCS	Mixed-Criticality System
MOEA	Multi-Objective Evolutionary Algorithm
MPT	MultiPARTES (EU funded Project)
MTTF	Mean Time To Failure
NI	Network Interface
NoC	Network-On-Chip
PN	Processing Node
RAM	Random Access Memory
RAMSS	Reliability, Availability, Maintainability, Safety and Security
RM	Rate Monotonic
RWCET	Remaining Worst Case Execution Time
SDC	Silent Data Corruption
SER	Soft Error Rate
SEU	Single Event Upset
SIL	Security Integrity Level
SPEA	Strength Pareto Evolutionary Algorithm
SWAT	SoftWare Anomaly Treatment
TBFD	Trace Based Fault Diagnosis
TDMA	Time Division Multiple Access
TLTL	Timed Linear Temporal Logic
TMR	Triple-Modular Redundancy
TSP	Time-Space Partitioning
TT	Time-Triggered
TTE	Time-Triggered Ethernet
TTP	Time-Triggered Protocol
TT-FS	Time-Triggered scheduling with Flexible Slack
TT-SP	Time-Triggered with Static Priority
VL	Virtual Link
WCET	Worst Case Execution Time

1 Introduction

This document is the deliverable *D4.1.1 - Initial collection of offline adaptation strategies for mixed-criticality* of the DREAMS project. It is the first deliverable of task *T4.1 – Offline adaptation strategies in mixed criticality*, and also of work package *WP4 – Architecture, tooling, scheduling and analysis*. This deliverable presents an initial collection of offline adaptation strategies. This acts as a starting point for consideration of possible choices of offline schedulers for mixed criticality systems in DREAMS. It also provides definition of certain ambiguous terms related to scheduling to establish a common notion amongst the partners. Since, this is the first deliverable from WP4, it also provides an overview of the model-driven development (MDD) process in DREAMS. The preparation process for this deliverable involved collection of the algorithms/approaches found in the literature and related building blocks provided in DREAMS deliverable D1.1.1 – *Architecture conceptualization: requirements, terms and principles*, guided by the objectives and partner roles mentioned in the description of work for T4.1 related to the offline adaptation strategies.

1.1 Objectives of this Deliverable

The objective of this deliverable is *"to document an initial set of collected adaptation strategies which forms the basis for the offline configurations for mixed-criticality applications"*. The collected offline adaptation strategies then act as candidates to be considered for selection in the next phase of T4.1 (D4.1.2), that relates to initial implementation of chosen candidates for scheduling of mixed criticality systems in DREAMS.

1.2 Positioning of the Deliverable in the Project

The goal of work package *WP4 - Tools, scheduling and analysis* is to define and implement algorithms for the transformation steps of the model driven development process defined by the DREAMS project. To achieve this goal, the work package WP4 is divided in four tasks: T4.1, T4.2, T4.3, and T4.1.

- Task *T4.1 – Offline adaptation strategies in mixed criticality* aims at generating offline configurations for mixed-criticality applications satisfying specified global constraints. This is achieved by deriving platform specific model from generic application model and platform model.
- Task *T4.2 – Generation of platform configuration files* aims at automatic generation of hardware and software configuration files out of a platform-specific model (computed by Task T4.1) instead of error-prone human translation.
- Task *T4.3 – Explicit variability configuration* aims at specification of the variability for the generic application model and platform model (obtained from task T1.4) and binding the variability with generic application model and platform model. These generic models with bounded variability then serve as input to the task T4.1.
- Task *T4.4 – Tool integration and demonstrator support* aims at promoting and supporting the usage of WP4 results in WP6 avionics demonstrator, WP7 wind power demonstrator and WP8 – healthcare demonstrator.

Task T4.1, in order to achieve it's aim, needs the following input from other tasks in the project:

- The generic application model and platform model from task *T1.4 – Development of methods for application, platform and variability modelling* of *WP1 - Architecture*,
- Variability extensions from task T4.3 of WP4,

- Local resource management schemes and strategies from *T2.2 - Resource management and adaptation services for mixed criticality* and *T2.3 – System software extensions for SoC virtualization* of *WP2 - Multicore virtualization technology*, network scheduling and
- Global resource management services from *T3.2 – Cluster-level safety and security* of *WP3 - Mixed-criticality network,* and
- The power models from T1.4 of WP1.

On the other hand, the output (offline configurations) generated by the task T4.1 (in D4.1.2 and D4.1.3) is needed as input by the following tasks:

- Task *T2.1 - Virtualization and memory interleaving extensions at network interface* of *WP2* to (a) limit contention of time-triggered packets in inter and intra-clusters, and (b) prevent inconsistent configurations.
- Task T2.2 of WP2 needs the strategies and algorithm for mixed criticality and the parameters used at runtime to decide the switching between configurations determined in task T4.1
- Task *T3.3 - Global resource management* of WP3 needs the pre-computed configurations from task T4.1 to reconfigure the system at the global level, when needed.

Since, this deliverable relates to task T4.1, we list below the deliverables of task T4.1 that will be provided during the course of the project and highlight the relations between them:

1. *D4.1.1 - Initial collection of offline adaptation strategies for mixed-criticality* (due in July 2014 – this deliverable)
   The collected offline adaptation strategies in this deliverable will act as possible candidates for extension, to solve the scheduling problem related to mixed criticality systems in DREAMS, in the next phase of T4.1 i.e. deliverable D4.1.2.
2. *D4.1.2 – Definition of offline adaptation strategies for mixed criticality and initial implementation* (due in March 2015)
   In D4.1.2, based on the collected offline adaptation strategies in deliverable D4.1.1, criteria would be determined for selection of appropriate candidates. Based, on the criteria, the selected approaches will then be considered for scheduling in mixed-criticality systems as envisioned in the project.
3. *D4.1.3 - Final Implementation and improvement of the offline adaptation strategies for mixed criticality* (due in July 2016)
   Based on the selected candidates in D4.1.2, D4.1.3 will provide final model implementation towards scheduling of mixed-criticality systems.

The dissemination level of this deliverable is public (PU) i.e. once approved by the European Commission (EC), it will be freely available for download through the DREAMS project website (http://dreams-project.eu).

1.3 Organization of the Deliverable

The deliverable is structured into two parts. Part A presents the general notions related to scheduling and comprises of chapters 2, 3, 4, 5, and 6. Part B presents approaches to generate schedules for time-triggered systems and comprises of the rest of the chapters from 7 to 11.

In part A,

- Chapter 2 presents the overview of model-driven development (MDD) process in DREAMS. This is relevant as task T4.1 relates to the mapping and scheduling in the MDD process.

- Chapter 3 establishes a common notion of the scheduling related terms that could be used ambiguously.
- Chapter 4 provides an overview of the classical scheduling approaches using servers, partition scheduling algorithms, approaches for static schedule generation and global and partitioned multiprocessor scheduling.
- Chapter 5 presents techniques for task timing analysis, message timing analysis, end-to-end timing analysis and partitioned systems scheduling analysis.
- Chapter 6 provides a background of the two faults - hardware faults and temporal faults, and then presents different fault recovery strategies.

In part B,

- Chapter 7 formalizes the scheduling problem considering timing constraints, two-level (IMA-level and intra-partition scheduling), and different execution models.
- Chapter 8 briefly shows two recent mapping algorithms.
- Chapter 9 presents initial solutions available for scheduling of mixed criticality applications in distributed hard real-time systems and various approaches for offline scheduling table generation for distributed hard-real-time systems.
- Chapter 10 provides an overview of different message scheduling strategies and constraints.
- Chapter 11 describes the multi-objective evolutionary algorithms (MOEA) based architectural exploration. It also provides an overview of the product-line testing technology.

Each chapter (from chapter 3) also has a dedicated section on challenges/shortcomings of the presented approaches w.r.t. the scheduling in DREAMS for mixed-criticality systems, which is relevant for the next phase of T4.1 i.e. deliverable D4.1.2.

2 Overview of the Model Driven Development Process

2.1 Introduction

The purpose of this section is to summarize the DREAMS development process with a focus on the platform configuration activities performed in the specification and implementation branch of the development process. In following, the suggested model-driven approach and its relation to the offline adaptation strategies collected in this deliverable will be discussed. More details on the development process and the corresponding tool support will be presented in the following deliverables:

- D1.3.1 “Description of development process with model transformations”
- D4.4.1 “Tools feature map and interoperability capabilities”

Furthermore, certification aspects in the development process and the tool-chain will be addressed in D5.4.1 “Guidelines for process and tool integration”.

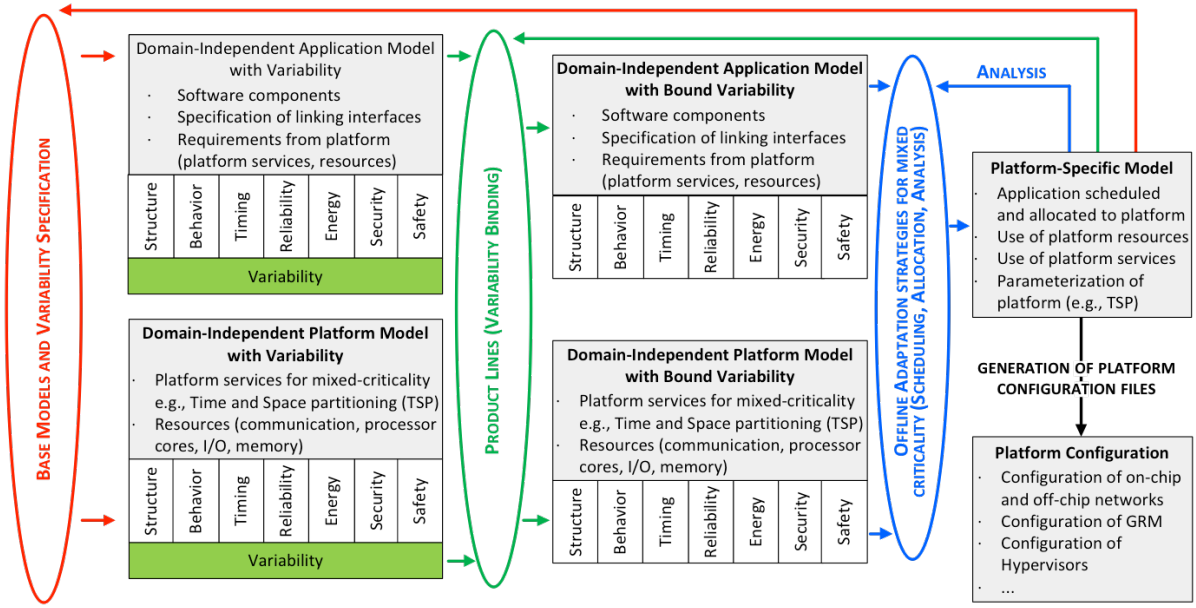

Figure 2-1 Proposed DREAMS Model Driven Development Process

The development process is structured into a processing chain (colored ovals in Figure 2-1, from left to right) which operates on models that provide different views onto the system under design. Here, each of the steps defines an entry point to the development process that starts at a different level of abstraction.

In the following, the basic workflow that is required to put a DREAMS system into operation will be summarized:

- In the DREAMS model driven development process, *offline adaptation strategies for mixed criticality systems,* (blue oval) serve as the entry point to the development process. The applied methods are used to compute a deployment of an application to an instance of the DREAMS platform. This step uses models of the application subsystems and a platform model as input. The result of this process is a platform-specific model that contains information about the deployed application. In the figure, the blue backward arrow (“analysis”) indicates that this step is an optimization process where different deployment alternatives are explored and that it provides analysis methods to rate the eligibility or

quality of a particular solution. The required algorithms and tools for this step are developed in T4.1.

- In the next step (bottom right in the figure), the platform-specific model is used as input for the backend of the processing chain that *generates configuration files* for the different HW/SW components of the target platform. T4.2 covers this part of the workflow.

Based on the above basic workflow that handles the configuration of a single system, the following extended development process can be defined which considers entire product-line families.

- In this case, the process starts with the definition of *base models and variability specification* (red oval in Figure 2-1). Here, a system model consisting of an application model and a platform model (see above) are used as base models. Additionally, the system designer provides a (separate) variability specification that defines which parts of the base model can be varied. Hence, the base model serves as template which is augmented with appropriate variation points. This step in the workflow is covered by task T4.3.
- Together, both models span an entire product-line family from which particular members can be selected. In the figure, this selection step is designated as variability binding (green oval), since for all variation points, a concrete choice is made. Task T4.3 also covers this step in the workflow. The result of this process is a system model that can be further processed using the basic workflow pointed out above.
- The green and red backward arrows in the figure indicate that also in this workflow, the eligibility and quality of the deployed system is rated. In case the selected solution does not satisfy all requirements, the following two options exist: At first, a different variability binding is selected (indicated by the green backward arrow), i.e. a different member of the product-line family is used as input for the basic workflow. In case the last step was not successful, the designer changes the definition of the product-line by modifying the base model and/or the variability specification (red arrow).

2.2 Models

The system model introduced in D1.2.1 summarizes the information to be provided by the DREAMS meta-models[1]. In Figure 2-2, the structure of a DREAMS system is sketched such way that is divided into a logical view (of the application) and a physical view (of the platform).

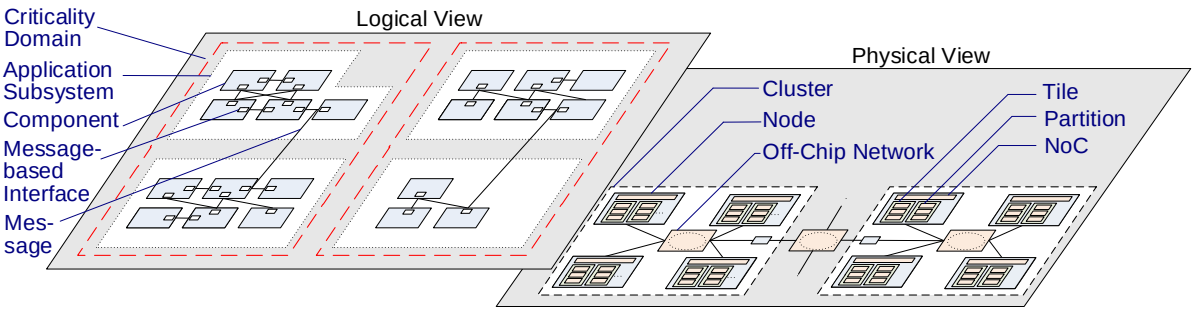

Figure 2-2 System Structure of Application (Logical View) and Structure of Platform (Physical View) [D1.2.1]

The purpose of the logical view is to provide the following information about mixed-criticality applications in a platform-independent way:

[1] The meta-models will be defined in D1.4.1 and D1.6.1. A preview of meta-models and model-transformations as well as an overview of existing building blocks will be presented in D1.3.1.

- Criticality-levels: A system is structured into criticality levels from the different application domains (e.g., DAL A to E in avionics, SIL 1-4 in multiple domains according to IEC-61508 and ASIL A to D in automotive – out-of-scope for DREAMS demonstrators).
- Component service: The specification of a component's interface defines its services, which are the intended observable behaviour 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, namely *periodic messages*, *sporadic messages*, and *aperiodic messages.*

The physical view of a DREAMS system defines the following hierarchical structure:

- The overall system is physically structured into a set of *clusters*.
- Each cluster consists of *nodes*.
- Each node is a multi-core chip containing *tiles*.
- 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.
- The processor cores within a tile can run a *hypervisor* that establishes time-and-space partitions, each of which executes a corresponding software component.

The communication between the above entities is provided by the following components:

- The connection between clusters is provided by *inter-cluster gateways* that are formed by *off-chip gateways* located between two clusters.
- Nodes are interconnected by an *off-chip real-time communication network*.
- Tiles are interconnected by a *Network-on-Chip* (NoC). Each tile provides a Network Interface (NI) to the NoC offering ports for the transmission or reception of NoC messages.
- An *on/off-chip gateway* is responsible for the redirection of messages between the NoC and the off-chip communication network.
- Off-chip and on-chip networks are responsible ensuring for time and space partitioning as well as the integrity of messages between the respective communication partners.

In order to provide their specified services, components of the logical view must be mapped to the resources of the physical platform:

- Components must be assigned to partitions with suitable computational resources
- Messages must be mapped to the communication networks, taking into account the required timing and reliability properties, and the routing between different parts of the physical platform.

3 Terminology/Definitions

In section 3.1, we provide a common notion of 14 terms related to scheduling to remove ambiguity amongst partners in Task T4.1 in DREAMS . In section 3.2, we again list the terms and their definitions related to scheduling from the terminology provided in the deliverable D1.1.1 in DREAMS.

3.1 Newly Defined Terms

3.1.1 Aperiodic Task

It is a sequence of jobs or single jobs with variable inter-arrival time or unknown arrival time.

3.1.2 Application-specific Constraints

Constraint used to enforce or prevent the mapping of tasks to specific processing elements in order to satisfy application-specific requirements. For instance, a task that requires access to I/O signals must be mapped to that processor core that can access the corresponding peripheral.

3.1.3 End-to End Flow (ETEF)

A sequence (sorted list) of one or more execution units (tasks, threads, or processes) and associated timing constraints (period, deadline, offset, etc.)

3.1.4 End-to-End Latency

An end-to-end latency is the maximal time required to execute a functional chain, where a functional chain describes the behavior of a sequence of functions, not necessarily hosted on a shared core (or multi-core), from an input until the production of an output.

3.1.5 Fault Model

A fault model describes the types of failures that could encounter a system. “A fault model is nothing more than a statement of how the system is expected to fail.”[Koo96]

3.1.6 Mapping

Mapping simply refers to the assignment of tasks (frames or messages) to processors (virtual links).

3.1.7 Mode

In time-triggered (TT) systems, a mode refers to a phase of operation of a real-time system. For example, an aircraft goes through various phases during its flight: take-off, normal-flight and landing phases. The tasks performed in each of these phases change. So, a mode best captures a particular phase of operation of a real-time system, through a corresponding scheduling table [Foh93].

3.1.8 Mode-change

In TT systems, a mode-change refers to the deterministic switching among a number of modes (essentially TT scheduling tables) such that the offline-scheduled real-time system is able to adapt to changing environmental situations. It is a system-wide change [Foh93].

3.1.9 Partition Scheduling Plan

In XtratuM, a partition scheduling plan provides information about which partition executes at each point in time based on the global time for each CPU.

3.1.10 Partition Slot

A partition slot is the amount of time that a partition can execute without other partition's preemption.

3.1.11 Periodic Task

It is an infinite sequence of jobs with constant inter-arrival time (period).

3.1.12 Restartable Task

Defined in the WP6. “Asynchronous process associated with a manager process in charge of the activation and cancellation of the computation” (DREAMS D6.1.1)

3.1.13 Schedule Reconstruction

Two-step encoding used in MOEA-based architectural exploration that has initially been introduced in [Luk07] in order to minimize the size of the chromosome by only storing the task mapping and the redundancy configuration in it. If the full schedule is required in the fitness evaluation of a candidate solution, a scheduler is used to rebuild the schedule from the information contained in the chromosome.

3.1.14 Scheduling

Scheduling deals with the allocation of resources in time to tasks such that the specified constraints like temporal, communication, synchronization constraints etc. are met.

3.1.15 TT Scheduling Table

A TT-scheduling table provides information about which task needs to be assigned the TT slot based on the global time. A TT scheduling table represents a feasible schedule. At runtime, the dispatcher simply executes the decisions in the TT scheduling table.

3.1.16 Single Event Upset (SEU)

A single event upset (SEU) is a change of state caused by electromagnetic rays. The effect of such particles on multi-core platform is the switch of a bit memory (in the DDR, local buffers in the internal bus, the caches or the processor registers).

3.1.17 Sporadic Task

It is an infinite sequence of jobs with a minimum inter-arrival time.

3.1.18 Task

A task is a piece of code executing sequentially on the CPU.

3.1.19 TT Slot

In TT systems, the time axis is statically partitioned into slots referred as TT slots. All TT slots have the same fixed-length [Kop11].

3.2 Terms Defined in D1.1.1 - *Architecture Conceptualization: Requirements, Terms and Principles*

3.2.1 Assurance Level

The assurance level is determined from the safety assessment process and hazard analysis by examining the effects of a failure condition in the system.

For example, DO-178B distinguishes five assurance levels in avionics: Level A (Catastrophic) refers to systems where a failure that may cause a crash, Level B (Hazardous) implies a large negative impact on safety), Level B (Major) involves a significant, but lesser impact than a hazardous failure, Level C (Minor) refers to an even lesser impact on safety. The failure of a Level E system has no safety effect. [DREAMS D1.1.1]

3.2.2 Error

An error is that part of the system state which is liable to lead to a subsequent failure. A failure occurs when the error reaches the service interface. [DREAMS D1.1.1]

3.2.3 Failure

A failure occurs when the delivered service deviates from fulfilling its specification. [DREAMS D1.1.1]

3.2.4 Fail-operational System

A fail-operational system is able to tolerate one or several faults. Fail-operational systems send correct messages despite the failure of their subsystems. [DREAMS D1.1.1]

3.2.5 Fail-safe system

If a fail-safe system one or more safe states can be reached in case of a system failure. Fail-safeness is a characteristic of the controlled object, not the computer system. In fail-safe systems the computer system must have a high error-detection coverage. [DREAMS D1.1.1]

3.2.6 Fault

A fault is the adjudged or hypothesized cause of an error. Faults can be internal or external of a system.

Examples of types: An external fault (e.g. a malicious attack) causes an error, and possible a subsequent failure. An internal fault (i.e. vulnerability) allows an external fault to harm the system and has to pre-exist in the system. [DREAMS D1.1.1]

3.2.7 Fault-Containment Region

A Fault Containment Region (FCR) is a subsystem that operates correctly regardless of any arbitrary logical or electrical fault outside the region. [DREAMS D1.1.1]

3.2.8 Fault Hypothesis

The fault hypothesis is the specification of the faults that must be tolerated without any impact on the essential system services. The fault hypothesis states the assumptions about units of failure (see Fault Containment Region), failure modes, failure frequencies, failure detection, and state recovery. [DREAMS D1.1.1]

3.2.9 Latency Constraint

A latency constraint describes how occurrences of a “target” event are placed relative to each occurrence of a “source” event. Source and target events are specified by a timing event chain.

Every instance of the source event must be matched by an instance of the target event, within a time window starting at lower and ending at upper time units relative to the source occurrence. [DREAMS D1.1.1]

3.2.10 Mixed-Criticality Systems

Mixed-criticality is the concept of allowing application subsystems that must meet different assurance levels (e.g., ranging from DAL A to DAL E in RTCA DO-178B, SIL1 to SIL4 in EN ISO/IEC 61508) to seamlessly interact and co-exist on the same networked distributed computational platform. [DREAMS D1.1.1]

3.2.11 Partition

A partition is the execution environment for a component with corresponding resources (e.g., processor, memory, communication, input/output). The resources for a partition are protected by temporal partitioning and spatial partitioning in order to avoid unintended feature interaction and fault propagation between components. [DREAMS D1.1.1]

3.2.12 Repetition Constraint

A Repetition constraint describes the distribution of the occurrences of a single event. Typical examples of these events are Task Activation, Frame Instantiation, Task Execution End, Frame Transmission End.

Prominent examples of repetition constraints are periodic repetition with jitter and sporadic repetition with minimal inter-occurrence time. [DREAMS D1.1.1]

3.2.13 Spatial Partitioning

Spatial partitioning ensures that the service in one partition cannot alter the code or private data of another partition. Spatial partitioning shall also prevent a partition from interfering with control of external devices (e.g., actuators) of other partitions. [DREAMS D1.1.1]

3.2.14 Synchronization Constraint

A Synchronization constraint describes how tightly the occurrences of a group of events follow each other. This is typically expressed by a temporal window, i.e. an upper bound on the temporal distance between the occurrences of the events of the group.

An example is the reading of input data from different sensors, which must occur in a small time window to ensure a temporally consistent view of the environment. [DREAMS D1.1.1]

3.2.15 Temporal Partitioning

Temporal partitioning ensures that a partition cannot affect the ability of other partitions access shared resources, such as the network or a shared CPU. This includes the temporal behavior of the services provided by resources (latency, jitter, duration of availability during a scheduled access). [DREAMS D1.1.1]

3.2.16 Timing Constraint

A Timing Constraint is a constraint on the occurrence times of one or more Timing Events. [DREAMS D1.1.1]

3.2.17 Worst Case Execution Time (WCET)

The Worst Case Execution Time is the maximal delay needed to execute all instructions of a task, excluding interruption or preemption delays. [DREAMS D1.1.1]

3.2.18 Worst Case Response Time (WCRT)

The Worst Case Response Time is the worst delay between the occurrence time of the Task Activation and the occurrence time of the Task Execution End. With respect to the WCET, it includes interruption/preemption or initial blocking delays (non-preemptive scheduling). [DREAMS D1.1.1]

3.3 References

[Foh93] G. Fohler, "Changing Operational Modes in the Context of Pre Run-Time Scheduling," *IEICE Transactions on Information and Systems,* vol. Special Issue on Responsive Computer System, 1993.

[Luk07] M. Lukasiewycz, M. Glaß, C. Haubelt, and J. Teich. (2007). SAT-decoding in evolutionary algorithms for discrete constrained optimization problems. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 07)*, pages 935–942.

[DREAMS D1.1.1] C. DREAMS, "D1.1.1 - Architecture Conceptualization: Requirements, Terms and Principles," DREAMS Consortium 2014.

[Koo96] Philip J. Koopman, “Lost Messages and System Failures”. Embedded Systems Programming, 9(11), October 1996, pp. 38-52.

[Kop11] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*, Second Edition ed.: Springer-Verlag New York Inc, 2011.

4 Hierarchical Scheduling Algorithms

4.1 Overview

In the last years, embedded systems have increased its processing capabilities and its computational resources in such a way that they are capable of executing several concurrent real-time applications that would formerly have required to be implemented on separate hardware platforms. Allocating several applications to a single processor not only reduces cost, but also ensures better reliability, as the overall complexity of the system is reduced.

CPU-sharing is implemented by some kind of partitioning scheme that time-multiplexes the physical resources among the different application task groups, with the goal that each application task group may be programmed as if it had dedicated access to a physical resource, i.e. , without interference from other task groups due to resource sharing. When composing a system comprising a number of applications, it is typically a requirement to provide temporal isolation between the various applications. Temporal isolation means that the processor time budget available for an application is guaranteed in spite of other applications possibly overrunning their budgets.

There is currently considerable interest in hierarchical scheduling as a way of implementing partitioned systems.

4.2 Hierarchical Scheduling in Partitioned Systems

In a hierarchical system, a global scheduler is used to determine which application should be allocated the processor at any given time, and a local scheduler is used to determine which of the chosen application's tasks should actually execute. A number of different scheduling schemes have been proposed for both global and local scheduling. These include cyclic or time slicing frameworks, dynamic priority based scheduling and fixed priority scheduling. For example, ARINC653 defines a cyclic scheduler at the global level and a fixed-priority scheduler at the local level [ARINC653].

Next Figure shows an example of a partitioned system with a hierarchy of two levels. In the global level, the scheduling is based on a table written in a static configuration file that establishes the temporal windows (or slots) in which partitions will execute. These slots are passed to each local scheduler that manages tasks execution inside these slots.

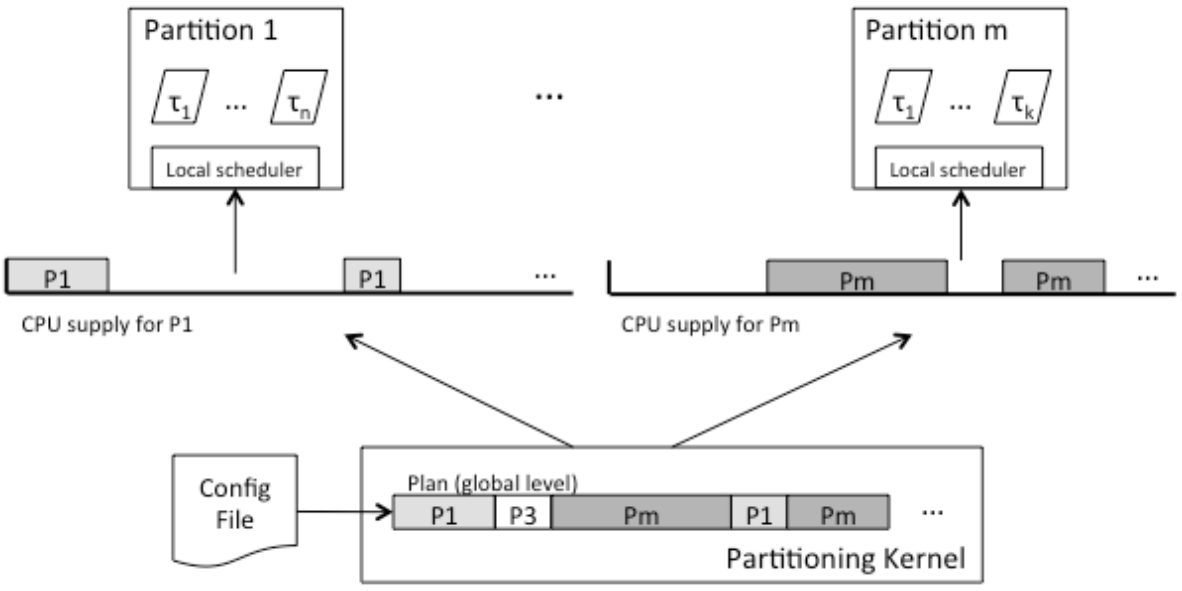

Figure 4-1 Example of a partitioned system with two levels of hierarchy

In the next sections, scheduling algorithms and schedulability analysis will be described for the global (hypervisor) and the local (partition) levels.

4.2.1 Hypervisor Scheduling

Traditionally, the scheduling method used in the global level has been static or dynamic.

4.2.1.1 Static Scheduling

Under this approach the system is analysed off-line and a static schedule (or table) is generated.. Each row of the table defines a sequence of jobs that are executed on the occurrence of a timer signal, usually an interrupt generated by a hardware timer. Such a sequence is called a *frame* or a *minor cycle*. The whole table defines a global repetitive behaviour for the system, which is called a *major cycle*. The major cycle period is the *hyperperiod* of the system, which is the least common multiple (lcm) of all the task periods:

$$H = \text{lcm}\ T_i$$

This method is sometimes used in critical systems as it provides a deterministic real-time behaviour, which can be easily analysed with respect to real-time requirements for verification and validation purposes. It can be implemented with a simple, robust run-time scheduler (commonly called a *cyclic executive*), without the need for a full-fledged operating system kernel. Access to shared data poses no problem as long as critical regions are kept within a single. Schedulability analysis is performed at the time of building the scheduling table, i.e. at design time.

In spite of its simplicity and robustness, static scheduling has some drawbacks that may make it inappropriate for systems with some degree of complexity:

- Static scheduling requires that all tasks are periodic. Sporadic tasks can only be incorporated in the form of *poll servers*, i.e. periodic tasks that check if an event has occurred and execute the associated sporadic job if this is the case.
- The generation of the scheduling table may be quite a complex problem. In the general case it has been proved to be NP-complete, although some heuristics have been proposed that can be used in many practical situations to compute a feasible schedule in polynomial time [Li00][BW09]. In practice, designers often adjust the task periods so that they are harmonic, i.e. multiples of each other.
- Even with harmonic periods, tasks with very long periods are very difficult to accommodate.
- Tasks with long execution times may have to be split into a number of shorter segments in order to find a feasible schedule.

4.2.1.2 Dynamic Scheduling

In dynamic scheduling, it is common to use server-based algorithms. Server-based scheduling provides a way to reserve a fraction of processor time. In this approach, a separate server is allocated to each application. Each server has an execution capacity and a replenishment period, enabling the overall processor capacity to be divided up among the different applications. Each server has a unique priority that is used by the global scheduler to determine which of the servers with remaining capacity, and which tasks that are ready to execute, should be allocated the processor. Several kinds of server algorithms have been defined, which differ in the way their budget is replenished, and how they deal with unused budget. The most common ones for fixed-priority are:

- *Periodic server*. A periodic server or *polling server* is a periodic task with period T_s and budget C_s. Whenever it is activated, if there is any aperiodic job pending, it is executed as long as there is some budget available. Otherwise, the server suspends itself until the next period. Unused budget cannot be saved for possible future jobs. If an aperiodic job is requested later in the same period, its execution is postponed until the next period.

  If the budget is fully consumed, the job execution is suspended until the start of the next period, at which time the budget is replenished.

A periodic server is commonly assigned the highest priority, and its interference on other tasks can be calculated using response-time equations.

- *Deferred server*. A deferred server [LSS87] is similar to a periodic server, but it saves its unused budget until the end of its period. If an aperiodic job is requested at any time within the period, it is immediately dispatched for execution. The budget is replenished at the start of each period, and unused budget from the previous period is lost.

  Deferred servers are usually assigned the highest system priority. RTA equations can be adapted to deal with server interference on real-time tasks.

- *Sporadic server*. A sporadic server [SSL89] is similar to a deferred server, but its budget is replenished using more complicated rules, which enable better processor utilization and faster response times for sporadic jobs. There are several variants of the basic sporadic server algorithm, but in any case the effects of the sporadic server on the timing behaviour of other tasks are not worse than those of an equivalent periodic task.

  Sporadic servers can be assigned any priority in the system priority range. In terms of schedulability analysis, a sporadic server is equivalent to a sporadic task and can thus be easily accommodated within RTA.

Although dynamic-priority versions of deferred and sporadic servers that can be used with Earliest Deadline First (EDF) have been defined (see e.g. [Bu10]), there are some kinds of servers that have been developed specifically for EDF.

The best-known dynamic priority servers are [SB94]:

- *Priority exchange server*. A priority exchange server is similar to a deferrable server, but its unused capacity is not lost, but exchanged for the execution time of a lower-priority task, i.e. a task with a longer absolute deadline.
- *IRIS (Idle-Time Reclaiming Improved Server) (*[MSP04]): Based on the Constant Bandwidth Server (CBS, [AB04]), it allows the coexistence of hard, soft and non real-time tasks. The proposed algorithm is specifically designed to handle computational overload. A task that needs more CPU-time than reserved can re-use the spare bandwidth, without interfering with the others tasks. With respect to other reclamation schemes, the novelty of IRIS algorithm is that the spare bandwidth is fairly distributed among the needing servers.
- *Total bandwidth server*. A total bandwidth server tries to shorten the response time of aperiodic requests (1,..k) by assigning them all the available processor time. More specifically, if a maximum utilization U_s is assigned to the server at design time, a new aperiodic request with computation time C_k released at r_k is assigned an absolute deadline

  $$d_k = \max(r_k, d_{k-1}) + \frac{C_k}{U_s}$$

  and then it is scheduled using EDF, as any other task.

Overall, it can be said that the total bandwidth server has best performance [Bu10].

4.2.2 Partition Scheduling

Most used scheduling algorithms for the local level (internal scheduling of tasks inside a partition) are priority based scheduling algorithms. Most popular priority based algorithms Rate-Monotonic and Deadline Monotonic if the priority is fixed. Earliest Deadline First is the most popular if the priority can change during run time execution. However, ARINC-653 allows a fixed-priority scheduler on the local level.

4.2.2.1 Fixed-Priority Assignment

Fixed-priority pre-emptive scheduling (FPPS) is a scheduling method in which each task has a fixed priority, and all its jobs are executed with the same priority. If a system has several operating modes, a task may have different priorities in each mode. The set of tasks is usually considered to be static in each operating mode.

Some possible ways to assign priorities to tasks are:

- *Rate-monotonic scheduling* (RMS). Priorities are assigned according to task periods: the task with the shortest period is assigned the highest priority. This method is optimal for sets of periodic, independent tasks with deadlines equal to periods [LL73].
- *Deadline-monotonic scheduling* (DMS). Priorities are assigned according to task deadlines: the task with the shortest deadline is assigned the highest priority. This method is a generalization of RMS, and is optimal for sets of periodic and sporadic independent tasks with arbitrary deadlines [LM82].
- *Audsley's algorithm* [ATB93] can be used to assign priorities in more complex situations.

4.2.2.2 Dynamic-Priority Assignment

Earliest-deadline first (EDF) is a real-time scheduling method in which priorities are dynamically assigned to individual jobs based on their absolute deadline times. In general, it provides better processor utilization that FPPS at the cost of a slightly greater complexity.

The job with a closest deadline gets the highest priority and is thus executed first. Different jobs of the same task can have different priorities, although a job's priority remains fixed for all of its execution. Next Figure shows a sample execution with EDF. In the figure, blue areas correspond with execution of tasks, while down arrows are the release time of the tasks.

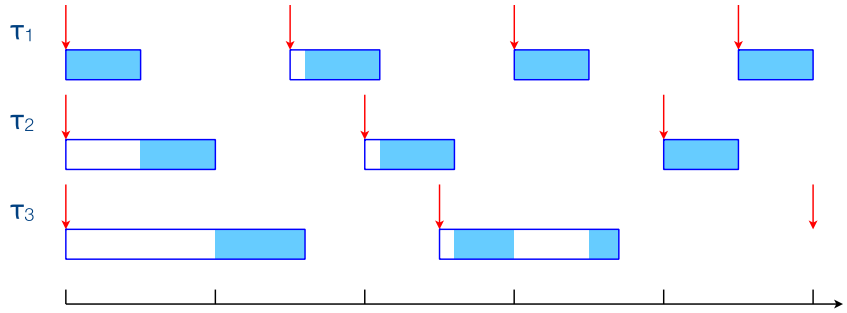

Figure 4-2 Sample task execution with EDF.

EDF is optimal for a static set of independent periodic or sporadic tasks with deadlines equal to periods (Liu & Layland 1983). For more complex situations, schedulability analysis provides insight on the suitability of the method for an application.

4.3 Considerations of Static versus Priority-based Scheduling for the Hypervisor

This comparison will be made from the point of view of mixed criticality and certification issues, that are key point in DREAMS.

4.3.1 Mixed Criticality

Many embedded systems that are found in industrial practice are made up of several applications with different criticality levels. Such systems are known as *mixed-criticality systems*. Mixed-criticality systems have been built in the past using a *federated* approach, in which applications with different criticality levels are executed on different, physically isolated, hardware platforms. The need for reducing cost by better using the computing power of modern processors led to an *integrated* approach, in which applications with different criticality levels can run on the same computer platform. Such an approach implies that failures in low-criticality applications, which have been

developed with less strict standards, do not compromise the integrity of higher-criticality applications.

Temporal separation can be implemented on a fixed-priority system, by assigning higher priorities to tasks in high-criticality applications, and lower priorities to other tasks. Such a scheme is simple to implement, and can be made robust enough by adding run-time monitoring mechanisms [PUZ06], but is often inefficient in terms of processor utilization, and is difficult to combine with spatial separation mechanisms [UPL08]. The multi-criticality task model proposed by [Ve07] follows a related approach. Its key idea is to solve the under-utilization problem at design time. The basic scheduling algorithm is fixed priority preemptive scheduling (FPPS) on a uniprocessor. In this class of scheduling algorithms, the schedulability of a task depends on the WCET values of the tasks of equal or higher priorities. But, according to Vestal, from the point of view of a less-critical task, the WCET values of higher-critical tasks are needlessly pessimistic. The schedulability of lower-priority (and lower criticality) tasks can be improved by using different WCET values for each task, one for its own criticality level, and one for each lower criticality level. In this way, schedulability tests can be carried out independently for each criticality level. By always giving the highest priority to the most critical tasks, tasks with a lower-criticality can be accommodated in the scheduling analysis without interfering with the higher-criticality tasks. This work was extended by [BV08]. In spite of the improvements in processor utilization, though, the problem of providing spatial separation in priority-based systems still remains.

Although some of the above approaches show promising characteristics, none of them has reached industrial acceptation yet, as they only considered very simple tasking models and lack robustness against some of the failures that can arise in real-life systems.

ARINC 653 is a standard for avionics systems that provides time and space separation in a simple, robust way, with certification at the highest criticality levels in view. It supports the concept of Integrated Modular Avionics (IMA), by specifying a set of application executive services (APEX) (ARINC653). The ARINC 653 standard provides support for dividing a set of applications into a fixed number of partitions. Temporal separation is implemented by using a static, table-driven global scheduler. Each partition contains a set of *processes*, which can be periodic or aperiodic, and are scheduled with FPPS as long as the partition is active. Other features of the ARINC standard include a system partition that can access system resources, including hardware devices, message-based inter-partition communication, and a health monitor for fault detection and confinement.

The ARINC 653 static scheduling approach has been taken as a starting point for XtratuM, based on the predictability and temporal isolation requirements for the latter (see e.g. [CRM10]) in the aerospace domain.

4.3.2 Certification

The scheduling used in the local and global level will depend on the standard used to certify the system. In general, partitions within the system are scheduled on a fixed, cyclic basis. The order of partition activation is defined at configuration time using configuration tables. This provides a deterministic scheduling methodology whereby the partitions are a predetermined amount of computing time. Tasks within the partition can be scheduled statically or dynamically. This is the case of ARINC-653.

4.3.3 Incremental Certification

In many systems, there is often a need to perform updates to the software to provide additional functionality or capability to the system. This will need a re-certification of the system. If the system architecture is defined so it uses modular components a re-certification of the entire system may not be necessary but an incremental certification of the changed modules. This requires a strong isolation between all components. Incremental certification refers to the certification of a system that is a re-used of a previously certified system. Incremental certification allows:

- to change application or configuration entities without affecting the entire system and without requiring re-testing or re-certification of other independent entities
- to reuse applications from one IMA project on the next IMA project and Without having to re-write and re-test the entire application

As far as temporal requirements are concerned, changes in temporal parameters of one partition shall not affect other partitions scheduling (see section 4.8). In the hypervisor level, added partitions will execute only in idle slots. If a partition changes its temporal parameters, it will only use its slots and/or idle slots. In any case, unchanged partitions will execute exactly in the same slots of the original schedule.

4.4 Computational Model in Partitioned Systems

4.4.1 Tasks, Partitions, Execution flows

In the classical scheduling theory, a "task" is the execution element that contains all the scheduling attributes: period, deadline, relations with other tasks, etc. But this model does not capture properly the operation of complex systems. However, it is how traditionally application requirements have been expressed.

A more general and flexible model consists of the following entities: Partitions, tasks, ETEFs and slots, in accordance with MARTE-UML standard (adds capabilities to UML for model-driven development of Real Time and Embedded Systems).:

Partition: Is the container of the tasks. The tasks are scheduled by the partition internal scheduler. Main attributes:

- Set of tasks that belong to the partition.
- The local scheduling policy.

Task: Is an elemental activity that is executed in a partition. It can be an Ada task, a POSIX process, a POSIX thread, a "C" function, or any other block of code that performs clearly identifiable work. Each task can only be executed by one and only one partition. Main attributes:

- WCET: Worst Case Execution Time.
- Period.
- Deadline.
- The set of mutual exclusion resources used during its execution.

ETEF: End To End Flow. Defines the desired temporal behavior of a set of tasks. Main attributes:

- Period. If no period is given, then the period is the MAF (Major Application Frame).
- Relative global deadline. It may be explicit or implicit (the same than the period).
- Offset. The earliest the ETEF may start its execution.
- Task instances that shall be executed every period.
- Precedence relation between instances.

Note that:

- A task may appear on multiple ETEFs.
- An ETEF can have tasks of different partitions.
- On a given ETEF, a task may have more than one task instances.

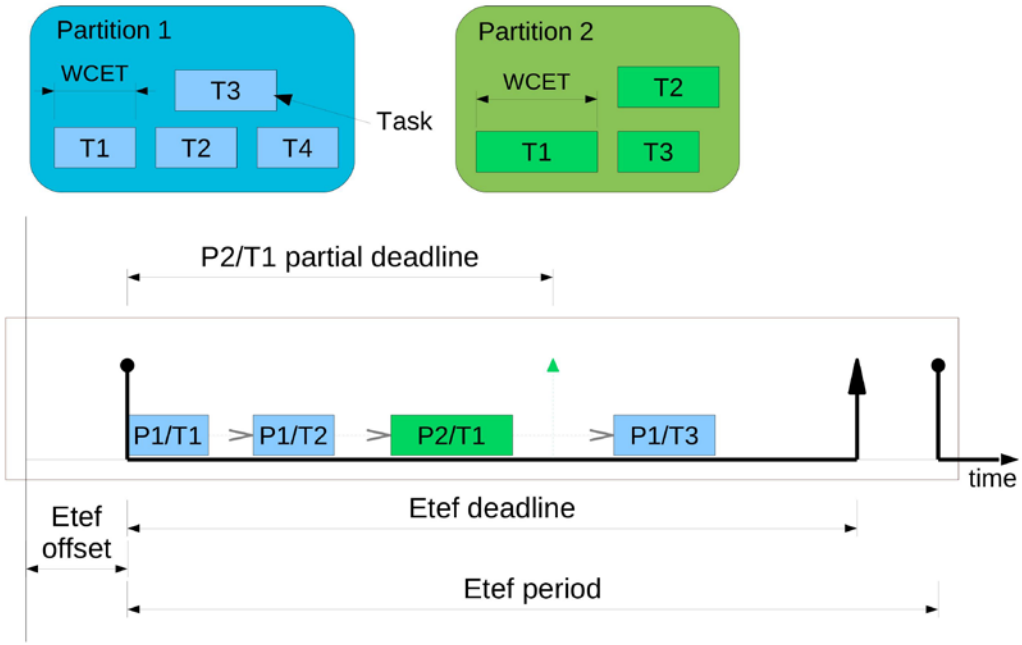

Figure 4-3 Example of temporal workload (Tasks, partitions and ETEFs)

Regarding the scheduling plan in the hypervisor level, the following definitions are needed:

Slot: A contiguous sequence of workload allocated to a partition. The workload allocated to partition can be: jobs whose associated task belongs to the partition, the time reserved for overheads and the context switch that may occur at the start of the slot.

Plan: It is the sequence of slots allocated to each partition. The plan has a duration of MAF units of time.

The previous model can successfully model the traditional periodic task model. This is accomplished by defining an ETEF for each task, as the next figure shows:

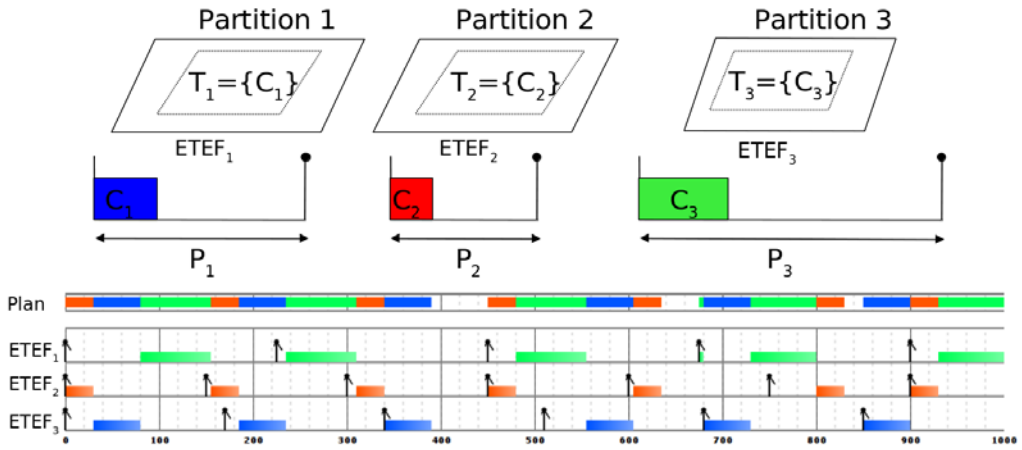

Figure 4-4 Each partition modeled as an ETEF with a unique task.

In the previous figure, each partition has one task and a unique ETEF is defined for each task.

4.5 Techniques for Static Schedule Generation

4.5.1 Approaches

Basic guidelines to generate a static schedule have been outlined in section 4.2.2.1, However, there are other methods to generate the schedule both at the global and the local level.

4.5.1.1.1 Compositional Scheduling

The basic idea of this approach is to extend the classical and widely used "divide and conquer" strategy to the temporal requirements.

Component technology has been widely accepted as a methodology for designing large complex systems through systematic abstraction and composition. The complexity of each component is hidden and abstracted through a clean and well-defined interface. One of the goals of the compositional scheduling model is to avoid performing a global schedulability analysis that considers the timing requirements of all the tasks in all the task groups. Ideally, each task group can be analysed by itself for schedulability.

The compositional model addresses the problem of guaranteeing the correct temporal operation of the composed system. The following two problems need to be addressed:

- Scheduling component abstraction problem: analyse the timing properties of a component independently. This problem lies in abstracting the set of real-time requirements of a component as a single real-time requirement, called scheduling interface. Ideally, the single requirement is satisfied, if and only if, the collective requirements of the component are satisfied.
- Scheduling component composition problem: compose independently analysed local timing properties into a global timing property. This problem is defined as composing the scheduling interfaces of components as a single real-time requirement

The work presented in [SL08] proposes a compositional real-time scheduling framework where global (system level) timing properties are established by composing together independently analysed local (component-level) timing properties. All the workload executed by a subsystem (which is assumed to be periodic) is modelled as a single execution task. Although starting from a completely different problem statement, the compositional scheduling and the aperiodic server models seem to arrive to the same solution. This approach has the following advantages:

- Clean isolation of scheduling concerns between partition developers and system integrator. The partition developers do not have to provide details about its internal operation (task attributes), just the temporal abstract interface of the partition (computation time, period).

On the other side, it has some drawbacks:

- The abstract interface is an upper bound of the real needs of the partition, therefore there is a non-negligible utilisation penalty. The more partitions are in the system, the less processor utilisation can be granted.
- Inter-partition resource sharing may be difficult to implement and also take into account in the schedulability analysis.

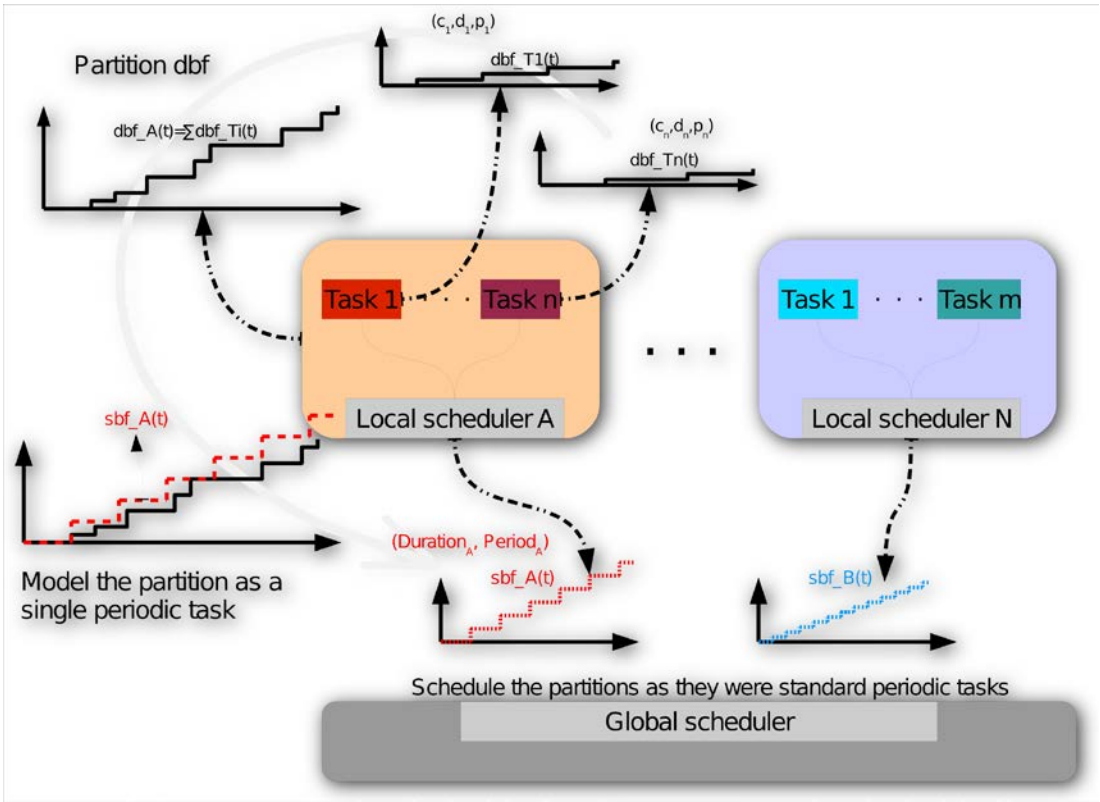

Figure 4-5. Compositional system mode

4.5.1.2 Flat Scheduling Model

This approach consists of removing the barriers defined by the partitioned system and to consider all the systems tasks at once. Then suppose that a single global scheduler is in charge of managing all the tasks, and conduct the corresponding schedulability analysis. The last step is to adapt the solution back to the partitioned system by grouping (trying to put together) the tasks of each partition in order to reduce the number of partition context switches.

This approach has the following advantages:

- Dependencies between tasks of different partitions can be analysed and solved.
- Mature theory support for this model.
- The resulting schedule (or scheduling policy) can be very efficient. Depending on the task model, it may be possible to find the optimal solution.

It has also the following drawbacks:

- If an optimized solution is desired, a deep knowledge of the timing attributes of all the tasks is needed in order to carry out schedulability analysis.
- There is no clean separation of concerns between partition developers and system integrator, or even among partition developers.
- A change of an attribute of a task may require the whole schedule to be reworked.

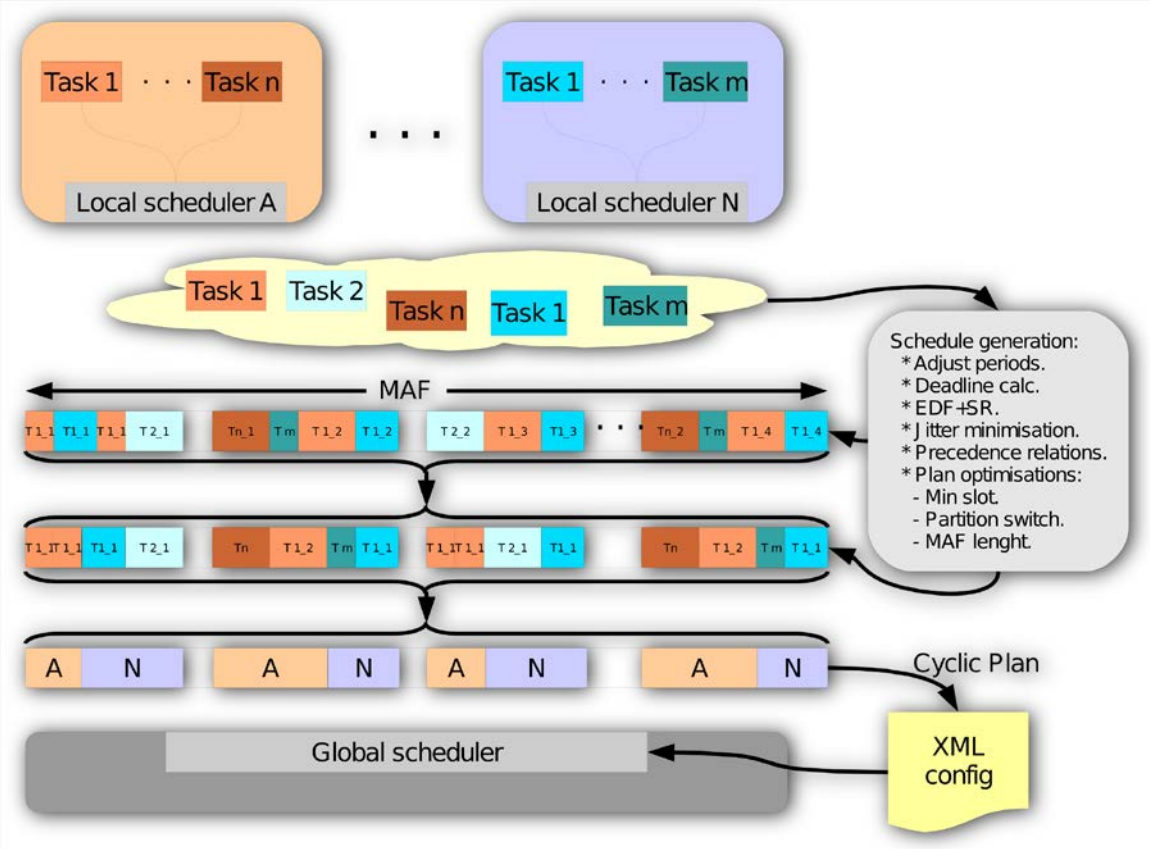

Figure 4-6 Flat system model.

4.5.2 MAF Optimization

In cyclic scheduling the hyperperiod (least common multiple of task periods) is referred as the major application frame (MAF), which is the interval at which the entire schedule is repeated. The minor time frame(MIF) represents intervals within the schedule at which the start of execution of a process may be synchronised. The minor time is a submultiple of the major time. The designer shall verify that the temporal constraints are met along the major time interval. Once the schedule is built, it is stored in a table which will be consulted on-line to select the active tasks. The shorter the hyperperiod the shorter the table, and consequently the smaller the memory footprint. For these two reasons, it is important to have a relatively small hyperperiod.

The most common technique is to select task periods to be harmonic, that is, all the periods have large common divisors. For example, in radar dwell applications, assigning for every task a harmonic period has the low overhead of maintaining constant temporal distance and schedulability analysis but is usually over-reserving the resources. To overcome this disadvantage, in [SGG03] an algorithm is developed to transform task periods into synthetic ones, but they do not have to be harmonic. In partitioned systems, tasks of different partitions may have different time scales and different temporal requirements, therefore the usage of harmonic periods is not always possible.

In [XP10] a method is proposed to reduce the hyperperiod by only reducing task periods. A more flexible and efficient method is presented in [BB12] where periods can be increased or decreased inside a defined range.

4.5.3 Jitter minimization

In control systems, the influence of jitter on performance quality is not always easy to analyze. From a control perspective, sampling jitter and latency jitter can be interpreted as disturbances acting upon the control system. The input-output latency decreases the stability margin and limits the

performance of the system. If the jitter and the latency are small, then these can be overlooked. Otherwise, they should be accounted for in the control design or, if possible, compensated for at run-time. Further information regarding these issues is detailed in [CA05]. In general, output jitter has a negative effect on the control performance. This can be significant depending on the term known as control effort introduced by Albertos and Olivares ([AC00]). The control effort measures how sensitive a control task is to time delays. The reduction of the output jitter and the subsequent improvement in the control performance, is directly associated with the reduction of task deadlines.

Regarding output jitter, several studies use scheduling solutions in order to reduce this. In [LH96] the difference between two consecutive finishing times of the same task is required to be bounded by a specific value. Natale and Stankovic[NS00] presented a new scheduling approach to minimise jitter in distributed real-time systems. In [L92],[KR93], high priority tasks are used to reduce the jitter. Kim et al. [KS00] assign new deadlines to tasks scheduled under EDF, using an integer linear program. It can be noted that, in all these studies dealing with deadline reduction except for the work developed by Hoang et al., this reduction is not optimal in the sense that deadlines can be minimised any further.

4.6 Scheduling in Multicore Systems

4.6.1 Overview

To address demands for increasing processor performance, silicon vendors no longer concentrate wholly on the miniaturization needed to increase processor clock speed, as this approach has led to problems with both high power consumption and excessive heat dissipation. Instead, there is now an increasing trend toward using multiprocessor platforms for high-end real-time applications. As before, in order to ensure that the specified real-time behaviour is guaranteed at run time, an appropriate scheduling method has to be found, for which schedulability analysis can be performed.

The multiprocessor scheduling problem consists thus in finding a feasible schedule for n tasks running on m processors. In the following we assume that $n \geq m$. Multiprocessor real-time scheduling is intrinsically a much more difficult problem than uniprocessor scheduling. The main reason is that few of the results obtained in uniprocessors can be directly applied to the multiprocessor case [DB11].

4.6.1.1 Types of Multiprocessor Systems

From the perspective of scheduling, multiprocessor systems can be classified into three categories.

- Heterogeneous multiprocessor systems. Processors are different.
- Homogeneous multiprocessor systems. Processors are identical.
- Uniform multiprocessor systems. The rate of the execution of a task depends only on the speed of the processor.

The community has focused mainly on uniform and homogeneous multiprocessor scheduling.

4.6.1.2 Types of Scheduling Algorithms

Multiprocessor scheduling has to solve two problems: the allocation problem, that consists in deciding on **which** processor a task should execute, and the priority problem, that is, **when** a task should execute.

Depending on these two problems, we can classify scheduling algorithms in:

- Allocation
  - No migration
  - Task-level migration
  - Job-level migration
- Priority

- Fixed task priority
- Fixed job priority
- Dynamic priority

Scheduling algorithms where no migration is permitted are referred to as partitioned, whereas those where migration is permitted (either at task or at job level) are referred to as global.

Partitioned and global approaches to static-priority scheduling on identical multiprocessors are incomparable in the sense that:

1) There are task sets that are feasible on m identical processors under the partitioned approach but not under global scheduling on the same m processors, and

2) There are task sets that are feasible on m identical processors under the global approach, which cannot be partitioned into m distinct subsets such that each individual partition is feasible on a single static-priority uniprocessor.

4.6.2 Partitioned Systems

In partitioning, each task is assigned to a single processor. This has the following advantages:

- Task overruns have only consequences in the same processor.
- There is no penalty in terms of migration cost.
- The implementation uses a separate run queue per processor rather than a single global queue in the global approach. This reduces overheads due to queue management.

On the contrary, the main disadvantages are:

- Finding an optimal allocation of tasks to processors is a bin-packing problem, that is NP-hard in the strong sense.
- There are task sets that are only schedulable if migration is allowed.
- Partitioned scheduling algorithms are not work-conserving, as a processor may become idle, but cannot be used by ready tasks allocated to a different processor.

Still, partitioning is widely used by system designers.

4.6.3 Global Systems

Global scheduling cannot be used to reduce the multiprocessor scheduling problem to many uniprocessor scheduling problems, contrary to partitioned scheduling. The fact that tasks are allowed to migrate in the global approach gives rise to many unexpected effects and disadvantages that complicate the design of scheduling and allocation algorithms for the global scheme. The most significant problems are:

- Migration of tasks to processors introduce a high overhead in the system.
- Migration increases the information flow between processors. This kind of communication may require the use of shared memory or communication channels.
- Predictability is much lower than that associated to the partitioned scheme.
- Some scheduling anomalies may occur, for example:
  - The Dhall effect: tasks sets with very small utilization may be unschedulable [DL78].
  - In global multiprocessor scheduling, the amount of execution of higher priority tasks is not the only reason for the delay of lower priority tasks, but the delay also depends on whether higher priority tasks execute at the same time.

On the contrary, the main advantages are:

- There are typically fewer context switches/preemptions. This is because the scheduler will only preempt a task when there are no idle processors.
- An advantage of the global scheme is its generality. Since tasks can migrate from one processor to another, the processor system “could be” better utilized.

- Global scheduling is more appropriate for open systems, as there is no need to run load balancing/task allocation algorithms when the set of tasks changes.

4.6.4 Scheduling Policies

Once a scheduling algorithm, like the ones described in the previous sections, has been selected to schedule a set of *n* tasks on m processors, schedulability analysis can be used in order to find out if all the deadlines can be guaranteed, i.e. the task set is schedulable. This problem appears to be much more difficult than in the uniprocessor case. As the schedulability analysis depends on the scheduling algorithm, a distinction between partitioned and global scheduling must be made.

4.6.4.1 Partitioned Scheduling Algorithm.

As the bin-packing problem is NP-hard in the strong sense, heuristics and global optimization techniques, such as simulated annealing, have been used to find good sub-optimal static allocations.

In the bin-packing problem, it is required to put *n* objects (tasks) with weight u_k (utilization) into the minimum possible number of bins (processors), such that the total weight of the objects on each bin do not exceed its maximum capacity (c).

For the partitioned scheme, the allocation of task to processors depends not only on the allocation algorithm itself, but also on the schedulability condition used. The value of *c* in the bin-packing problem depends on the schedulability condition.

The allocation problem is solved using typical memory allocators such as *First Fit* (FF), *Worst Fit* (WF), and *Best Fit* (BF). These heuristics use the task period parameter as the key for allocation. Others, such as FFDU (*First Fit Decreasing Utilization*) [cite], use the task utilization as a key for choosing the next task to allocate. These allocators combined with classical scheduling algorithms (FPPS or EDF) gives rise to the most popular partitioned scheduling algorithms, such as RMFF, EDFBF, RMFFDU, etc. A comparison of these allocation schemes can be found in [PM03]

4.6.4.2 Global Scheduling Algorithms

In a global scheduler, a single queue of ready tasks (or jobs) is maintained, from which tasks are extracted at runtime to be scheduled on the available computing resources, depending on their priority. To classify global scheduling algorithms we use the concepts of fixed job priorities or fixed task priorities. In the former approach, the priority of a task can only change at job boundaries, while in the latter all jobs generated by the same task have identical priorities.

Fixed job priorities

- The best-known scheduling algorithm for global multiprocessor scheduling is the so-called global EDF from which jobs are dispatched to any available processor according to a global priority scheme following EDF rules.
- [SB02] proposed the EDF-US[ς] algorithm that gives the highest priority to tasks with utilization greater than the threshold ς.

Fixed task priorities

- Global-FP in which the global priority scheme is based on fixed priorities.
- [An08] proposed a “slack monotonic” algorithm, where priorities are ordered according to the slack of each task given by $T_i - C_i$. This algorithm is known as SM-US.

Dynamic priorities

- *Pfair*[BCP96] .Pfair is a schedule generation algorithm that is applicable to periodic task sets with implicit deadlines. Pfair is based on the idea of fluid scheduling, where each task makes progress proportionate to its utilization (or *weight* in Pfair terminology). Pfair scheduling divides the timeline into equal length quanta or slots. At each time quantum t, the schedule allocates tasks to

processors, such that the accumulated processor time allocated to each task τ_i will be either $\lceil t_{ui} \rceil$ or $\lfloor t_{ui} \rfloor$.

A number of variants on the Pfair approach have been introduced (ERFair, PD, PD2, BF).

- EDZL. [Le94] introduced the Earliest Deadline until Zero Laxity.

4.7 Multiple Execution Plan Generation

The ARINC-653 specification Part 2 [ARINC653] defines several additional services as extended. One of these services, is the Multiple Module Scheduler, related to the ability to extend the single static module schedule by several scheduling plans defined in the configuration file and the possibility to change the current scheduling plan. This feature is useful to define a plan for each situation of the system. For example, in the next figure, the first plan deals with initialization issues and then, the system changes to a normal operation plan. Specific plans are defined for other situations such as maintenance or others.

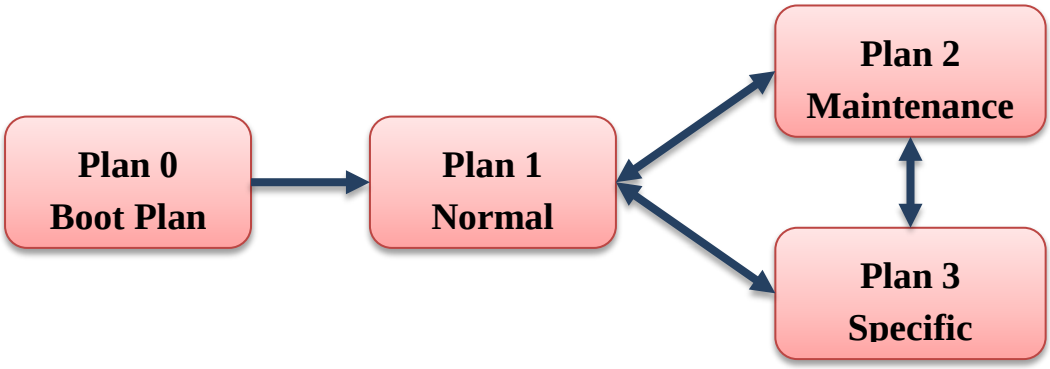

Figure 4-7 Changing between multiple scheduling plans

In order to change the scheduling plan in a secure way, all tasks must not be executing and waiting for the next period. Therefore, eve if a plan change can be requested anytime, it is only when a MAF is completed. If not, mode changes have to be considered.

4.8 Incremental Scheduling Generation

In long term projects it is likely to have specification upgrades which may yield in workload changes; new activities (partitions/tasks/etefs, etc.) may be defined or removed from the system. In such cases is usual that some parts of the software are developed early to be used by other components and acquire sufficient soundness to be considered as stable and no susceptible of further changes. In some business and research domains, where the software has a critical classification, the stability of the software is exhaustively tested involving significant expenses in terms of monetary and temporal resources. When considering partitioned real-time systems, this is stated by the independent certification of the partitions, since the underlying infrastructure (the partitioning kernel) is assumed to provide the proper temporal and spatial isolation.

It is a must to preserve certain properties of the resource allocation to the partitions so the partitioning kernel is able to guarantee isolation. As far as CPU allocation is concerned, it shall be guaranteed that when the workload of the system is modified the new assignation of processing power, required to cope with the modifications, does not impact in the partitions not involved in the modifications.

The problem to be addressed is concerned with the management of a plan in which the workload inside partitions changes.

There are several mechanisms that can be used to add new workload on an already generated plan:

1. Merge the previous and new workload definition to generate a new plan that will meet all the requirements. That is, to regenerate the new plan from scratch.
2. Move the position of the existing slots to accommodate the new workload.
3. Employ only the unused time to schedule the new workload.

If the goal is to preserve the temporal properties of the existing plan then building the new plan from scratch is not the right choice for obvious reasons. The scheduling algorithm will try to generate a good/optimal new plan (meet deadlines, guarantee precedence constraints, minimise response time, etc.) but the new plan may be quite different from the previous one. The same happens with the second mechanism.

As an example, let's consider 3 partitions. The generated plan at hypervisor level is the following:

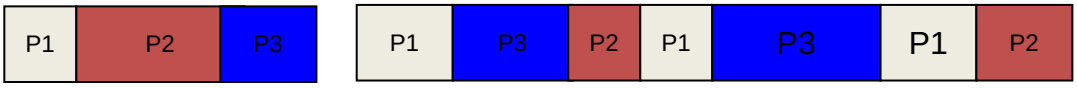

Figure 4-8 Generated plan at hypervisor level

Once the system has been certified, P3 is removed from the system and a new partition P4 is added. If the new plan is generated from scratch or if existing slots preserves duration but not position, it is mandatory to re-certify all partitions. However, if P1 and P2 schedules are respected, and P4 can be scheduled in P3 wholes (mechanism 3) it is only necessary to certify P4. Next figure shows a comparison of the three alternatives. The scheduling algorithm applied in mechanism 3 is what we call incremental scheduling, that is, the scheduling of a workload where there are already occupied slots by other partitions.

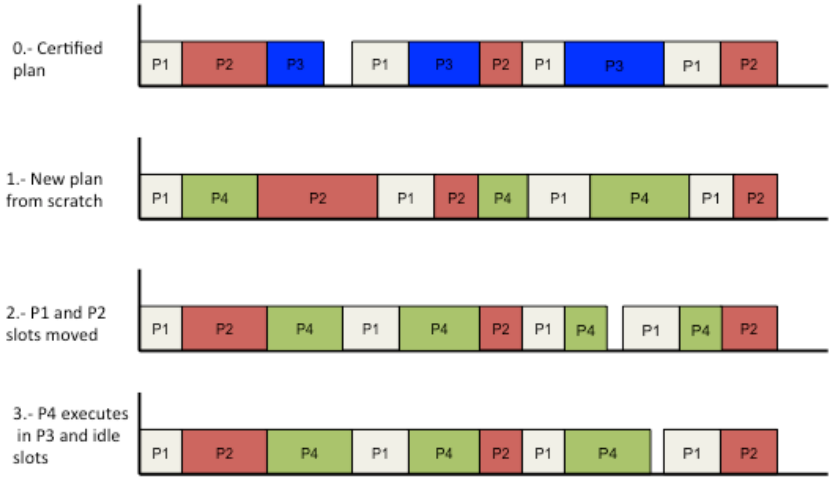

Figure 4-9 Comparison of alternative scheduling plans once P3 is removed from the certified scheduling plan

4.8.1 Related Work

There are no papers in the literature that directly addresses the aforementioned problem of the incremental scheduling in the exact terms that we have stated. However, there are papers that proposed scheduling strategies to use the idle time of a certified plan that can be used by other

partitions or new partitions. These papers can lay the foundations of the mechanism that uses the idle time combined with a scheduling strategy that respects other partition's slots.

Zabos et al. presented the integration of a spare reclamation algorithm into a middleware layer [Za09] that is placed on top of a real-time OS, but not underneath as a hypervisor.

In the context of the ACTORS EU project, an adaptive reservation-based multicore CPU management for soft real-time systems was developed, as well based on CBS and EDF [ARS11]. The IRIS algorithm can also be used for spare reclamation since it is a resource reservation algorithm that handles overload situations by spare bandwidth allocation among hard, soft, and non-real-time tasks [MSP04].

Lee et al. presented a compositional scheduling framework for the Xen hypervisor [LXC11]. Resource models are realized as periodic servers and enhancements to the server design in order to increase the resource utilization are introduced. Their work-conserving periodic server lets one lower-priority non-idle server benefit when a high-priority server idles. Their capacity reclaiming periodic server allows idle time of a server to be used by any other server. Close to this work, Groesbring et al [GAS14] propose an adaptive computation bandwidth management for mixed criticality multicore systems in an Hypervisor-based virtualization architecture which is compatible with a potential certification based on the guarantee of specified bandwidth minimums and the isolation of overruns of virtual machines.

Theis and Fohler present in [TF13] and algorithm to handle changes in criticality of tasks. The interest of this paper is that it leaves the existing schedule table unchanged, so there is no need for re-certification.

4.9 System Partitioning in MultiPartes

4.9.1 System Partitioning Algorithm and Scheduling

In MultiPARTES EU FP7 project, a component of the toolset generates automatically the partitioning of the system. As far as we know, apart from MPT, automatic partitioning algorithms for mixed-criticality multicore systems based in hypervisors has not been integrated into development toolsets in other FP7 projects. Therefore, although DREAMS DoW does not explicitly state that partitioning algorithms will be integrated in the tools to be developed in WP4, it may be interesting to check if reusing this experience in DREAMS is possible.

The partitioning consists of a set of partitions, and each partition has:

- a set of applications (functional parts) assigned
- an operating system
- a processor assigned
- resources assigned (CPU share, memory and other devices required by its applications).

The main requirements that the set of partitioning has to meet are the following:

- All applications (functional parts) must be allocated to partitions
- All restrictions of all partitions have to be met
- Resources assigned to partitions must not exceed available resources

As far as scheduling is concerned, XtratuM follows a cyclic executive policy, meaning that the scheduling is repeated cyclically, defining an execution slots to partitions. The size of the slot must take into account the required processing capability (described in the application model)

4.9.2 Input to the Partitioning Algorithm

The partitioning algorithm implemented in MPT by the Universidad Politecnica de Madrid (see SA13, SZ13 and http://www.dit.upm.es/~str/publicaciones.html for related work) takes as input the set of models composing the system:

- Platform model
- Application model
- Partitioning restriction model

One of the most interesting models is the restrictions model. This model contains the restrictions that resulting partitioning must comply. Restrictions can be divided into two groups:

- Explicit constraints: defined by developers and system integrator. For example, these constraints can define the specific hardware devices that an application must use, or can force specific allocations of applications to run into specific partitions defined by the user.
- Implicit constraints: are deduced automatically from the system model and are mainly intended to guarantee that non-functional requirements specified in the model are met. As an example, two applications with different criticality level cannot be in the same partition.

The toolset checks these constraints ensuring that partition meets the constraints and, therefore, are consistent with non-functional requirements specification. The toolset takes also into account the usage of resources and real-time conditions, checking that time requirements are coherent and that partitioning meets some necessary conditions, as for example, that deadlines are greater or equal that worst-case computation time.

4.9.3 Output of the Partitioning Algorithm

The output of the partitioning algorithm is what is called the deployment model which defines the system partitioning and description of each partition: allocated applications, operating system, required hardware resources.

4.9.4 Partitioning Algorithm

The algorithm is based in a “divide and conquer” strategy, an appropriate approach due to the complexity of the problem and the requirements for extensibility of the algorithm.

The problem solution process is divided into four stages:

- Allocation of application into partitions trying to minimise the number of partitions while ensuring that their restrictions are met.
- Allocation to partitions to processor cores. The result of this stage must meet the restrictions related with hardware devices.
- Cyclic plan scheduling design: as said before XtratuM relies on a cyclic scheduling that is *statically* defined. In this stage of the algorithm this plan is generated taking into account the allocation to cores and the CPU needs (all of them defined in the application model).
- Validation of the deployment model: the last step is to validate the resulting system partitioning with respect to general or non-functional requirements. This activity is performed by external tools that could be integrated in the toolset. As example, a response time analysis tool is to be used. Its aim is to ensure that time requirements are met by the proposed partitioning and scheduling plan.

Two initial stages are instances of a general allocation problem that is a well-known NP-Hard problem, which means that no known algorithm is able to solve it in a polynomial time. In the case of MPT, the algorithm implemented is a greedy algorithm of Iterated Register Coalescing (IRC) [GA96].

The IRC algorithm is based on the graph colouring theory. The allocation problem is modelled as a graph in which nodes are the functional parts to allocate, colours are the resources and arcs represent restrictions. This algorithm, 15 years after its first publication, is the base algorithm used in many research projects. See [Per08] for a survey on register allocation.

The IRC algorithm was originally conceived to help in the allocation of variables into hardware registers for code generation. There are a number of similarities such as the existence of a number of restrictions that must be followed. The algorithm was selected in MPT due to its good balance between the quality of the result and the implementation complexity.

The use of IRC for allocating applications on partitions required some changes to the algorithm. In the proposed solution, nodes represent applications, colours stand for partitions, and vertices are restrictions. The original IRC algorithm assumes a fixed number of resources (registers), however, in this allocation case, the number of resources (partitions) is not limited, and then, the proposed algorithm generates new colours when the allocation is not feasible or additional solutions are required.

The developed algorithm for the allocation of partitions to cores has also been adapted, with the aim of producing solutions where the cores workload is balanced.

Both algorithms have been improved with respect to the original IRC, in order to generate alternative solutions. When a proposed system partitioning at this point is invalid, this may be due to not being able of generating a feasible cyclic plan or by failing in the validation stage. Then, alternatives partitioning are generated, if feasible.

In MPT first working version of the two initial stages has been successfully tested with a number of system models. There is on-going work for performing a more exhaustive and systematic test of these algorithms..

4.10 Challenges/Shortcomings w.r.t. MCS

Real-time scheduling on mono-processor platforms is a mature discipline, with well-defined scheduling methods and schedulability analysis techniques that can be used to guarantee hard deadlines or to provide best-effort response time for tasks with soft deadlines or aperiodic tasks. All methods are supported by industrial-grade real-time operating systems and kernels, but the most widely used real-time scheduling method is fixed-priority pre-emptive scheduling, as it reaches a good compromise between simplicity, robustness, and efficiency. Alternatively, static scheduling may be a better choice for system with high integrity level requirements, as it provides fully deterministic time behaviour. On the other hand, dynamic priority methods may be preferable when efficiency and flexibility are required with not so high-level integrity requirements.

Temporal separation is fundamental to manage applications with different criticality levels. In this sense, standards such as ARINC-653 and separation kernels such as hypervisors assure temporal and spatial isolation. There is an important challenge in the generation of the scheduling in incremental systems, since no literature has been found that gives solution to this problem

4.11 References

[AB04] Abeni, L. and Buttazzo, G. (2004). Resource Reservation in Dynamic Real-Time Systems. Real-Time Systems, 27(2):123–167.

[ARINC653] Avionics Application Software Standard Interface (ARINC- 653), March 1996.

[Li00] J. Liu (2000). Real-Time Systems. Prentice-Hall.

[BW09] A. Burns, A. Wellings (2009). Real-Time Systems and Programming Languages. 4thed. Addison-Wesley.

[LSS87] J. P. Lehoczky, L. Sha, J. Strosnider (1987). Enhancing aperiodic responsiveness in a hard real-time environment.IEEE Real-Time Systems Symposium, RTSS 1987.

[SSL89] B. Sprunt, L. Sha, J. P. Lehoczky (1989). Aperiodic task scheduling for hard real-time systems.Real-Time Systems, vol. 1, no. 1.

[Bu10] G. Butazzo (2010). Hard Real-Time Computing Systems. 2nd ed. Springer.

[SB94] M. Spuri, G. C. Buttazzo (1994). Efficient aperiodic service under earliest deadline scheduling.IEEE Real-Time Systems Symposium, RTSS 1994.

[SL08] I. Shin, I. Lee (2008). Compositional real-time scheduling framework with periodic model. ACM Tr. Embedded Computing Systems, 7 (3), 1-39.

[SRL02] S. Saewong, R. Rajkumar, J. Lehoczky, M. Klein (2002). Analysis of hierarhical fixed-priority scheduling. Euromicro Conference on Real-Time Systems.IEEE Computer Society.

[DB05] R. Davis, A. Burns (2005). Hierarchical Fixed Priority Pre-Emptive Scheduling. 26th IEEE International Real-Time Systems Symposium (RTSS'05).IEEE Computer Society.

[BRC09] P. Balbastre, I. Ripoll, A. Crespo (2009). Exact Response Time Analysis of Hierarchical Fixed-Priority Scheduling. Real-Time Computing Systems and Applications (RTCSA'09).IEEE Computer Society.

[LB05] G. Lipari, E. Bini (2005). A methodology for designing hierarchical scheduling systems. Journal of Embedded Computing , 1 (2).

[EAL07] A. Easwaran, M. Anand, I. Lee (2007). Compositional Analysis Framework Using EDP Resource Models. IEEEReal-Time Systems Symposium — RTSS 2007.

[ZB07] F. Zhang, F. A. Burns, A. (2007). Analysis of Hierarchical EDF Pre-emptive Scheduling. IEEEReal-Time Systems Symposium.

[DB11] R.I. Davis, A. Burns (2011).A Survey of Hard Real-Time Scheduling for Multiprocessor Systems.ACM Computing Surveys, vol. 43, no.4.

[PUZ06] J. Pulido, S. Urueña, J. Zamorano, T. Vardanega, J.A. de la Puente (2006). Hierarchical Scheduling with Ada 2005. In L.M. Pinho, M.González-Harbour (eds.) Reliable Software Technologies — Ada-Europe 2006.Springer LNCS 4006.

[UPL08] S. Urueña, J. A. Pulido, J. López, J. Zamorano, J. A. de la Puente (2008). A New Approach to Memory Partitioning in On-board Spacecraft Software.Reliable Software Technologies — Ada-Europe 2008, Springer LNCS 5026.

[Ve07] S. Vestal (2007). Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance.28th IEEE Real-Time Systems Symposium RTSS'2007.

[BV08] S. K. Baruah, S. Vestal (2008). Schedulability analysis of sporadic tasks with multiple criticality specifications.20th Euromicro Conference on Real-Time Systems, ECRTS 2008.

[CRM10] A. Crespo, I Ripoll, M. Masmano (2010). Partitioned Embedded Architecture based on Hypervisor: The XtratuM approach. 2010 European Dependable Computing Conference.IEEE Computer Society.

[DL78] S. Dhall, C.L. Liu (1978). On a Real-Time Scheduling Problem.Operations Researchvol. 26, pp. 127-140.

[PM03] O. Pereira-Zapata, P. Mejia-Alvarez (2003). Analysis of Real-Time Multiprocessors Scheduling Algorithms.IEEE Real Time Systems Symposium.

[SB02] A. Srinivasan, S. Baruah (2002). Deadline-based scheduling of periodic task systems on multiprocessors. Inf. Process. Letters.84, 2.

[An08] B. Andersson (2008). The utilization bound of uniprocessor preemptive slack-monotonic scheduling is 50%. ACM Symp. on Applied computing —SAC '08.

[BCP96] S. Baruah, N. Cohen, C. Plaxton, D. Varvel(1996). Proportionate progress: A notion of fairness in resource allocation. Algorithmica, 15.

[Le94] S. K., Lee., (1994). On-line multiprocessor scheduling algorithms for real-time tasks.IEEE Region 10's Ninth Annual International Conference.

[FB05] N. Fisher, S. Baruah (2005).A Fully Polynomial-Time Approximation Scheme for Feasibility Analysis in Static-Priority Systems with Arbitrary Relative Deadlines.ECRTS 2005.

[Za09] A. Zabos et al., "Spare Capacity Distribution Using Exact Response time Analysis," in Proc. International Conference on Real-time andNetwork Systems (RTNS), 2009, pp. 97–106.

[ARS11] K. Arzen, Vanesa Romero, Stefan Schroo, Gerhard Fohler, "Adaptive Resource Management Made Real," in Proc. Workshop on Adaptive and Reconfigurable Embedded Systems (APRES), 2011.

[MSP04] Marzario, L. ; Scuola Sup., Pisa, Italy ; Lipari, G. ; Balbastre, P. ; Crespo, A. "IRIS: a new reclaiming algorithm for server-based real-time systems". 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), 2004.

[LXC11] J. Lee, S. Xi, S. Chen, L.T.X. Phan, C. Gill, I. Lee, C. Lu and O. Sokolsky. "Realizing Compositional Scheduling Through Virtualization," in Proc. Real-Time and Embedded Technology and Applications Symposium (RTAS), 2011, pp. 13–22.

[GAS14] Stefan Groesbrink, Luis Almeida, Mario de Sousa and Stefan M. Petters "Towards certifiable adaptive reservations for hypervisor-based virtualization" in Proc of the 20th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), 2014

[TF13] Jens Theis and Gerhard Fohler, "Mixed criticality scheduling in time-triggered legacy systems". 1st International Workshop on Mixed Criticality Systems, 2013

[SL08] I. Shin, I. Lee (2008). Compositional real-time scheduling framework with periodic model. ACM Tr. Embedded Computing Systems, 7 (3), 1-39.

[SRL02] S. Saewong, R. Rajkumar, J. Lehoczky, M. Klein (2002). Analysis of hierarhical fixed-priority scheduling. Euromicro Conference on Real-Time Systems.IEEE Computer Society.

[XP10] Xu, J., (2010). A method for adjusting the periods of periodic processes to reduce the least common multiple of the period lengths in real-time embedded systems. In: Mechatronics and Embedded Systems and Applications (MESA), 2010 IEEE/ASME International Conference on. pp. 288 –294.

[SGG03] Chi-Sheng Shih and Gopalakrishnan, S. and Ganti, P. and Caccamo, M. and Lui Sha. Scheduling real-time dwells using tasks with synthetic periods. Real-Time Systems Symposium, 2003.

[NS00]M. DiNatale and J. A. Stankovic. Scheduling distributed real-time tasks with minimum jitter. IEEE Trans. Computers, 49(4):303–316, 2000.

[AC00] P. Albertos, A. Crespo, I. Ripoll, M. Valle´s, and P. Balbastre. Rt control scheduling to reduce control performance degrading. In Proceedings of the 39th IEEE Conference on Decision and Control, 2000.

[CA05]A. Crespo, P. Albertos, K. Arzen, A. Cervin, M. Torgren, and Z. Hanzalek. Artist2 roadmap on real-time techniques in control system implementation. Technical report, Control for Embedded Systems Cluster. EU/IST IST-004527, 2005.

[KS00] T. Kim, H. Shin, and N. Chang. Deadline assignment to reduce output jitter of real-time tasks. In 16th IFAC Workshop on Distributed Computer Control Systems, 2000.

[KR93] M. Klein, T. Ralya, B. Pollak, R. Obenza, and M. G. Har- bour. A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems. Kluwer Academic Publishers, 1993.

[LH96] K. Lin and A. Herkert. Jitter control in time-triggered sys- tems. In Proceedings of the 29th Hawaii International Con- ference on System Sciences, 1996.

[L92] C. Locke. Software architecture for hard real-time applica- tions: cyclic executives vs. priority executives. Journal of Real-Time Systems, 4(1):37–53, 1992.

[GA96] George, L., Appel, A.W. *Iterated register coalescing*. TOPLAS 18(3), 300–324 (1996).

[Per08] Fernando Magno Quintañao Pereira. *A Survey on Register Allocation*. UCLA Compilers Group. http://compilers.cs.ucla.edu/fernando/publications/drafts/survey.pdf, October 2008

[SA13] E. Salazar, A. Alonso, M.A. de Miguel, J.A. de la Puente. ”A Model- Based Framework for Developing Real-Time Safety Ada Systems”. In H.B. Keller, et al (eds.), Reliable Software Technologies Ada-Europe, LNCS 7896, pp. 126–141. Springer-Verlag, 2013.

[ZE13] Juan Zamorano, Ángel Esquinas, Juan A.delaPuente. On real-time partitioned multicore systems.
*16th International Real-Time Ada Workshop — IRTAW16.*York, April 2013. Published in *Ada Letters*, **33**, 2, pp. 33--39. August 2013. ISSN: 1094-3641; DOI 10.1145/2552999.2553003.

[LL73] C. L. Liu, J. W. Layland (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM, 20, 1.

[LM82] J. Y. T. Leung, J. Whitehead (1982). On the Complexity of Fixed-Priority Scheduling of Periodic Real-Time Tasks. Performance Evaluation, 2, 4.

[ATB93] N. C. Audsley, K. Tindell, A. Burns (1993). The End of the Line for Static Cyclic Scheduling? *5th Euromicro Workshop on Real-Time Systems.* IEEE Computer Society.

5 Timing Analysis Algorithms

5.1 Overview

Timing analysis has been extensively studied over the last 40 years, however there are no results available for the mixed-criticality scheduling schemes that will be chosen in DREAMS both for tasks (section 4) and network (section 10). Further a compositional framework must guarantee the correct temporal operation of the composed DREAMS system. Finally the recovery strategies (section 6) also need to be addressed in terms of timing analysis.

5.2 Classical Approach for Timing Analysis

5.2.1 Monoprocessor

This section will present the schedulability analysis of scheduling algorithms presented in section 4.2.1 and 4.2.2.

5.2.1.1 Static scheduling

Schedulability analysis is performed at the time of building the scheduling table, i.e. at design time.

5.2.1.2 Priority scheduling

Schedulability analysis for fixed priority scheduling is generally based on response-time analysis (RTA) [JP86].[2] The worst-case response time of a task τ_i can be computed using the following formula:

$$R_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

where C_i is the worst-case execution time of a task job, B_i is the maximum blocking a job can incur due to priority inversion, T_j is the period (or minimum inter-arrival time) of task τ_j, and *hp(i)* is the set of tasks with a priority higher than τ_i.

Once response times have been computed for all tasks, the analysis is reduced to comparing each task response time with its specified deadline. Deadlines are guaranteed if

$$\forall i \; R_i \leq D_i$$

When the priorities are dynamic, for a simple computational model with only static, independent tasks (i.e. no communication), and deadlines equal to periods, all the deadlines are guaranteed as long as the processor utilization is less than 100%, i.e.

$$U = \sum \frac{C_i}{T_i} \leq 1$$

However, in practice this simple situation is seldom found, and arbitrary deadlines and inter-task communication must be accounted for, as in FPPS. A common test is based on processor demand criteria (PDF), which is based on a characterization of the load of the system at any time *t* (demand bound function h(t)):

$$h(t) = \sum_{i=1}^{N} \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor C_i$$

Quick processor-demand analysis (QPA) provides an efficient schedulability test based on calculating the values *h(t)* at selected points in time [ZB09]. The test is valid for a computational model with a

[2] Although some simple schedulability tests based on processor utilization have been widely publicized, they are valid only for very basic situations, which are seldom representative of real-life systems.

static set of periodic and sporadic tasks with arbitrary deadlines, scheduled with EDF, and using a static set of protected shared data objects accessed the SRP.

5.2.2 Multiprocessor

In the case of multiprocessor scheduling, finding an optimal scheduling algorithm depends on task deadlines and a priori knowledge of the temporal parameters. For example, optimal algorithms exist for completely determined jobs, that is, a task set where all the arrival times and execution times are known a priori. This means that optimality is not possible without clairvoyance [HL82]. Even with clairvoyance, optimal algorithms found in the literature are tractable for task sets for relatively short hyperperiods. On the other part, optimal algorithms are also known for task sets with deadlines equal to periods [Fi07].

Multiprocessor schedulability analysis is mostly based on *h*(t). Additionally, the processor load is the maximum value of the processor demand bound divided by the length of the time interval:

$$load(t) = \max_{\forall t}\left(\frac{h(t)}{t}\right)$$

As a task set cannot be schedulable according to any algorithm if the total execution released in the interval exceeds the processor capacity in the same interval, the following expression is a simple necessary condition for feasibility [FB05]:

$$load(t) \leq m$$

Where *m* is the number of processors.

Regarding partitioned scheduling, [OB98] showed that any system of independent periodic tasks with total utilization

$$U < m\left(2^{1/2} - 1\right)$$

can be scheduled on *m* processors using FFDU assignment of tasks to processors and RM local scheduling. They also showed that for any *m*≥2 there is a system of tasks with

$$U = (m+1)/\left(1 + 2^{1/(m+1)}\right)$$

that cannot be scheduled with m processors using any partitioning algorithm and RM local scheduling.

[LDG01] refined and generalized this result, showing that any system of periodic tasks with total individual utilizations $u_i \leq u_{max}$ and total utilization

$$U < (m\beta_{LLB} + 1)\left(2^{1/(\beta_{LLB}+1)} - 1\right)$$

can be scheduled on m processors using FFDU or any other "reasonable allocation decreasing" assignment of tasks to processors, where

$$\beta_{LLB} = \lfloor 1/\log_2(\mathrm{u}_{\max} + 1) \rfloor$$

They showed that this result is tight for "reasonable" partitioning schemes based on the Liu & Layland utilization bound, and claimed that the tightness extends to all other partitioning schemes.

5.3 Partitioned System Scheduling Analysis

According to the techniques explained in sections 4.2 and 4.6, this section introduces the schedulability analysis of the techniques explained in those sections.

5.3.1 Server-based Scheduling Analysis

The schedulability analysis of this kind of systems is based on the calculation of the worst-case response time when FPPS is the local scheduling policy. Several works give an estimation of this worst-case response time. Without entering into details, the first approximation is given by [SRL02]. Exact schedulability tests for fixed priority systems (where both the global and the local scheduler are fixed priority) for the periodic, sporadic and deferrable servers where presented in [DB05]. A correction to this work was presented in [BRC09]. [LB05] provide a kind of sensitivity analysis of the global level for a two-level hierarchical system, providing a methodology for calculating the domain parameters that make the task set feasible. Regarding the case when the local scheduler is based on dynamic priority, the results are focused on calculate the demand bound function (see next section). In this case, [EAL07] proposed a necessary but not sufficient feasibility condition. The worst-case response time of a task in a hierarchical system was calculated in [ZB07] but the resulting formulation was very complex and difficult to obtain in a reasonable amount of time for an on-line algorithm.

5.3.2 Compositional Scheduling Analysis

The schedulability analysis of a compositional system is based on the demand bound function $h_\tau(t)$ (where τ represents the subsystem task set) as defined in [RCM96]. This function represents the amount of computation time that has been requested by all the activations whose deadline is less than or equal to the function's argument of the tasks belonging to subsystem τ.

The subsystem can be modeled as a single periodic task (interface task, Φ) such that the demand bound function of this interface task $h_{\Phi\Phi}(t)$ is a upper bound of $h_\tau(t)$. This task acts as a periodic server. If the periodic server is described as a budget, deadline and period it is called an "Explicit Deadline Periodic" (EDP) resource server.

The curve in figure 6.3 represents graphically the schedulability analysis of a compositional system. The demand bound function is compared with the supply bound function (sbf). The sbf represents the minimum amount of a resource that is guaranteed by a resource model for a subsystem. As the resource is the processor, sbf provides the available execution units than can be allocated to the subsystem. It is obvious that the demand cannot be greater than the offer, so the schedulability condition is that $h(t) \leq sbf(t)$ for each interface Φ. In the figure, even when there exists a point in which both functions have the same value, the system is schedulable.

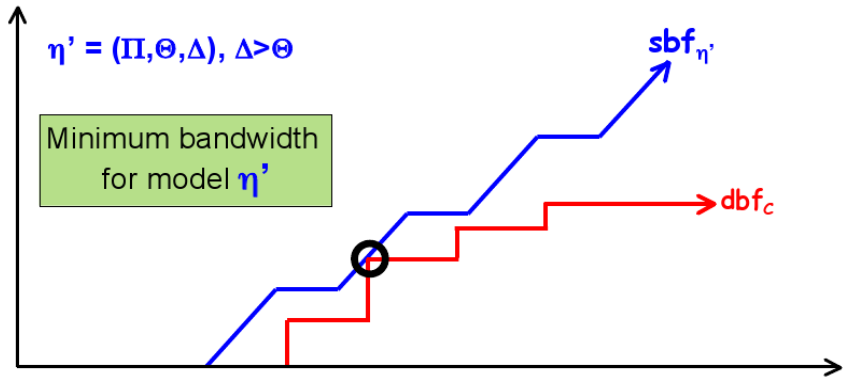

Figure 6.3. Demand (dbf) and supply (sbf) bound functions.

5.3.3 Flat Model Schedulability Analysis

In this kind of systems, all tasks of all partitions are treated as in a traditional task set. Therefore, traditional scheduling policies such as RM, DM or EDF are applied as well as its own scheduling analysis.

5.4 Task Timing Analysis

Timing analysis has been extensively studied over the last 40 years; however there might not be all results available for the mixed-criticality model that will be chosen in DREAMS. Amongst the different techniques used so far we can mention the following ones which are to be considered in the DREAMS context:

- For priority-driven scheduling algorithms: worst-case response time analysis based on the trajectory model [Migge1999] or demand-bound functions.
- For static- scheduling algorithms: worst-case response time analysis by analysis of the schedule table(s) [Monot2012].
- Hierarchical scheduling timing analysis using compositional frameworks [Anand2013] or simpler approach such as “layered priorities” [Migge99].

All these techniques make the following assumptions:

- Task may have mixed criticality levels (can a partition mix tasks of different criticalities?)
- Tasks may have precedence constraints, e.g. be modeled as a classical Directed Acyclic Graph (DAG).
- Tasks may have offsets constraints.
- Tasks may share resources.
- Scheduling can be preemptive or non-preemptive.
- Scheduling can be static-cyclic or priority-driven.
- Scheduling can be hierarchical: scheduling among the partitions at the hypervisors levels, and scheduling among the tasks of a partition at the virtual machine level.
- Multiprocessor scheduling algorithms are partitioned (i.e., not global): a task is statically assigned to a processor and cannot migrate to another processor at run-time.
- The workload submitted to the computing resources is bounded in any time interval, and it can be described by a deterministic work-arrival function (e.g. stair-cases for the periodic task model).

The algorithms are usually of pseudo-polynomial complexity but can reach exponential complexity level. In this case approximations are required to find a balance between the computing time and the WCRT accuracy.

5.5 Message Timing Analysis

The distribution of DREAMS application over network requires message worst-case traversal time (WCTT) analysis and simulation. Mixed-criticality scheduling, at the communication network level in the DREAMS architecture context might impose additional constraints that existing analysis may not be able to cope with. In the existing analysis we foresee to use the following in the DREAMS context: Network-Calculus [Boyer11] for the rate-constrained and best-effort parts and the communication schedule analysis for the time-triggered part [Steiner11]. Experiments [Boyer12] show that the max. Pessimism is typically less than 15%. Analysis results can be cross-checked using simulation.

5.6 End-to-End Timing Analysis

As seen in previous sections, timing analysis formalisms for the different resources can be heterogeneous: network-calculus for the network and time-triggered or classical fixed priority preemptive analysis for the scheduling of tasks. Still it is required to provide bounded end-to-end delays at system level for a functional chain. State of the art system level timing analyses rely on the event-streams as described in [Henia2005] or the more general form described in [Albers2008]. Event streams define precise and understandable interfaces to interconnect existing analyses. In the context of DREAMS, the diversity of the scheduling paradigms, the scale of the system envisaged and mixed-criticality issues goes well beyond the complexity of the examples treated in the literature.

5.7 Mode Change

A system can be schedulable in every mode, but not schedulable during a mode change. This must be paid attention to in the DREAMS context. Mode changes can be both in the periods and execution/transmission times of the tasks/messages, as well as changes in the criticality levels of the tasks/messages. No suitable technique ready out-of-the-box could be identified for DREAMS, existing mode-change protocols (see [Burns13] for a starting point) should be adapted to the execution platform of DREAMS.

5.8 Temporal Interference in Multicore Systems

5.8.1 Sources of Indeterminism

The success of embedded system verification depends greatly on the capacity to determine the exact behavior of the system. The complexity of the hardware is not the only barrier as hardware integrators often release only documentation of certain parts considered important. Precise timing estimation is a key factor and it's influenced by the predictability of the hardware architecture.

Precise determination of the WCET is a challenging task even in monocore architectures. The problems are accentuated in the multicore context mainly due to the resource sharing that can lead to highly complex interactions or to indeterminism. Modern features already present in safety-critical embedded systems make it impossible to compute the exact WCET. The possibility to determine an upper bound for the system is crucial in hard real time systems.

Most of the units that generate behaviors that are hard to take into account can be deactivated, but it's not always easy to predict the impact on the performances.

These sources of indeterminism are:

- Internal architecture aspects

- Cache management (L2 and L3 caches mainly)
- Shared memory accesses
- Shared controller units
- Bus arbitration
- DMA management

When WCET estimation of tasks running on multicore platforms is possible, a consequent large safety margin should be considered (given the complex heuristics used, the data dependency and the resource sharing).
The precision of WCET estimation is influenced by the choice of the analysis tool, but more intrinsically by the implementation of heavy architecture and execution optimization heuristics. Of particular interest is the multicore context, as a resource sharing example and more precisely the bus sharing. The WCET estimation in multicore has different levels of difficulties.
The first one is inherited from the single core world and comes from the complexity of implemented optimization strategies that can eventually invalidate a feasibility test for a given platform that has to be submitted to demanding certification standards. Certain modern features in processors cannot be precisely analysed, which drives some processors to a point of impossibility of certification. Other features generate imprecision that will increase the safe margins needed to take in order to comply with the safety constraints.
The second one is introduced by the architecture of multicore and can lead to the impossibility to determine the WCET. State of the art works deal with this issue either by constraining some platforms, or by handling only a part of the issues and giving some new architectural workarounds that are custom tailored for some applications ([GSY09], [HPP09], [HP08], [LSL09], [ZY09], [KFM11], [CR11]). Another approach is to gather best practices for future multi-core architectures [CFG10], after acknowledging that analysing current multicore architectures is impossible in general. Nevertheless a unified and detailed approach is yet to be available.
The WCET estimation consists of two main steps, namely the control-flow analysis (also called high-level analysis or path analysis) that determines the (un)feasible paths in a program, and the processor-behavior analysis (also known as low-level analysis or hardware modeling), that evaluates instruction timing.

5.8.2 Modeling and Analysis

A precise WCET analysis must take into account all the details of the hardware that can influence the timing of the executed code. Nevertheless, the code has also an impact on the timing and therefore it has to respect certain constraints (like the termination) before it can be deployed in hard real time systems. Furthermore the code activates hardware operations in combinations that can be more or less easy to analyse. Therefore controlling the code through a platform-aware compilation process can prove useful in order to ensure the respect of timing delays of a task on a given hardware.
Timing anomalies inside a given core can influence the WCET estimation of multi-cores. A timing anomaly occurs when a local worst case contributes to the global favorable case (in our case WCET-wise). In the case of multi-core, such an example is a cache miss on one core that generates a series of cache hits on the other and the other way around. Several types of timing anomalies exist. Some are inherent to instruction execution order and are generally caused by greedy scheduler that will change the instruction execution order causing inversion or amplification of the execution time difference. Others are caused by parallel decomposition and divide and conquer approaches to WCET estimations. As the first ones cannot be avoided, the others may prove essential for the possibility to construct an efficient processor behavior analysis that does not need to search the whole state space for the whole program at once.
Puschner et al. [KKP10] formalize the different types of timing anomalies and present cases when the parallel timing anomalies can lead to the underestimation of the WCET with parallel composition. In [KKP10] Puschner et al. also described a solution to take into account timing anomalies in general. The method uses compilation techniques and modifies the binary by

instruction injection in order to avoid timing anomalies. The main idea is to interfere with the prefetching stage and ensure that we start with an empty or flushed prefetching window hence there is never an excess instruction waiting to be executed.
The impact of the shared cache is high and global and gets amplified in the context switch case. Modeling the behavior of shared caches between cores is practically impossible because of the possible interactions between concurrent threads running on different cores [S09]. When using a shared cache with parallel programs running on the multi-core processor, a cache-coherency mechanism must be implemented. The WCET analysis of such systems must calculate the worst-case delay caused by maintaining the cache coherence between different cores. Furthermore, resource contention and inter-thread conflicts among the program threads should be considered. Under the assumption that the bus strategy can be statically analyzed, the second level of cache can be made predictable by partitioning the L2 cache for each core [SH12].

5.9 Challenges/Shortcomings w.r.t. DREAMS

A system can be schedulable in every mode, but not schedulable during a mode change. This must be paid attention to in the DREAMS context. Mode changes can be both in the periods and execution/transmission times of the tasks/messages, as well as changes in the criticality levels of the tasks/messages. No suitable technique ready out-of-the-box could be identified for DREAMS, existing mode-change protocols (see [Burns13] for a starting point) should be adapted to the execution platform of DREAMS.

5.10 References

[Fi07] N. Fisher (2007). The Multiprocessor Real-Time Scheduling of General Task Systems.PhD Thesis, Univ. North Carolina at Chapel Hill.
[FB05] N. Fisher, S. Baruah (2005).A Fully Polynomial-Time Approximation Scheme for Feasibility Analysis in Static-Priority Systems with Arbitrary Relative Deadlines. ECRTS 2005.
[HL82] K.S. Hong, J.Y. Leung (1982). On-line scheduling of real-time tasks. IEEE Tr. Computers, 41.
[Le94] S. K., Lee., (1994). On-line multiprocessor scheduling algorithms for real-time tasks. IEEE Region 10's Ninth Annual International Conference.
[LSS87] J. P. Lehoczky, L. Sha, J. Strosnider (1987). Enhancing aperiodic responsiveness in a hard real-time environment. IEEE Real-Time Systems Symposium, RTSS 1987.
[LB05] G. Lipari, E. Bini (2005). A methodology for designing hierarchical scheduling systems. Journal of Embedded Computing , 1 (2).
[Li00] J. Liu (2000). Real-Time Systems. Prentice-Hall.
[LDG01] J. M. Lopez, J. L. Diaz, D. F. Garcia (2001). Minimum and maximum utilization bounds for multiprocessor RM scheduling. In Proc. 13th Euromicro Conf. Real-Time Systems, pages 67–75, Delft, Netherlands, June 2001.
[OB98] D. I. Oh, T. P. Baker (1998). Utilization bounds for N-processor rate monotone scheduling with stable processorassignment. Real Time Systems, 15(2):183–193, September 1998.
[KKP10] Avoiding Timing Anomalies using Compile-Time Instruction Scheduling and Instruction Insertion, A. Kadlec, R. Kirner and P. Puschner, 13th IEEE ISORC, 2010.
[CR11] Scalable and Precise Refinement of Cache Timing Analysis via Model Checking, Sudipta Chattopadhyay and Abhik Roychoudhury, Real-Time Systems Symposium (RTSS '11), 32nd edition, 2011.
[KFM11] Bus-Aware Multicore WCET Analysis through TDMA Offset Bounds, Timon Kelter, Heiko Falk, Peter Marwedel, Sudipta Chattopadhyay, Abhik Roychoudhury, · ECRTS '11 Proceedings of the 2011 23rd Euromicro Conference on Real-Time Systems, 2011.

[ZY09] Accurately Estimating Worst-Case Execution Time for Multi-core Processors with Shared Direct-Mapped Instruction Caches, Wei Zhang, Jun Yan, RTCSA '09, Embedded and Real-Time Computing Systems and Applications, 2009.

[LSL09] Timing Analysis of Concurrent Programs Running on Shared Cache Multi-Cores, Yan Li, Vivy Suhendra, Yun Liang, Tulika Mitra, Abhik Roychoudhury, RTSS'09, 30th edition of the IEEE Real-Time Systems Symposium, 2009.

[HP08] WCET analysis of multi-level non-inclusive set-associative instruction caches, Damien Hardy Isabelle Puaut, RTSS '08, Proceedings of the 2008 Real-Time Systems Symposium, 2008.

[HPP09] Using Bypass to Tighten WCET Estimates for Multi-Core Processors with Shared Instruction Caches, Damien Hardy, Thomas Piquet, Isabelle Puaut, RTSS 2009. 30th edition of the IEEE Real-Time Systems Symposium, 2009.

[GSY09] Cache-Aware Scheduling and Analysis for Multicores, Nan Guan, Martin Stigge, Wang Yi, Ge Yu, EMSOFT '09 Proceedings of the seventh ACM international conference on Embedded software, 2009.

[CFG10] Predictability Considerations in the Design of Multi-Core Embedded Systems, by Christoph Cullmann , Christian Ferdinand , Gernot Gebhard, Daniel Grund, Claire Maiza (Burguière), Jan Reineke, Benoît Triquet, Reinhard Wilhelm, ERTSS'10, Embedded Real Time Software and Systems, 2010.

[S09] Time-Predictable Cache Organization, M. Schoeberl, Future Dependable Distributed Systems, 11-16, 2009.

[SH12] Data cache organization for accurate timing analysis, M. Schoeberl, B. Huber, W. Puffitsch, Real-Time Systems, Springer US, 0922-6443, June 2012.

[JP86] M. Joseph, P.K. Pandya (1986). Finding Response Times in Real-Time Systems. *BCS Computer Journal*, 29, 5.

[ZB09] F. Zhang, A. Burns (2009). Schedulability Analysis for Real-Time Systems with EDF Scheduling. *IEEE Trans. Computers*, 58, 9.

[SRL02] S. Saewong, R. Rajkumar, J. Lehoczky, M. Klein (2002). Analysis of hierarhical fixed-priority scheduling. *Euromicro Conference on Real-Time Systems.* IEEE Computer Society.

[DB05] R. Davis, A. Burns (2005). Hierarchical Fixed Priority Pre-Emptive Scheduling. *26th IEEE International Real-Time Systems Symposium (RTSS'05).* IEEE Computer Society.

[BRC09] P. Balbastre, I. Ripoll, A. Crespo (2009). Exact Response Time Analysis of Hierarchical Fixed-Priority Scheduling. *Real-Time Computing Systems and Applications (RTCSA'09).* IEEE Computer Society.

[LB05] G. Lipari, E. Bini (2005). A methodology for designing hierarchical scheduling systems. *Journal of Embedded Computing , 1* (2).

[EAL07] A. Easwaran, M. Anand, I. Lee (2007). Compositional Analysis Framework Using EDP Resource Models. *IEEE Real-Time Systems Symposium — RTSS 2007*.

[ZB07] F. Zhang, F. A. Burns, A. (2007). Analysis of Hierarchical EDF Pre-emptive Scheduling. *IEEE Real-Time Systems Symposium*.

[RCM96] I. Ripoll, A. Crespo, A. Mok (1996). Improvement in feasibility testing for real-time tasks. *Journal of Real-Time Systems , 11* (1).

[Davis2014] R.I. Davis "A Review of Fixed Priority and EDF Scheduling for Hard Real-Time Uniprocessor Systems ". ACM SIGBED Review - Special Issue on the 3rd Embedded Operating Systems Workshop (Ewili 2013). , vol. 11, n° 1, pages 8-19, Feb 2014

[Davis2011] R.I. Davis and A. Burns "A Survey of Hard Real-Time Scheduling for Multiprocessor Systems." ACM Computing Surveys, 43, 4, Article 35, October 2011.

[Monot2012] Monot, N. Navet, B. Bavoux, F. Simonot-Lion, "Multi-source software on multicore automotive ECUs - Combining runnable sequencing with task scheduling", IEEE Transactions on Industrial Electronics, vol 59, n°10, pp 3934 - 3942, 2012.

[Migge1999] J. Migge, "Scheduling of recurrent tasks on one processor: A trajectory based Model," Phd Thesis from the University of Nice Sophia-Antipolis, 1999.

[Anand2013]Madhukar Anand, Sebastian Fischmeister, Insup Lee, “A comparison of compositional schedulability analysis techniques for hierarchical real-time systems”, ACM Trans. Embed. Comput. Syst., pp 1-37, vol. 13, n°1, 2013.

[Boyer2011] M. Boyer, N. Navet, J. Migge, "A simple and efficient class of functions to model arrival curve of packetised flows", First International Workshop on Worst-case Traversal Time (WCTT), in conjunction with the 32nd IEEE Real-time Systems Symposium (RTSS), Vienna, Austria, November 29, 2011.

[Steiner2011] W. Steiner, "Synthesis of Static Communication Schedules for Mixed-Criticality Systems," Proceedings of the 1st IEEE Workshop on Architectures and Applications for Mixed-Criticality Systems (AMICS 2011), IEEE Computer Society, 2011.

[Boyer2012] M. Boyer (Onera), N. Navet, M. Fumey (Thales Avionics), “Experimental assessment of timing verification techniques for AFDX“, Embedded Real-Time Software and Systems (ERTS 2012), Toulouse, France, February 1-3, 2012.

[Henia2005] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, R. Ernst, "System level performance analysis - the SymTA/S approach," Computers and Digital Techniques, IEE Proceedings - , vol.152, no.2, pp.148-166, Mar 2005.

[Albers2008] K. Albers, F. Bodmann, F. Slomka, ”Advanced Hierachical Event-Stream Model”, ECRTS 2008, pp211-220, 2008.

[Flemmin2013] T. Fleming and A. Burns. Extending mixed criticality scheduling. In Proc. WMC, RTSS, pages 7–12, 2013.

[Baruah2011] S.K. Baruah, A. Burns, and R. I. Davis, “Response-time analysis for mixed criticality systems”, In IEEE Real-Time Systems Symposium (RTSS), pages 34–43, 2011.

[Burns2013] Burns and R. Davis, “Mixed Criticality Systems - A Review”, 2013. Available at www-users.cs.york.ac.uk/~burns/review.pdf

6 Real-Time Faults and Recovery Strategies

6.1 Introduction

“Dependability aims at assessing the potential risks, foreseeing the failure occurrences and minimising the consequences of the catastrophic situations when occurring” [Vill92]. Dependability can be called the science of failures: it includes their knowledge, their assessment, their prediction, their measure and their control. Jean-Claude Laprie [Lap85] has formalized the taxonomy defining the dependability. The dependability [ALR04] [HM01] is composed of three elements:

1. Attributes: a way to assess the dependability of a system. Those attributes are known as RAMSS for reliability, availability, maintainability, safety and security;
2. Threats: an understanding of the things that can affect the dependability of a system. The threats are expressed as a succession of faults → errors → failures;
3. Means: ways to increase the dependability of a system. The main manners to improve the dependability are: fault prevention, fault removal, fault forecasting and fault tolerance.

For the DREAMS project, the dependability attribute that is considered is *safety*. Safety addresses the analysis of the failures that can lead to significant damages to the system, the environment and a risk of death or injuries. As a consequence, safety critical applications must fulfill safety

requirements in order to support hardware failures while ensuring the rare occurrence of catastrophic consequences. These requirements can in particular be expressed as quantitative objectives (such as the minimal probability required to lose the application) or qualitative (such as the minimal number of failures required to lose the application).

We therefore want to ensure safety requirements for application executing on the DREAMS platform which is composed of several multi-many-cores connected via a TT-Ethernet network. The avionic demonstrator in WP6 will highlight the capacities of the DREAMS platform to offer the expected level of safety for the avionic application, the FMS (Flight Management System) the DAL (Design Assurance Level) of which is C. The DAL is the level of rigor of development assurance activities performed on the applications and the (software, hardware) items. This notion has been defined in the EUROCAE ED-79 / SAE ARP 4754 (Aerospace Recommended Practice - Guidelines For Development Of Civil Aircraft and Systems). The requirements of the FMS application are defined in WP6 and among them, we must ensure the following failure conditions:

- The non respect of the internal deadlines is MAJ (Major, less than 10^{-9} failures / flight hour)
- The loss of the FMS is HAZ (Hazardous, 10^{-7}/flight hour)

We must identify the combination of failures leading to each failure condition and we must show that the FMS executing on the DREAMS platform respects the safety objectives. For this purpose, the first work is to identify the possible failures of the DREAMS platform. The second work consists in providing fault-tolerance mechanisms to support failures and prove that the safety objectives are fulfilled.

6.2 Fault model

We collect the fault model found in the literature.

6.2.1 Hardware Failures on the Multi-core

Permanent faults cause non-recoverable device defects. Hence, they result in hard errors and have an impact on the system's lifetime. The massive integration of small devices onto single chips increases the susceptible permanent failures due to various phenomena such as aging, wear-out or infant mortality [Bor05]. The fault-tolerance strategies presented in Section 6.3 focus on the independent occurrence of permanent faults in the processing elements of the underlying platform. Hence, they assume the application of appropriate processing-element wise containment of faults, provided by physically independent processor cores, and/or *Time-Space Partitioning* TSP mechanisms. For permanent faults, the basic reliability models at the component-level can be derived from an analysis of the physical failure mechanisms, resulting in empirical models (e.g., [DGB95; GCJ01]). Recent work [XCD10] proposes a framework that integrates device, component and system level models. The most commonly used metric to consider permanent faults is Mean-Time-To-Failure (MTTF).

Transient faults do not fundamentally damage a device, but may affect the application execution, e.g. due to external events such as cosmic particles striking a circuit. The results include corruption of the execution (e.g., segmentation errors, "hanging" applications, etc.), and the delivery of incorrect results. In IEC 61508-4 terminology, this type of faults results in so-called "soft-errors", that can be removed by rewriting correct data to the circuit. For transient faults, a classic model is to assume that they occur according to a Poisson law with constant error rate [SW89]. This reliability model has also been extended to cover the effects of voltage scaling on reliability [ZAZ09; ZA06]. While the Poisson fault model is used in many approaches (e.g., [ASE11; GK09; SW89; ZA09]), it assumes transient faults to be independent events. It is important to note that this choice prevents the consideration of Common-Model-Failures (CMF), which would violate the independence assumption due to the possible correlation of faults. When transient faults (or soft errors) are considered, the system level reliability is typically described by the failure probability or Failure-In-Time (FIT).

Memory regions such as register/cache/DDR can be corrupted by a single event upset (SEU) [SRP13]. If an electro-magnetic ray hits a memory cell, the data stored in the area might be corrupted because some bit is flipped. In this case, the value is changed in unforecast way. If the data is never used, the fault remains dormant, but, if the erroneous value is used, this could lead to an incorrect result or run-time error (e.g. division by zero mentioned above). A corruption on a pipe register can affect (1) the operation (via OPCODE data), (2) the result (via operand registers or during write back) or (3) the control flow (via CP).

One possibility to catch faults are hardware traps. A hardware trap ([Hyde10], XtratuM doc) is caused by an exceptional condition (e.g., breakpoint, division by zero, invalid memory access). This type of fault is detected by the hardware and XtratuM can inform the user of the occurrence. Such a fault can result of a bug in the design or an unexpected modification in the memory of the application.

Intermittent faults represent malfunctions of the device that appear and disappear repeatedly. In multi-core platforms, such faults [WKS08], caused in part by manufacturing, thermal, and voltage variations, can generate regular bursts that may last from several cycles to several seconds. This will result in temporarily suspended execution on a core during these faulty burst periods. The analysis summarized in Section 6.3.2.2 will not distinguish between intermittent faults and permanent faults, since it computes the probability that a single iteration of a periodic application is executed correctly.

6.2.2 Temporal Faults

Another source of problem comes from unexpected temporal behaviors of the applications on the platform. In embedded systems, applications agree to respect temporal contracts, which can be expressed in several ways: e.g., bandwidth on a network, pre-defined execution slots (e.g., in an IMA schedule). The application may encounter temporal faults that force the application to violate the initial contract.

In [But97], the chapter 8 focuses on the *overload conditions,* that are "*critical situations in which the computational demand requested by the task set exceeds the time available on the processors, and hence not all tasks can complete within their deadlines*". Such a situation can result for several reasons such as environmental solicitations, fault of peripheral devices (e.g. babbling idiot fault) or system exceptions.

In the DREAMS project, we consider many/multi-core systems that have a dynamic difficult-to-predict behavior. They include schemes that focus on enhancing the average performance, such as cache memories and branch predictors that take decisions dynamically, rather than on providing predictable behaviors that guarantee a safe and tight estimation of the WCET. In addition, several resources are shared and the concurrent accesses to these resources introduce timing variations, e.g. concurrent requests in the communication network and in the memory hierarchy. However, static WCET analysis tools are able to deliver reliable results in a so-called isolation mode that considers the case when a given task runs in sequence with no other concurrent tasks. When parallel execution is allowed, no complete assurance of the reliability of the worst-case execution time is possible. For the multi-core COTS platform Freescale T4240 that is used in the DREAMS WP6 work package, static WCET analysis tools, such as Ait or OTAWA, support this isolation mode, and can therefore be used to obtain the required results under the above assumptions.

6.2.3 Fault Model for the DREAMS Project

We will consider the following failures:

a) A core is halted
b) Temporal overload situations due to low criticality tasks accessing shared resources

6.3 Fault-tolerant Strategy

Once the fault model has been identified, the designer imagines, for each application, an architecture (or implementation) which fulfils the required level of safety by integrating redundancy

and safety mechanisms. One of the means for designing such architecture is to use *fault-tolerant* mechanisms. Fault-tolerance is the ability for a system to continue to operate while faults have occurred (detected or eventually not) [ALR04] [HM01]. It is not possible to tolerate all possible faults, since adding safety mechanisms increases system overhead, complexity, and validation challenges. IEC 61508-4 defines fault-tolerance as "the ability of a functional unit to continue to perform a required function in the presence of faults or errors" [IEC01]. In other words, fault-tolerance aims at reducing the probability of failure despite the presence of faults or errors. An important building block in this process is the estimation of a system's reliability.

In general, any fault-tolerance approach consists of the following two main steps:

a) Detection of an error situation,
b) System recovery in order to reach a correct state.

For fail-safe systems, faults that are only detected but not corrected are considered as harm-free, since the system can execute a safe shut-down. In contrast, both detectable and undetectable faults are considered as failure in fail-operational systems. Therefore, the analysis summarized in Section 6.3.1 considers Detectable Unrecoverable Faults (DUFs) and Silent Data Corruption (SDC).

The Avionics Handbook, in the chapter 28 dedicated to fault-tolerant avionics, [HM01], identifies fault-tolerant elements, which in general should appear in some form in any fault-tolerant system:

- *Error Detection*: recognition of the incidence of a fault.
- *Damage Assessment*: diagnosis of the locus of a fault.
- *Fault Containment*: restriction of the scope of effects of a fault.
- *Error Recovery*: restoration of a restartable error-free state.
- *Service Continuation*: sustained delivery of system services.
- *Fault Treatment*: repair of fault.

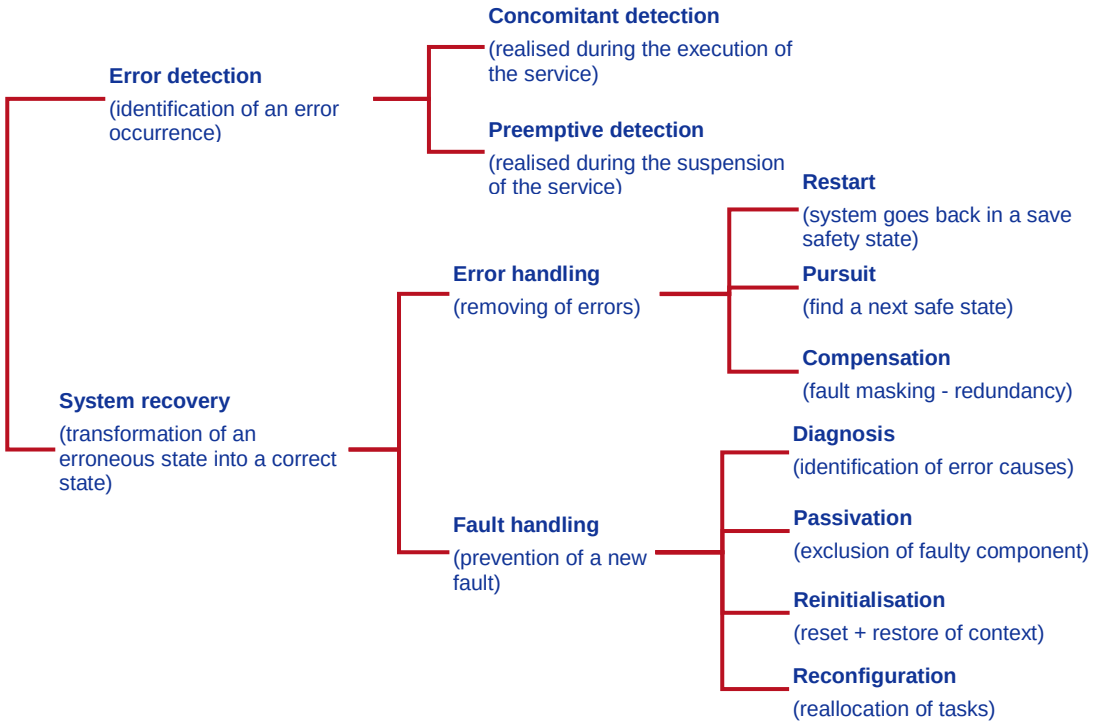

Figure 6-1 Fault-tolerance tree [Jean-Claude Laprie 90]

6.3.1 Recovery Strategy for the Fault Model: Transient Fault

The analysis summarized in this section assumes that only the current task is affected by a transient fault, and that subsequent tasks can be executed normally after a recovery process. Moreover, the approach assumes that the system software responsible for the recovery process (e.g., OS) is fault-free. In the following, it will be argued why simple fault-models such as the Poisson model mentioned above are an adequate choice for the analysis and fault-tolerance mechanisms summarized in this section. While these fault models do not allow the consideration of CMF (see above), the application of active redundancy (as suggested in the approach presented in this section) is not a solution to CMF (as pointed out by [MSM00]). While it is possible to consider CMFs in the reliability analysis, e.g. using the techniques presented in [MSM00], the real problem is to estimate the probability of CMF. For this reason, a lot of research effort has been devoted to CMF avoidance. The approach summarized in this section assumes that these techniques (e.g., design diversity or architectural-level fault-containment) have systematically been applied.

Similar to fault models, also fault-tolerance mechanisms can be applied at different system levels. As pointed out in [MWE03], one possibility is to consider fault-tolerance at the architectural level in order to reduce the probability of fault-to-error transition [MWE03]. The analysis summarized in this section follows the second alternative, i.e., the application of fault-tolerance techniques at the application-level in order to reduce the error-to-failure transition [MWE03], which can typically be achieved using active redundancy. Here, the replication of components enables to tolerate certain faults due to the availability of replica, which can be implemented both in the space and the time domain [Fo97].

In the following, a reliability analysis for time-triggered scheduling policies TT-SP and TT-FS (see Section 11.1.1) under the impact of transient faults will be discussed. Permanent faults can be considered using an extended encoding technique (see Section 6.3.2.2). Both scheduling policies have in common that hardware and software redundancy must be supported at the same time. Since shared time slots are possible, the actual utilization of a slot depends on the execution status of previous slots. Hence, system-level reliability analysis supporting this type of schedules requires investigating all combinations of faults tolerable by a given schedule. In the following, a corresponding tree-based analysis will be summarized. Here, a combination of faults occurring in a system is described by a so-called fault scenario, which is defined as follows: A fault scenario is a vector $\mathbf{x} = \{x_0, x_1, \cdots, x_n\}$, where $x_i \in \{1, 0, \mathrm{NA}\}$ models the execution result for each scheduling slot s_i (1 = successful execution, 0 = execution failure, NA = slot is unused). For a given job, a fault scenario $\mathbf{x}$ is tolerable by a schedule if it is executed correctly despite the presence of the faults specified in $\mathbf{x}$. Here, the working set $W(S,J)$ of a job J is defined as the set of fault scenarios that is tolerable by a schedule S. Since the members of $W(S,J)$ are disjoint, the probability that the job J is executed correctly can be computed as follows:

$\Pr(S,J) = \sum_{\boldsymbol{x} \in W(S,J)} \Pr(\boldsymbol{x})$.

Hence, the computation of the reliability under the aforementioned assumptions is split into the following two sub-problems:

- Computation of the working set $W(S,J)$.
- Computation of the success probability $\Pr(\boldsymbol{x})$ of a single fault scenario $\boldsymbol{x} \in W(S,J)$.

In the following, the computation of the working set $W(S,J)$ using a Binary Tree Analysis (BTA) will be illustrated using the example in Figure 6-2.

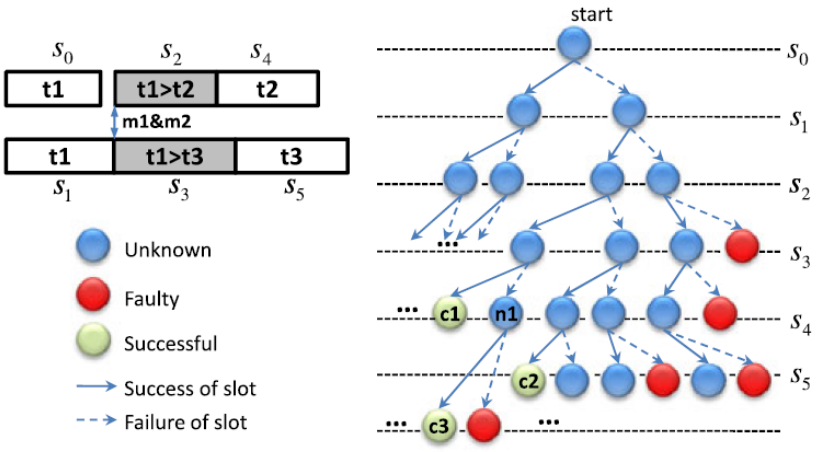

Figure 6-2 Binary Tree Analysis (Example)

In the tree, the i[th] level is associated with the i[th] scheduling slot. Edges leaving a node in the i[th] level represent the execution result of that slot, where left branches (solid lines in Figure 6-2) represent the case that the slot executes some task correctly, and right branches (dashed lines in Figure 6-2) represent a slot with failed execution. In case all tasks are either not ready or have finished earlier, the current slot remains unused and the corresponding level in the tree is skipped (see node n_1 in Figure 6-2). Following this strategy, the path form the root node of the tree to a given node represents a unique fault scenario, resulting in an exhaustive enumeration of all possible scenarios.

As detailed in [HRB12], for each of the nodes the task that is actually executed in a given slot is computed based on the execution results of the previous slots. Assuming the availability of perfect fault detectors, a task is executed successfully if at least one of its (replicated) instances executed without faults. Based on these results, each node in the tree is assigned to one of the following states:

- Faulty, iff based on the path from the root of the tree to the particular node, there exists no possibility to execute the job successfully in the remaining slots.
- Successful, iff the entire job is already finished using the successful slots specified in $\boldsymbol{x}$, i.e., the remaining slots are not needed.
- Unknown, otherwise.

A recursive implementation of the analysis terminates as soon as either no more nodes with status unknown exist, or when a tree has been expanded to a maximum depth equal to the number of scheduling slots (corresponding to the size $|\boldsymbol{x}|$ of a fault scenario).

Based on the working set, the probability of a successful node can be computed as follows:

$$\Pr(\boldsymbol{x}) = \prod_{x_i \in \mathbf{x} \wedge x_i = 1} \Pr(s_i) \cdot \prod_{x_i \in \mathbf{x} \wedge x_i = 0} (1 - \Pr(s_i)),$$

where the success probability $\Pr(s_i)$ of the individual tasks can be computed based on the underlying fault model.

Since the worst case complexity of the tree-based analysis sketched above is exponential [HBR11], a safe approximation is used in order to improve scalability. As it can be observed from the above formula, fault scenarios with a higher number of slots have a lower occurrence probability. Hence, bounding the number of faulty slots in the path from the root of the tree to the current node is an approximation of the system reliability. Since this approach may only result in a number of successful nodes with a high number of faulty slots to be discarded during the analysis, the procedure constitutes a safe under-approximation of the system reliability [HBR11].

As pointed out above, the analysis summarized so far assumes that all transient faults are detected at the end of the execution of the affected task. In order to relax this unrealistic, or – if at all possible – very expensive assumption [LCP09; SSS10], an extension of the analysis that considers the use of imperfect fault detectors will be discussed in the following [HHR12]. In this case, the execution of a task can result into the following outcomes:

- Successful execution of the task.
- The task is affected by transient fault which is detected (allowing for the implementation of fail-silent behavior).
- A transient fault occurs which is not detected (possibility resulting in silent data corruption in the system).

If voters are used to mask errors, the following effects have to be considered on the outcome of the voted result in case multiple faults occur:

- The task executes successfully, or all faults are masked by the voter.
- The voter is able to detect, but not able to mask the fault.
- Multiple faults occur and the incorrect outputs mask the correct one.

Clearly, the two latter cases are not desirable since they reduce the reliability of the system. In order to consider this relaxed assumption, the tree-based reliability analysis needs to be generalized as follows:

- The possible outcomes of a slot in the definition of the fault scenario need to be extended to three states, representing the cases discussed above.
- Since the voter requires the availability of the data of its producer tasks, only strict schedules need to be considered [HHR12].

6.3.2 Recovery Strategy for the Fault Model: Core is Halted

In this section, we review some papers and projects that have dealt with the fault model *a core is halted*.

6.3.2.1 Classical Strategies

In the space domain, critical components can be replicated into multiple copies (so-called hardware redundancy). Hardware replication such as triple-modular redundancy (TMR) configurations enables to tolerate both transient and permanent faults. While hardware replication results in less timing overhead since the replicas can typically run in parallel, it results in increased cost such as the provision of additional hardware resources [XLK04].

In contrast to that, software replication is typically more cost-efficient (because of the possibility to share slack time for the temporal replication of different software components). However, there are a number of restrictions of software replication that need to be considered:

- Because of the lack of redundant hardware resources, software fault-tolerance is only suitable to tolerate transient faults.
- The approach comes with an overhead in time [IPE05].
- For real-time applications, the schedulability of the tasks (including replicas) must be considered.

The configuration of fault-tolerance mechanisms at the application-level, including the selection and placement of redundancy, is a critical design decision that requires the joint consideration of timing and reliability properties which need to be balanced with the corresponding overhead. The problem is complicated by the fact that the amount of redundancy influences the schedulability of the application [IPE05; KHM03]. In the following, different formulations of the reliability analysis that assume different underlying scheduling policies in the system will be summarized. The system's scheduling policy has an impact onto the formulation and the complexity of the reliability analysis [BCJ12].

If there are no dependencies between tasks are considered (general schedules), or the assumption is made that all replicas of a predecessor task have finished before the current task, the execution status (i.e., whether a fault has occurred) of predecessor tasks has no influence onto the start time of successor tasks. In these cases, tasks may be considered independently in the reliability analysis, resulting in the following closed formulation [GK09]:

$$\Pr(S,J) = \prod_{t \in J} (1 - (1 - \Pr(t))^{num(t)}$$

Here, $\Pr(S,J)$ is the reliability of job J achieved by schedule S, $\Pr(t)$ is the reliability of task t and $num(t)$ is the number of replicas of a task t.

The analyses in [BCJ12; GK09] exclusively consider hardware redundancy, i.e., the replicas of tasks are mapped to different processing elements, and hence do not consider shared slots.

6.3.2.2 Virtual Mapping Technique

The analysis summarized in Section 6.3.1 considers only transient faults. In the following, an extension that allows the consideration of permanent faults of processing elements will be summarized. For permanent faults, either each task must be spatially replicated from the beginning, or it must be guaranteed that it can be migrated to a slack slot on a different processing element. An additional constraint which enforces that each task is spatially replicated is a simple approach that comes with high resource and hardware overhead, however.

A more cost-efficient alternative to handle permanent faults is task migration. To design such a system, one of the most important goals is to minimize the overhead of migration which can be achieved using pre-computed task-mapping tables. In the following, the application of this strategy to the techniques described above will be sketched. Here, an important observation is that in the case of time-triggered scheduling, the migration cost is highly influenced by data dependencies.

The *virtual mapping technique* [HBR11] extends the strategy for transient faults from Section 6.3.1 to also consider permanent faults. The following formulation of the analysis assumes that it is embedded into the exploration process introduced in Section 11.1.2. Hence, it refers to the representation of the task mapping in terms of a chromosome, and requires a scheduler for the (re-) construction of the corresponding schedule. The idea of virtual mapping is to trace potential places for task migrations already at the time when the schedule is constructed from the chromosome. A virtual mapping of task t to a processing element p is represented in this encoding scheme using a negative integer −p, which encodes that p is the migration target of task t. When constructing the schedule, for each virtual mapping also a migration slot for the respective task will be instantiated. During normal execution, virtual slots are used as slack slots for other tasks mapped onto the same processor in order to reclaim the time reserved for task migration and to improve the toleration of transient faults. Also, virtual mapping slots may be combined with other slack slots scheduled on the same processor to reduce the length of the schedule if the normal slack slot is at least as large as the virtual mapping slot and all data dependencies are obeyed.

The above technique uses task migration only as an emergency response in case a permanent fault occurs, i.e. transient faults occurring after the migration are not considered. However, the approach has the advantage that it can be efficiently integrated into the optimization process in Section 11.1 and does not require an additional objective function. It can be encoded using constraints onto the chromosome that state for which processing elements permanent faults need to be considered. Additionally, the implementation of task migration is cheap since it does not require to change the scheduling slots, but only to replace the corresponding tasks (e.g., by deploying the job images to the migration targets upfront, and only updating the priority table).

Hardware fault detection with SWAT: [HLRCA09]

Platform: multi-core
Fault model: Only faults in the core are considered, and a single core fault model is assumed (at most

one core is faulty).
Objective: lightweight fault detection (available recovery mechanisms) and diagnosis (trace replay based algorithm).

The paper presents an application of "symptom based detection and diagnosis" for faults in multicore architectures running multi-threaded software, developed during the project: SWAT (SoftWare Anomaly Treatment):

- it shows that using symptom-based detectors result in a very low Silent Data Corruption (SDC) rate for both permanent and transient hardware faults.
- given the new challenges posed by multicores, it proposes a permanent fault diagnosis algorithm for multithreaded applications running on multicore systems, using deterministic replay to diagnose the faulty core.

Four symptom detectors allow detection of different kinds of hardware faults:

- anomalous software behaviour (through fatal traps)
- hangs (through monitoring of branch instruction frequency)
- abnormal high OS instructions use (through performance counters)
- panic (through an unrecoverable error detection and centralized panic reporting routine).

The multicore diagnosis procedure is based on a TBFD (Trace Based Fault Diagnosis) that leverages the above recovery mechanisms

- to distinguish between software bugs and permanent or transient hardware faults.
- to isolate the faulty microarchitectural component in case of permanent faults.

6.3.2.3 Reconfigurable avionic platforms

Scarlett Project: [BNP09], [BBN+10], [BBG+12]

Platform: distributed system composed of IMA module (uni-processor) connected via AFDX network
Fault model: failure of module. Hypothesis: fail silent module (a unique failure mode: loss)
Objective: Reconfigurable IMA should be able to change the configuration of the platform by moving applications hosted on a faulty computing module to spare computing modules. The main objective of such an extension is to reduce the cost of unscheduled maintenance and to improve the operational reliability of the aircraft while preserving current safety levels.

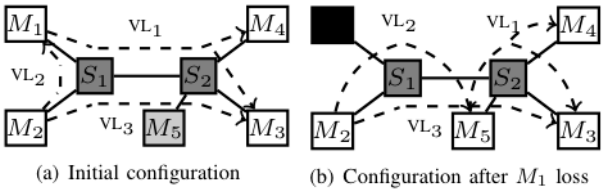

Figure 6-3 Example of a Scarlett reconfiguration

Let us illustrate the notion of reconfigurable IMA through a simple example: the platform is composed of five modules (M_1, …, M_5) and two communication switches (S_1,S_2). The initial configuration is drawn in **Figure 6-3** (a): module M_5 is a spare initially shut down and free of application (grey in the picture). If some failure occurs on module M_1, the applications initially hosted on this module can be reconfigured on the spare M_5 and all communications from and to M_1 (VL_1, VL_2, VL_3) must also be rerouted according to this new allocation. The reached configuration is shown in **Figure 6-3** (b).
Fault detection:

When a failure occurs, it is first detected by the CMS (Centralized Maintenance System) and the failure must be confirmed by a Module test via the CM (Cabinet manager). CMS already exists in aircraft while the CM is a new entity.
System recovery: reconfiguration

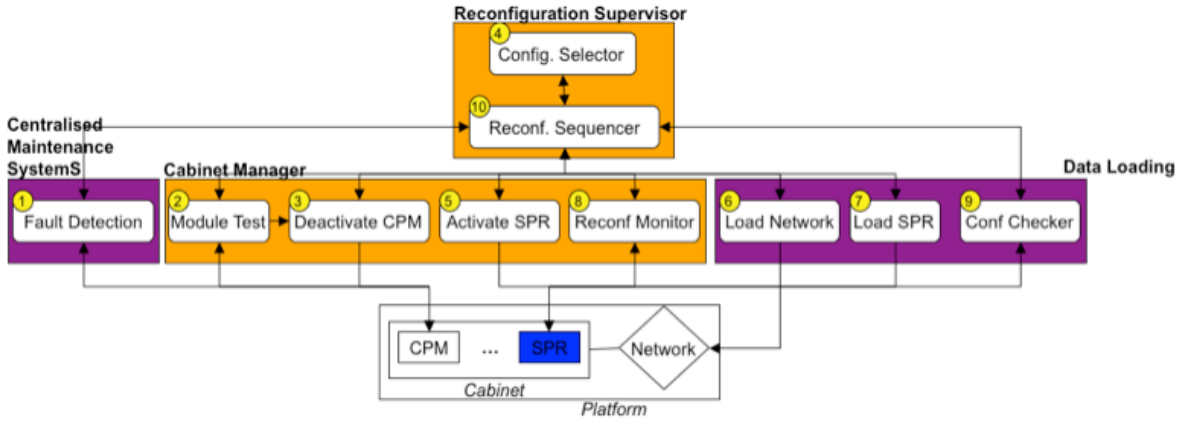

Figure 6-4 IMA reconfiguration architecture

The RS (reconfiguration supervisor) is the entity in charge of handling any reconfiguration. If a fault is detected and confirmed then the CM switches off the faulty module and the RS tries to perform a reconfiguration. The Configuration selector stores a graph of validated platform configurations. Given the current configuration and the failed module, the function identifies an available spare which can host the software of the failed module. If a new configuration exists, the reconfiguration is done in three steps: (1) The activated Spare module (SPR) function turns on the selected spare. (2) The load network function updates the routing table of the switch for adding the spare. (3) The load SPR function downloads the software on the spare. There are also several verification activities. The reconfiguration monitor function monitors that the network and the spare download the correct configuration. The configuration checker verifies at the end of the downloading that the spare has loaded the expected software. This is the last check and if every went fine, the reconfiguration sequencer returns to a waiting state until the next reconfiguration.
The reconfigurable IMA has been implemented on a real demonstrator.

DIANA Project: [SSE10]

Platform: distributed system composed of IMA modules supporting ARINC653 (APEX), connected via an avionic network
Fault model: failure of module. Hypothesis: fail silent module (a unique failure mode: loss)
Objective: reduction of software development costs, reduction of the amount of hardware resources needed to achieve the required level of dispatchability.

An alternative architecture for Reconfigurable IMA platform was proposed in the DIANA project. This architecture shares the goals and assumptions of the Scarlett architecture. The goal is also to improve aircraft operational reliability and it is assumed that the set of authorized configurations is computed off-line. The project introduces the concept of multi-static reconfiguration, or set of pre-qualified configurations from which the active one will be autonomously selected according to the system health state at system start-up. The DIANA architecture is different because Fault detection, Fault diagnosis, Reconfiguration Supervision and Data Loading are implemented in a distributed way whereas they are implemented in a centralized way in the Scarlett architecture. In the DIANA architecture, each module hosts a component that is able to test the health status of its module and to exchange it with other modules until a consensus is reached on the identity of faulty modules. Then, based on the result of the consensus protocol, the non-faulty modules select the new

configuration and they load the new applications. The authors claim that the distributed implementation improves the availability and integrity of reconfiguration mechanisms. A configuration selection mechanism, exploiting a Byzantine Agreement algorithm, is discussed. The way the adopted algorithm is shown correct is detailed. Finally, the implementation of the mechanism on top of an ARINC 653 Application is briefly described.

Reconfigurable Military Avionic Systems: [See96], [Ell97], [ASAAC04a], [ASAAC04b]

Platform: INMOS T800 Transputer based system [Ell97]. F-22 hardware architecture, including several digital signal and data processing modules and global bulk memory interconnected by High Speed Data and Test & Maintenance Busses [See96].
Fault model: failure of core
Objective: maintain expected functionality after a fault.

The reconfigurable avionic systems use reconfiguration as a mean to achieve fault tolerance. In [See96] "*Reconfiguration is one fault tolerance mechanisation for providing expected functionality after a fault*". The methodology, the software and/or hardware architecture design followed by these works are:

- error diagnostic → selection of new configuration → application of configuration
- pre-computed and certified configurations
- timing constraints for applying any reconfiguration

Platform: distributed system composed of IMA modules connected by a network
Fault model: failure of a component or flexibility for mission critical systems.
Objective: Improvement of mission & operational performance. For certification purpose, the reconfiguration process must be provably predictable.

The Allied Standard Avionics Architecture Council (ASAAC) established standards to be applied to the embedded avionics of future military aircraft. Among others, [ASAAC04a], [ASAAC04b] standards define reconfiguration principles for military IMA aircraft. Again, the idea is to pre-compute the configurations. The documents enumerate guidelines for dealing with faults and reconfiguration in the IMA context, targeting operational reliability.
The main recommendation is: "*the design process shall take into account all possible system events when defining the configuration/reconfiguration process, such that all eventualities can be taken into account an analysed during the design of the system. It shall never happen that a system is in a state where unexpected events occur*".

A proposed reconfiguration concept includes intermediate static states, in order to provide predictability through the process of reconfiguration. In order to maintain consistency throughout system state transitions, each transition consists of a series of smaller mechanisms. Each of these mechanisms is considered as a single atomic action that cannot be interrupted and is closely bounded in terms of function and needed time. Atomic actions and state transitions must be in place for all the identified faults.

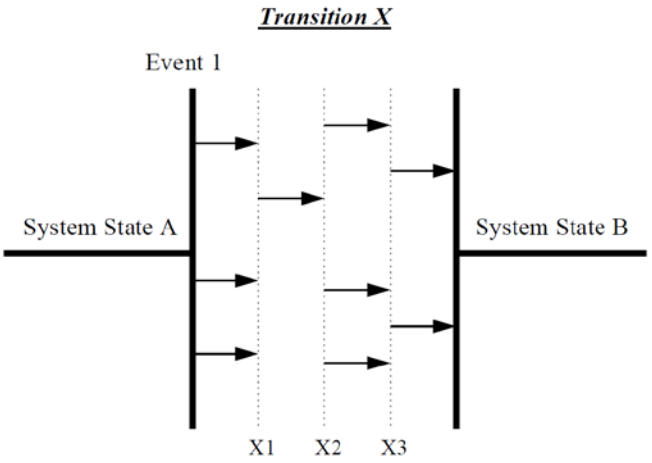

Figure 6-5 A reconfiguration action decomposed into atomic actions

Another interesting aspect, especially in the context of multi-core processors, concerns the type or reconfiguration progression. Two alternatives can be defined:

- Reconfiguration is terminated when all atomic actions have completed sequentially and when a given (predefined) period of time has passed since the reconfiguration was triggered.
- Reconfiguration allows an independent parallel progression of different atomic actions, executed as soon as possible.

The first alternative is considered as more predictable, hence being a better solution for systems requiring higher confidence in behavior, while the second is more flexible and adapted for mission critical systems. The system designer will make his choice according to the certification requirements.

Reconfigurable Space Systems:

The classical FDIR (Failure Detection Isolation and Recovery) procedure embedded in most space systems uses also dynamic reconfiguration during the recovery phase.

6.3.2.4 Fault-Tolerant Scheduling: [GLS01a], [GLS01b]

Platform: distributed system composed of processors connected by a network, modeled by a graph. Ability to produce automatically (SynDEx) fault-tolerant distributed code for various targets (DSP – Digital Signal Processor, Transputer,…).

Fault model: fail-silent processor failures, the number of which is known.

Objective: produce automatically a fault-tolerant distributed schedule for a data-flow algorithm (Lustre, Esterel, Signal…) onto the given architecture.

A distribution heuristics replicates the computations and data-dependencies of the considered algorithm. The obtained static schedule achieves fault-tolerance with software redundancy for computations and time redundancy for data-dependencies. By taking into account the execution durations of all computations on all processors and the communication durations of all data dependencies on all communication links, the total execution time is calculated for the obtained schedule, both in presence or in absence of faults.

6.3.3 Recovery Strategy for the Fault Model: Temporal Overload Situation

In this section, we review some papers and projects that dealt with the fault model *a temporal overload situation occurred.*

6.3.3.1 Monitoring of real-time behaviors: [BLS06], [RRF10], [BFR13]

Platform: sequential code running on a core
Fault model: non respect of timing behaviour, e.g. overrun of deadline
Objective: detection of the fault with a run-time monitor. Recovery is ad hoc.

The objective of run-time monitoring is to check on-line, during the real execution, the timing behaviors of the system and verify if they are compliant with an abstract view of the expected behavior. If the system diverges from the specification, then a recovery may be applied.
In [BLS06] and [RRF10], the timing specifications are expressed with Timed Linear Temporal Logic (TLTL) formulas. Practically, timed automata are used to implement the valid behaviors and the decision layer stores the automaton description including the location invariants and the transition table. The verification function works for each event as follows: either it is valid and the execution continues or the event is invalid, in which case a recovery procedure or an error is called. Such an implementation requires a strong synchronization between the application and the monitor, and in particular it is necessary to ensure mutual exclusion.
An example of a COM/MON architecture extended with timing monitoring is given in Figure 6-6. Two systems, the command (COM) and the monitoring (MON), are in active redundancy. They compare their outputs and if they disagree, the COM/MON is deactivated. This is a typical design pattern for a *fail silent* architecture. The monitoring is integrated both in the COM and in the MON parts (denoted as watchdog).

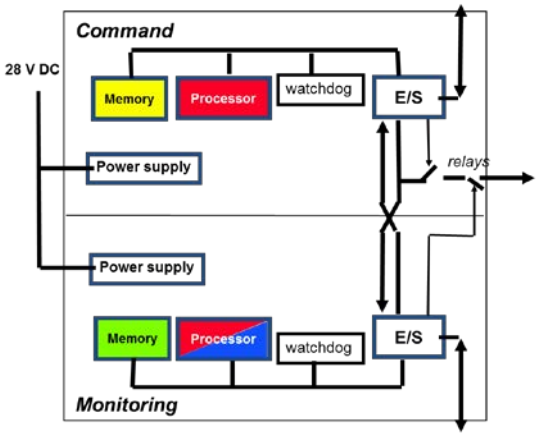

Figure 6-6 COM/MON architecture with timing monitoring

In [BFR13], the monitoring strategy has been extended for multi-threaded code. The monitor is decomposed into pieces that apply local detection while exchanging messages to ensure a coherent checking.

6.3.3.2 Multi-mode applications: [NES12]

Platform: partitioned multi-processor
Fault model: non respect of timing behavior
Objective: change the mode without violating timing constraint. Pre-computation of behaviors.

The paper focuses on the automotive domain where tasks can exhibit a dynamic real-time behavior, e.g. the period depends on the engine speed. This variability leads to a continuous change in the configurations taken into account by the OS schedule on the processors. We then speak of multi-mode applications that can switch between different operational modes at run-time. Such a change of rate may make the system unschedulable and the engine control inefficient or even unstable.
The possible recoveries are the following: degraded functional execution (with restricted WCET), abort or suspend low criticality tasks.

The approach consists in analyzing all potential run-time scenarios and study in details the critical ones. The mode changes can follow two strategies:

- Synchronous protocols do not allow switching in a new mode tasks until all tasks to be dropped have completed their last activation. Advantages: segregation between modes, no specific schedulability analysis during the transition phase. Drawback: delayed beginning of the new mode.
- Asynchronous protocols allow functions of the new mode to be started simultaneously to the old mode. The execution of functions of both modes generates an increased workload during the transition phase that can lead to timing violations.

6.3.3.3 Run-time Control to Avoid Timing Violation

Several approaches propose resources reallocation based on information derived from monitoring their utilization, e.g. the memory accesses. For instance, in [NPBTWS13] interference-sensitive WCETs are computed based on a preliminary analysis of the resource usage of tasks. The shared resources are off-line partitioned among tasks. A run-time monitoring device observes the resource usage of each task and suspends the task that overtakes the allocated capacity. In [NP13] the approach is extended by allowing safe dynamic changes in the resource partitioning, when resources are underutilized. In [YYPC13] an approach has been developed to reserve memory accesses for critical tasks. A run-time controller has been implemented which regulates the accesses to the shared memory and ensures temporal isolation among tasks. An off-line profiling technique has been proposed in [MDBCCP13] which finds the most frequently accessed memory pages in a task. Then, this information is used to modify the variables' position in the shared caches in order to reduce the interferences. Another hardware approach is described in [DIS14] where the monitoring is only performed when enough slack time exists which guarantees that the monitoring does not impact the meeting of the real-time constraints of the tasks. If the slack is insufficient, a dropping operation is executed to minimize the monitoring overhead.

6.3.3.4 Run-time Control to Increase Task Parallelism in Mixed-Critical Systems: [KBPRR14]

Platform: partitioned multi-processor
Fault model: non respect of timing behavior
Objective: detect at run-time a possible deadline overrun and suspend low criticality tasks.

When integrating mixed critical systems on a multi/many-core, one challenge is to ensure predictability for high criticality tasks and a good utilization for low criticality tasks. Since the considered platforms consist of multi/many-core COTS systems, the WCET estimations upper bound the effects of the possible interferences on the system shared resources by assuming full congestion. Therefore, a dramatic difference occurs in WCETs 1) when the application is executed alone on one core and 2) when other applications are concurrently executed on the remaining cores. This difference may lead to the system unschedulability in case:

$$WCET_{iso} < D_C < WCET_{max}$$

where $WCET_{iso}$ is the WCET of a high criticality application τ_C in isolated execution, D_C is its deadline and $WCET_{max}$ is its WCET in maximum load, i.e. when several applications are concurrently executed on the system.

The authors work on that issue for a task set consisting of a unique critical task and several low criticality tasks by proposing a run-time WCET controller that: (1) regularly checks if the interferences due to the low criticality tasks can be tolerated; (2) suspend them when a possible future overrun is detected; (3) resume them at the end of the critical task. To do so, the critical task is modeled by a set of *Extended Control Flow Graphs* (ECFGs), which is a control flow graph with observation points. A graph grammar formally describes the set of ECFGs and based on the obtained ECFGs, a safe WCET analysis is applied for the pre-computation of several partial remaining WCETs used by the run-time control. Indeed, at each observation point the run-time controller checks the

safety condition:

$$RWCET_{iso}(x) + RWCET_{max,PTP} + t_{RT} \leq D_C - ET(x)$$

where $RWCET_{iso}(x)$ is the remaining WCET of τ_C at an observation point x in the isolation scenario, $RWCET_{max,PTP}$ is the WCET until the next observation point, t_{RT} is the total execution time of the run-time control mechanism and ET(x) is the real execution time of τ_C until point x. If this safety condition does not hold, the low criticality tasks are suspended.

The correctness of the run-time computation algorithm is proved that ensures the respect of the deadline for the critical task. The gains are assessed through a series of simulations.

6.4 Challenges/Shortcomings w.r.t. MCS

In the DREAMS project, we will consider the following failures:

a) A core is halted
b) Temporal overload situations due to low criticality tasks accessing shared resources

For those fault model, we will have to:

- Identify adequate fault detection mechanisms for the DREAMS platform
- Identify adequate recovery strategies for the DREAMS platform based on
  a) adaptation with mode changes, reconfiguration
  b) existing approaches. Since they currently do not mix permanent core failures and temporal overload situations, we will have to council them.
  c) management of the network for distributed multi-core platform.
  d) Pre-computation of schedules and transition phases

Regarding the implementation, we will have to deal with several issues:

a) cohabitation between fault-tolerance mechanisms and XtratuM hypervisor
b) real multi-core COTS implementation (Freescale T4240)
c) applicability to the demonstrators and in particular to the avionic application FMS of WP6.

6.5 References

[ASE11] Axer, P., Sebastian, M., and Ernst, R. (2011). Reliability analysis for MPSoCs with mixed-critical, hard real-time constraints. In *Proceedings of the Seventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '11)*, pages 149–158, New York, NY, USA. ACM.

[BCJ12] Benoit, A., Canon, L.-C., Jeannot, E., and Robert, Y. (2012). Reliability of task graph schedules with transient and fail-stop failures: complexity and algorithms. *J. Sched.*, 15(5):615–627.

[DGB95] Degraeve, R., Groeseneken, G., Bellens, R., Depas, M., and Maes, H. E. (1995). A consistent model for the thickness dependence of intrinsic breakdown in ultra-thin oxides. In *Proceedings of the International Electron Devices Meeting (IEDM '95)*, pages 863–866.

[Fo97] Fohler, G. (1997). Adaptive fault-tolerance with statically scheduled real-time systems. In *Proceedings of the 9th Euromicro Workshop on Real-Time Systems*, pages 161–167.

[GCJ01] Gall, M., Capasso, C., Jawarani, D., Hernandez, R., Kawasaki, H., and Ho, P. S. (2001). Statistical analysis of early failures in electromigration. *J. Appl. Phys.*, 90(2):732–740.

[GK09] Girault, A. and Kalla, H. (2009). A novel bicriteria scheduling heuristics providing a guaranteed global system failure rate. *IEEE Trans. Dependable Secure Comput.*, 6(4):241–254.

[HBR11] Huang, J., Blech, J. O., Raabe, A., Buckl, C., and Knoll, A. (2011). Analysis and optimization of fault-tolerant task scheduling on multiprocessor embedded systems. In *Proceedings of the Seventh*

IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '11), pages 247–256, New York, NY, USA. ACM.

[HHR12] Huang, J., Huang, K., Raabe, A., Buckl, C., and Knoll, A. (2012a). Towards fault-tolerant embedded systems with imperfect fault detection. In *Proceedings of the 49th Annual Design Automation Conference (DAC '12)*, pages 188–196, San Francisco, CA, USA.

[HRB12] Huang, J., Raabe, A., Huang, K., Buckl, C., and Knoll, A. (2012b). A framework for reliability-aware design exploration on MPSoC based systems. *Des. Autom. Embed. Syst.*, 16(4):189–220.

[IEC01] International Electrotechnical Commission (2001). ISO/IEC 61508, functional safety of electrical/electronic/ programmable electronic safety-related systems, parts 1 - 7.

[IPE05] Izosimov, V., Pop, P., Eles, P., and Peng, Z. (2005). Design optimization of time- and cost-constrained fault-tolerant distributed embedded systems. In *Proceedings of the Design, Automation and Test in Europe Conference (DATE '05)*, pages 864–869 Vol. 2.

[KHM03] Kandasamy, N., Hayes, J. P., and Murray, B. T. (2003). Transparent recovery from intermittent faults in time-triggered distributed systems. *IEEE Trans. Comput.*, 52(2):113–125.

[LCP09] Lyle, G., Chen, S., Pattabiraman, K., Kalbarczyk, Z., and Iyer, R. (2009). An end-to-end approach for the automatic derivation of application-aware error detectors. In *Proceedings of the IEEE/IFIP International Conference on Dependable Systems Networks (DSN '09)*, pages 584–589.

[MSM00] Mitra, S., Saxena, N. R., and McCluskey, E. J. (2000). Common-mode failures in redundant VLSI systems: a survey. *IEEE Trans. Rel.*, 49(3):285–295.

[MWE03] Mukherjee, S. S., Weaver, S., Emer, J., Reinhardt, S. K., and Austin, T. (2003). A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '03)*, pages 29–40.

[SSS10] Schiffel, U., Schmitt, A., Süßkraut, M., and Fetzer, C. (2010). Software-implemented hardware error detection: Costs and gains. In *Proceedings of the 3rd International Conference on Dependability (DEPEND '10)*, pages 51–57.

[SW89] Shatz, S. M. and Wang, J.-P. (1989). Models and algorithms for reliability-oriented task-allocation in redundant distributed-computer systems. *IEEE Trans. Rel.*, 38(1):16–27.

[XCD10] Xiang, Y., Chantem, T., Dick, R. P., Hu, X. S., and Shang, L. (2010). System-level reliability modeling for MPSoCs. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis (CODES+ISSS '10)*, pages 297–306, New York, NY, USA. ACM.

[XLK04] Xie, Y., Li, L., Kandemir, M., Vijaykrishnan, N., and Irwin, M. (2004). Reliability-aware co-synthesis for embedded systems. In *Proceedings of the 15th IEEE International Conference on Application-Specific Systems, Architectures and Processors*, pages 41–50.

[ZAZ09] Zhao, B., Aydin, H., and Zhu, D. (2009). Enhanced reliability-aware power management through shared recovery technique. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD '09)*, pages 63–70.

[ZA06] Zhu, D. and Aydin, H. (2006). Energy management for real-time embedded systems with reliability requirements. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD '06)*, pages 528–534.

[ZA09] Zhu, D. and Aydin, H. (2009). Reliability-aware energy management for periodic real-time tasks. *IEEE Trans. Comput.*, 58(10):1382–1397.

[ALR04] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, Carl E. Landwehr: Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Transactions on Dependable and Secure Computing, Volume 1. 11-33. 2004

[AFA07] Luis Almeida, Sebastian Fischmeister, Madhukar Anand, and Insup Lee. A dynamic scheduling approach to designing flexible safety-critical systems. In EMSOFT '07: Proceedings of the 7th ACM & IEEE international conference on Embedded software, pages 67–74, New York, NY, USA, 2007. ACM

[BFR13] Olivier Baldellon, Jean-Charles Fabre, Matthieu Roy: Minotor. Monitoring Timing and Behavioral Properties for Dependable Distributed Systems. In 19th Pacific Rim International Symposium on Dependable Computing (PRDC'13) 2013: 206-215

[BLS06] Andreas Bauer, Martin Leucker, Christian Schallhart: Monitoring of Real-Time Properties. In 26th International Conference Foundations of Software Technology and Theoretical Computer Science (FSTTCS'06): 260-272

[BNP09] Pierre Bieber, Eric Noulard, Claire Pagetti, Thierry Planche et Francois Vialard. "Preliminary design of future reconfigurable IMA platforms". In : Proceedings of the workshop APRES - SIGBED Review. 2009, p. 7.

[Bor05] Shekhar Borkar. Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation. IEEE Micro, 25(6), 2005.

[But97] Giorgo Butazzo. Hard real-time computing systems. Predictable Scheduling Algorithms and Applications. Chap 8 Handling overload conditions. 1997.

[DIS14] T. C. Daniel Lo, Mohamed Ismail and G. E. Suh, "Slack-aware opportunistic monitoring for real-time systems," in 20th Real-Time and Embedded Technology and Applications Symposium (RTAS), 2014.

[DGL01] Catalin Dima, Alain Girault, Christophe Lavarenne, Yves Sorel: Off-Line Real-Time Fault-Tolerant Scheduling. PDP 2001: 410-417

[GLS01a] Alain Girault, Christophe Lavarenne, Yves Sorel, Mihaela Sighireanu: Fault-Tolerant Static Scheduling for Real-Time Distributed Embedded Systems. ICDCS 2001: 695-698

[GLS01b] Alain Girault, Christophe Lavarenne, Mihaela Sighireanu, Yves Sorel: Generation of Fault-Tolerant Static Scheduling for Real-Time Distributed Embedded Systems with Multi-Point Links. IPDPS 2001: 125

[HLRCA09] Siva Kumar Sastry Hari, Man-Lap Li, Pradeep Ramachandran, Byn Choi, Sarita V. Adve. mSWAT: Low-Cost Hardware Fault Detection and Diagnosis for Multicore Systems. In the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). 2009. p 122-132

[Hyde10] Randy Hyde. The Art of Assembly Language Programming. 2010

[KBPRR14] Angeliki Kritikakou, Olivier Baldellon, Claire Pagetti, Matthieu Roy, Christine Rochange. Run-time Control to Increase Task Parallelism in Mixed-Critical Systems. In 26th Euromicro Conference on Real-Time Systems (ECRTS14). To appear.

[MDBCCP13] R. Mancuso, R. Dudko, E. Betti, M. Cesati, M. Caccamo, and R. Pellizzoni, "Real-time cache management framework for multi-core architectures," in 19th Real-Time and Embedded Technology and Applications Symposium (RTAS'13), pp. 45–54, 2013.

[NES12] Mircea Negrean, Rolf Ernst, and Simon Schliecker, "Mastering Timing Challenges for the Design of Multi-Mode Applications on Multi-Core Real-Time Embedded Systems" in 6th International Congress on Embedded Real-Time Software and Systems (ERTS), (Toulouse, France), February 2012.

[NP13] J. Nowotsch and M. Paulitsch, "Quality of service capabilities for hard real-time applications on multi-core processors," in Proceedings of the 21st International conference on Real-Time Networks and Systems (RTNS'13), pp. 151–160, 2013.

[NPBTWS13] J. Nowotsch, M. Paulitsch, D. Bühler, H. Theiling, S. Wegener, and M. Schmidt, "Multi-core interference-sensitive wcet analysis leveraging runtime resource capacity enforcement," Tech. Rep. 2013-10, University of Augsburg, Germany, 2013.

[RRF10] Thomas Robert, Matthieu Roy, Jean-Charles Fabre. Early Error Detection for Fault Tolerance Strategies. In 18th International Conference on Real-Time and Network Systems (RTNS'10), 2010.

[SSE10] Tobias Schoofs, Peter Schmitt, Christian Engel, Eric Jenn, and Rodrigo Coutinho.

"enhanced dispatchability of aircrafts using multi-static configurations". In European Real-Time Systems and Software (ERTSS), 2010.

[SRP13] Rishad A Shafik, Gerard Rauwerda, Jordy Potman, Kim Sunesen, Dhiraj Pradhan, Jimson Mathew, and Ioannis Sourdis. Software modification aided transient error tolerance for embedded systems. 16th Euromicro Conference on Digital System Design, 2013.

[YYPC13] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha, “Memguard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms,” in 19th Real-Time and Embedded Technology and Applications Symposium (RTAS), pp. 55–64, 2013.

[WKS08] Philip M. Wells, Koushik Chakraborty and Gurindar S. Sohi. Adapting to Intermittent Faults in Multicore Systems. In 13th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-XIII). 2008. 255-264

7 Scheduling Problem Formalization

The problem of partitioning and scheduling tasks is equivalent to a multi-dimensional bin-packing problem which is NP-hard [CGJ96]. The DREAMS platform is composed of several multi/many-cores connected via a TTEthernet network. We will formalize the scheduling problem for the demonstrators and in particular the WP6 avionic demonstrator. Therefore, the hypotheses are the following: each multi-core hosts the Xtratum hypervisor and can encounter some failures (a core is halted or a temporal overload situation). The objective of the scheduling problem is to pre-compute off-line the schedules in the nominal mode, in the degraded modes and during the transition phases (from one configuration to another).

The mapping problem has two inputs:

1. Application model.
    a. The high criticality tasks are communicating dependent tasks (periodic, restartable or sporadic). A task can have several modes. These assumptions are compliant with the critical application FMS (Flight Management System) of WP6.
    b. There is no restriction on the low criticality tasks.
2. Platform model.
    a. Architecture and execution model of the multi-core
    b. Each core has two levels of scheduling:
        i. application level (pre-defined time slots compliant with IMA paradigm)
        ii. intra-partition level (RM, off-line scheduling)
    c. Fault model
    d. Network protocol is TTEthernet

The output are:

1. A static mapping of the partitions as time slots
2. A static mapping of applications to partition
3. A static mapping of intra application as off-line schedule
4. A static mapping of the sections (code, data, stack) on the platform memories
5. A static mapping of the communication between tasks
6. A static mapping of the communication on the network
7. A static graph reconfigurations for combination of failures
8. Static transitions procedures for moving from one configuration to another

7.1 General problem formulation

The task mapping problem consists in assigning (allocating) each task to a processor and ordering (scheduling) the execution of these tasks such that all functional (precedence and deadline) and non functional (memory capacity, bandwidth network, processor capacity, etc.) constraints are met. There are several equivalent ways to express this problem such as integer linear programming [Bar04] or counter example of a model checking analysis [GC01], [FPY02], [BLR05], [GGDGY07], [BBCNP12]. An easy way to express the problem is to use a constraint programming approach [Eke04], and we will use this approach for the WP4 formalization.

A *constraint satisfaction* problem (csp) is a tuple (X, D, C) where:

1. $X = \{x_1, \ldots, x_n\}$ is a set of variables;
2. D is a function that associates to each variable x_i its domain $D(x_i)$, i.e. the set of possible values of x_i;

3. $C = \{C_1 , C_2 , ..., C_k\}$ is the set of constraints. Each constraint C_j is a relation between some variables of X and reduces the values that can be taken simultaneously by the variables. The simplest constraints are comparisons defined over expressions of variables such as linear combinations, i.e. x ≤ y + z. More complex expressions such as min or max are also available.

A solution to the problem is an assignment of a value in $D(x_i)$ to each variable x_i in X such that all constraints are satisfied. When there are several solutions, it is interesting to express some preferences among the solutions. For this, the idea is to define a numerical function from the set of solutions whose values depend on a quality criterion. The objective is then to maximise this function. In this case, we speak of a Constraint Satisfaction Optimisation Problem, csop for short.

Example. A simple example of csp (X, D, C) describing the distribution of 4 tasks on 2 processors:

1. *$X = \{x_1 , x_2 , x_3 , x_4\}$ represents the set of the 4 tasks;*
2. *$D(x_1) = D(x_2) = D(x_3) = D(x_4) = \{0, 1\}$ corresponds to the variables domain. If x_i is equal to 0 it means that x_i is on the first processor, otherwise it is on the second one;*
3. *$C = \{x_1 = x_2, x_2 = x_4\}$ describes the constraints that x_1 and x_2 are on the same processor and that x_2 must not be on the same processor as x_4.*

There are several solutions: (x1 , x2 , x3 , x4) ∈{(0, 0, 1, 1), (0, 0, 0, 1), (1, 1, 1, 0), (1, 1, 0, 0)}. When adding the criterion f (A) =x1 + x3, the csop has only one solution {(1, 1, 1, 0)}.

Several formulations of similar problems can be found in the literature. The variations stand in the application model and/or platform model. We will reuse the existing formalizations as the basis for the task.

7.1.1 Formalization taking into account only timing constraints

An early approach to off-line scheduling of real-time tasks on multiprocessors is the work by [Xu93]. The task model considers dependent periodic tasks sets with mutual exclusion relations between program segments. The authors of [PSA97] consider synchronous implicit-deadline dependent task sets on a distributed architecture taking into account the network delays. [AS99] extend their results to schedule messages among the allocated tasks. From a fixed partition of the tasks, obtained by the previous approach, they compute a feasible local schedule and a near optimal message priority assignment. The approach by [SW00] tackles the generation of schedules for time-triggered systems, and combines the scheduling of tasks and messages on a shared bus. [HCD08] handle the mapping of tasks to processors communicating via CAN bus. They use preemptive fixed-priority scheduling and do not support precedence constraints.

Let us denote by

- *proc: T -> N* is the function that allocates the task t_i on processor $proc(t_i)$
- *s: T -> N* is the function that gives the start time of t_i

Example of standard constraints

- The mutual exclusion of two tasks for a partitioned schedule is simply: $proc(t_i) \neq proc(t_j)$
- A precedence constraint between two jobs for a non preemptive schedule is simply: $s(t_i) + C_i \leq s(t_j)$. This constraint can easily be extended for periodic tasks by applying the constraint on all jobs of an adequate window (hyper-period if the tasks have the same offset). It can also be extended for preemptive schedules.
- The schedulability can be harder to express. For synchronous mono-periodic tasks with implicit deadline and a partitioned schedule, it is : for all k, $\Sigma C_i \times (proc(t_i)=k) \leq T_i$

7.1.2 Formalization taking into account the two levels scheduling – IMA and intra-partition

[ABHP12] [ABCH13], [PA14] propose a formalization of the two levels scheduling for IMA platform. The partitions are strictly periodic, this means that each partition restarts exactly one period later. Let us denote by *s: TxN -> N* the function that gives the start time each job of t_i, then $s(t_i,k) = s(t_i,0)+kT_i$. The partitions communicate through VL (virtual link) on the AFDX (Avionics Full DupleX switched ethernet) network. The output is: computation of pre-defined slots on the processors and a mapping of each application on the slots. Once the mapping has been produced, a second mapping of the VL on the network is computed. The coding is based on MILP, game theory, branch-and-bound and Steiner trees.

The authors of [CPS12] define a method for mapping dependent periodic task sets on a time triggered architecture (mixing Arinc 653 modules and time triggered networks e.g. Flexray). The system model is a finite set of partitions where a partition comprises both a software application of the system and the execution and communication resources allocated to it. An application can also be associated with several modes, depending on the operational phase, and a switching between those modes is possible. The authors apply an off-line scheduling algorithm whose output is a scheduling table defining the allocation of processor and bus time to the various computations and communications.

7.1.3 Formalization taking into account an execution model

The authors of [BHPAJ08] compute a mapping of dependent periodic task sets on a distributed platform. The execution on the CPU is split into execution slices, in which tasks are executed, and communication slices, in which tasks exchange messages via a shared bus. They extend the work for multi-core platform [BCN12] subjected to the execution model that distinguishes, on each core, times of functional computation and times for accesses to the memory. These two types of computation occur in different slices which are statically defined. The sliced execution model operates as follows:

1. two kinds of slices alternate indefinitely on each core:
   - execution slices: a core executes a functional code without any shared resource access. All the instructions and data are locally stored in the caches;
   - communication slices: the core first flushes from local cache(s) to the RAM the values computed during the previous execution slice, and then fetches into local cache(s) all the codes and data required for the next execution slice.
2. a static synchronous scheduling of the slices on each core is defined off line. It describes a repetitive pattern where communication slices are synchronous. The figure below gives an example of such a static synchronous scheduling.

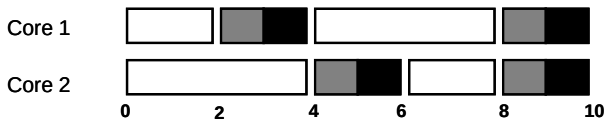

Figure 7-1 Example of static synchronous scheduling [BCN12]

The system model is tuple ⟨F,V,R⟩ with

1. $F = \{f_1,...,f_n\}$ finite set of tasks
   a) size_code: (resp. wcet:) $F \rightarrow N$ associates to each task fi the number of memory lines required to store the code (resp. its wcet, for instance computed by Otawa [BCRS10]);
   b) T : (resp. D :) $F \rightarrow N$ provides the period (resp. the relative deadline). $D(f_i) \leq T(f_i)$;
2. $V = \{v_1,...,v_k\}$ finite set of data
   a) size_var: $V \rightarrow N$ associates to each data its size;
   b) in: $F \rightarrow 2^V$ (resp. out: $F \rightarrow 2^V$) gives the set of data consumed (resp. produced) by each

task. A unique producer and several consumers;

c) $I \subseteq V$ (resp. $O \subseteq V$) the set of inputs (resp. outputs);

3. $R \subseteq F \times F$ relation of precedence relating tasks with the same period.

The mapping consists in computing the variables:

1. addrv: $V \rightarrow N$ associates an address to each data,
2. addrc: $F \rightarrow 2^N$ associates addresses to the task instructions
3. alloc: $F \cup I \cup O \rightarrow 2^{slices}$ indicates on which slices each task (or I/O) executes (is loaded/written)

7.2 Challenges/Shortcomings w.r.t. MCS

In the DREAMS project, we will unify the existing formulations to take into account the specifics of the platform:

a) Two levels scheduling due to Xtratum
b) Execution model rules imposed to be predictable
c) Faults recovery strategies: pre computed set of mapping/scheduling with safe transitions

7.3 References

[AS99] Abdelzaher TF, Shin KG (1999) Combined task and message scheduling in distributed real-time systems. IEEE Trans Parallel Distrib Syst 10(11):1179–1191. 1999

[APN11] M. Asberg, P. Pettersson, and T. Nolte. Modelling, verification and synthesis of two-tier hierarchical fixed-priority preemptive scheduling. In Real-Time Systems (ECRTS), 2011 23rd Euromicro Conference on, pages 172–181, 2011.

[BCRS10] C. Ballabriga, H. Cassé, C. Rochange, and P. Sainrat. Otawa: An open toolbox for adaptive wcet analysis. In 8th International Workshop on Software Technologies for Embedded and Ubiquitous Systems (SEUS'10), volume 6399 of LNCS, pages 35–46, 2010.

[BBCNP12] Julie Baro, Frédéric Boniol, Mikel Cordovilla, Eric Noulard, Claire Pagetti: Off-line (Optimal) multiprocessor scheduling of dependent periodic tasks. In Proceedings of the ACM Symposium on Applied Computing (SAC) 2012: 1815-1820

[Bar04] S. K. Baruah. Partitioning real-time tasks among heterogeneous multiprocessors. In ICPP '04: Proceedings of the 2004 International Conference on Parallel Processing, pages 467–474, Washington, DC, USA, 2004. IEEE Computer Society.

[BLR05] G. Behrmann, KG Larsen, JI Rasmussen. Optimal scheduling using priced timed automata. SIGMETRICS Perform Eval Rev 32:34–40, 2005.

[BHPAJ08] Frédéric Boniol, Pierre-Emmanuel Hladik, Claire Pagetti, Frédéric Aspro and Victor Jégu. A framework for distributing real-time functions. In: Formal Modeling and Analysis of Timed Systems (FORMATS'08), 2008, p 155–169

[BCN12] Boniol F, Cassé H, Noulard E, Pagetti C (2012) Deterministic execution model on cots hardware. In: 25th International Conference Architecture of Computing Systems (ARCS'12), Springer, Lecture Notes in Computer Science, vol 7179, pp 98–110. 2012

[CPS12] T. Carle, D. Potop-Butucaru, Y. Sorel, and D. Lesens, "From dataflow specification to multiprocessor partitioned time-triggered real-time implementation," INRIA, Research report RR-8109, Oct. 2012.

[CGJ96] Coffman Jr E, Garey M, Johnson D. Approximation algorithms for bin packing: A survey. In: Approximation algorithms for NP-hard problems, PWS Publishing Co., pp 46–93. 1996

[Eke04] Ekelin C. An optimization framework for scheduling of embedded real-time systems. PhD thesis, Chalmers University of Technology. 2004

[FPY02] Elena Fersman, Paul Pettersson, and Wang Yi. Automata with Asynchrounous Processes: Schedulability and Decidability. In Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, J.-P. Katoen, P. Stevens (Eds.). Grenoble, France, April 6-14, 2002. Lecture Notes in Computer Science 2280, pages: 67-82.

[GGJ96] M. R. Garey, R. L. Graham, D. S. Johnson, and A. C. Yao, "Resource constrained scheduling as generalized bin packing," Journal of Combinatorial Theory, vol. 21, pp. 257–298, 1976.

[GC01] E. Grolleau, A. Choquet-Geniet. Ordonnancement de tâches temps réel en environnement multiprocesseur à l'aide de réseaux de petri. In: Real-Time Systems, RTS'2001.

[GGDGY07] Nan Guan, Zonghua Gu, Qingxu Deng, Shuaihong Gao, Ge Yu: Exact Schedulability Analysis for Static-Priority Global Multiprocessor Scheduling Using Model-Checking. SEUS 2007: 263-272

[HCD08] Hladik PE, Cambazard H, Déplanche AM, Jussien N. Solving a real-time allocation problem with constraint programming. J Syst Softw 81(1):132–149. 2008

[LDY00] Lee, Y.-H. ; Daeyoung Kim ; Younis, M. ; Zhou, J. Scheduling tool and algorithm for integrated modular avionics systems. Proceedings of the 19th Digital Avionics Systems Conference (DASC), 2000.

[PSA97] Peng DT, Shin KG, Abdelzaher TF (1997) Assignment and scheduling communicating periodic tasks in distributed real-time systems. IEEE Trans Software Eng 23(12):745–758. 1997

[SW00] Schild K, Würtz J. Scheduling of time-triggered real-time systems. Constraints

5(4):335–357. 2000

[SP07] Frank Singhoff and Alain Plantec. AADL modeling and analysis of hierarchical schedulers. Proceeding SIGAda '07 Proceedings of the 2007 ACM international conference on SIGAda annual international conference Pages 41 – 50.

[Xu93] J. Xu. Multiprocessor scheduling of processes with release times, deadlines, precedence, and exclusion relations. Software Engineering, IEEE Transactions on 19(2):139 –154. 1993

8 Mapping Algorithms

The DREAMS architecture style allows distributing mixed criticality application on multicore distributed architecture. Allocating application tasks in such a context is a complex task in order to meet all the application requirements, some trade-offs are necessary. In the following we propose two task allocation algorithms: ESAMA focused on timing requirements and energy consumption minimization, and MIP-TECS13 focused on timing requirements and end to end delay minimization.

8.1 ESAMA

ESAMA was first proposed in [Zhang-2013], it is a genetic algorithm-based algorithm which performs a mapping of task on a distributed heterogeneous architecture. The criteria used to measure the performance of the algorithm are the energy consumption and the deadline misses.

ESAMA makes the following assumptions:

1. Non preemptive and periodic task model with precedence for the application,
2. Fixed communication delay (single bus)
3. Power consumption model

ESAMA outperforms a greedy local-search algorithm used as benchmarks but on small size application (30 to 50 tasks mapped on 3 or 4 nodes). ESAMA model does not exactly match the architecture style of DREAMS (larger system, partitioning, preemptive scheduling) but it is available out-of-the-box and can therefore be a source of inspiration for DREAMS.

8.2 MILP-TECS13

MILP-TECS13 was proposed in [Zhu-2013] which is an extension of MILP in [Zheng-2007]. These algorithms provide tasks mapping which minimizes the latencies while meeting the end-to-end constraints. The algorithms are based on Mixed Integer Linear Optimization, which is of exponential complexity. The system model is limited since it is specific to automotive systems.

In the context of DREAMS, a system model inspired from the one of [Zhu-2013] but extended to the DREAMS context and the use of a genetic algorithm as described in previous section seems a solution that could lead to reasonably good results in a reasonable computation time.

8.3 Challenges/Shortcomings w.r.t. MCS

ESAMA addresses MCS and is available out of the box. Its system model should be adapted to match the one of DREAMS. The challenge is also to move to large configurations with reasonable computation time.

8.4 References

[Zhang-2013] Xia Zhang, Jinyu Zhan, Wei Jiang, Yue Ma and Ke Jiang, “Design Optimization of Security-Sensitive Mixed-Criticality Real-Time Embedded Systems”, 1st workshop on Real-Time Mixed Criticality Systems (ReTiMiCS2013), Taipei, Taiwan, August 21, 2013

[Zhu-2013] Qi Zhu, Haibo Zeng, Wei Zheng, Marco DI Natale, Alberto Sangiovanni-Vincentelli Optimization of Task Allocation and Priority Assignment in Hard Real-Time Distributed Systems, ACM Trans. Embed. Comput. Syst., vol. 11 n°4, pp1-30, 2013.

[Zheng-2007] Wei Zheng; Qi Zhu; Di Natale, M.; Vincentelli, A.-S., "Definition of Task Allocation and Priority Assignment in Hard Real-Time Distributed Systems," Real-Time Systems Symposium, pp.161,170, 3-6 Dec. 2007.

9 Offline Scheduling Algorithms for DHRTS

9.1 Motivation

The Task *T4.1 – Offline adaptation strategies in mixed criticality* aims at generating offline configurations for mixed-criticality applications satisfying specified global constraints for the DREAMS platform architecture. The DREAMS platform architecture comprises of a distributed network where each processing node could be a multi/many-core chip. The generic distributed hard real-time systems (DHRTS) architecture (inspired by the MARS system [DRS89]), comprising of processing nodes connected by a network, could be considered as the predecessor of the proposed DREAMS platform architecture.

Since the 1980's, the offline scheduling in generic DHRTS architecture evoked considerable interest in the real-time community. The problem of computing an offline schedule for DHRTS is NP-Hard. Thus, different approaches have been proposed in the literature to solve this problem in the past 20-30 years. These approaches could act as a starting point for the construction of the offline scheduler for mixed-criticality applications in DREAMS. Thus, in section 9.2, we present various approaches available in the literature for computing an offline schedule meeting the specified temporal constraints and complex constraints in DHRTS.

Offline scheduling is based on the time-triggered (TT) paradigm. In TT paradigm, the system initiates activities only at pre-defined points in time. The main advantage of the TT paradigm is slot-level determinism [Foh12]. However, this also results in a major drawback - inflexibility i.e. what is not known a priori cannot be scheduled at runtime. This restricts the choice of task model for the system designer, when considering TT paradigm. Another paradigm - the event-triggered (ET) paradigm, where the occurrences of events in the system initiate activities does not have this issue of inflexibility. In ET paradigm, activities about which we do not have complete information a priori, can also be handled at runtime.

The approaches presented section 9.3.1 and 9.3.2 aim to provide flexibility in offline scheduling by integrating the benefits of ET paradigm. This results in a tradeoff. Now, the system is not deterministic at slot-level but only at interval boundaries.

In section 9.3.3, we present mode-change algorithm that considers the change in application characteristics based on the system state and environment to pre-compute different scheduling tables. The system switches between these scheduling tables at runtime based on the transition rules and mode-change semantics considered offline. In section 9.3.4, 9.3.5, and 9.3.6, we show the initial approaches that have been recently presented for scheduling of mixed-criticality applications in DHRTS. Finally, in section 9.4, we identify the shortcomings and challenges towards extending of the available approaches w.r.t. scheduling of mixed-criticality applications in DREAMS.

9.2 Offline Scheduling Table Construction

Kopetz defines offline scheduling as: "...making[] scheduling decisions at the compile time" [Kop11]. Offline scheduling involves the construction of scheduling table that represents a feasible offline schedule. A feasible schedule is one that meets specified complex constraints and temporal constraints of a task set. The least common multiple of the periodic tasks of the task set, called the hyperperiod, denotes the length of the schedule (when the deadlines of periodic tasks are less than or equal to their respective periods) represented by the scheduling table. At runtime, the dispatcher simply executes the decisions in the scheduling table, leading to minimal runtime overheads. When the time equal to the hyperperiod is reached, the scheduling table is usually repeated again. In

offline scheduling, periodic tasks form the base load of a system. This is because the timing constraints of the periodic tasks are known before the runtime of the system. Thereby, they can be scheduled in the offline phase.
In this section, we present various approaches available in literature to generate feasible offline schedule.

9.2.1 Branch and Bound Based Search

9.2.1.1 Hou and Shin 1992

Hou and Shin [HS92] consider the problem of task allocation, task scheduling and–network scheduling in DHRTS. The task model considers periodic tasks with precedence constraints, resource constraints and communication constraints. The system model is based on preemptive scheduling policy, homogeneous processing nodes (PNs) and time-triggered (TT) paradigm. SEND-RECEIVE-REPLY primitive is used for communication between PNs. In [HS94], the authors extend the proposed algorithm in [HS92] to further consider fault-tolerant constraints.

The authors make the following assumptions:

1. Each task can be decomposed into smaller units called modules.
2. The network and protocol support time-constrained communications, and the worst-case delay is bounded and predictable.
3. No restriction is imposed on the topology of the communication subsystem.
4. Each PN and each network link are assumed to fail independently with exponential rates.
5. The time required by an inter-module communication (IMC) within a processing node (PN) is assumed to be negligible, while that between two different PNs is expressed as the product of the IMC volume (measured in data units) and the nominal delay (measured in time units per data unit) between the two PNs on which the communicating modules reside.

The proposed algorithm aims at maximizing the value of the performance measure probability of no dynamic failure (P_{ND}). P_{ND} represents the (a) probability that tasks in the system will finish execution by their respective deadlines (b) reliability of communication links and PNs [HS92] [HS94]. The algorithm uses a tree-based search based on branch and bound technique where, at each level in the search tree, a module is considered for allocation. The branching test is used to determine a PN from the list of available PNs to which a module can be allocated. The bounding test is used to effectively prune vertices early in the search tree that do not improve the value of the performance measure i.e. P_{ND} . The terminal vertex with largest value of the performance measure P_{ND}, represents the feasible allocation of modules to PNs. Then, the Module Scheduling Algorithm (MSA) based on algorithm A [PS93] that minimizes maximum module tardiness provides the offline task schedule for each PN.

For the evaluation of the algorithm, the authors consider a maximum of 50 modules and 30 PNs. An important observation is that the percentage of vertices in the search tree visited decreases as the number of modules increase and/or number of PNs increase. This is because the bounding function helps in reducing the number of vertices expanded in the search tree. Another observation to note is the algorithm tends to allocate modules with tight timing constraints on the same PN.

9.2.1.2 Abdelzaher and Shin 1999

Abdelzaher and Shin [AS99] consider the problem of task scheduling and network scheduling. The task model comprises of periodic tasks having precedence constraints, resource constraints and communication constraints. The system model[3] considered is based on preemptive scheduling policy and TT paradigm.

[3] No mention is made on whether the PNs used are homogeneous or heterogeneous.

The authors make the following assumptions:

1. Tasks are pre-allocated to PNs.
2. A task is composed of one or more modules.
3. Each task resides permanently on one PN.
4. Modules residing on different PNs communicate using message passing based on a time bounded communication paradigm.
5. Communication among modules residing on the same PNs, is assumed to incur a fixed overhead, which is included in the execution time of the sending module.
6. Modules unlock/lock all their required resources together and hold them throughout the entire interval of their execution.

The algorithm is based on the branch and bound technique and aims to minimize the value of the performance measure *maximum task lateness* (MTL). *Maximum task lateness* represents the maximum value of task lateness amongst all possible tasks in the system. The tasks pre-allocated to each PN are scheduled using EDF with Deadline Inheritance (EDF-DI) scheduling policy, where a module blocking other modules inherits the earliest deadlines amongst the modules blocked by it.

Next, we briefly explain the working of the algorithm. The algorithm first generates an initial message priority order, where messages with tighter relative deadlines have higher priorities. Then, the start times of modules waiting for messages from their predecessor modules are modified to account for message delay bounds. In the next step, the offline schedule for each PN is generated and the value of the performance measure MTL is computed. If the value of the performance measure computed in the previous step is less then or equal to 0, the generated offline schedules for each PN are feasible. If not, the branching function is used to generate vertices such that the module with maximum task lateness can finish earlier. This is possible by either modifying the message priority order or scheduling some module earlier, such that the module under consideration finishes earlier. The bounding function is used to prune the vertices that are unlikely to result in a lower value of the performance measure MTL than the current lowest MTL value for a feasible offline schedule.

The algorithm is evaluated for a maximum of 4 PNs and 300 modules. An observation made by the authors is, if the value of the performance measure MTL is greater than 1, the algorithm tries to reduce the maximum task lateness of the module resulting in this value of MTL. In case the algorithm fails to minimize the MTL of this module (thereby of the schedule), the algorithm terminates even though there may be other modules whose task lateness could be minimized.

9.2.1.3 Peng and Shin 1989, Peng et al. 1997

[PS89] [PSA97][4] consider the problem of task allocation and task scheduling. The task model considered is a set of periodic tasks having precedence constraints and communication constraints. The system model considers preemptive scheduling policy, heterogeneous PNs and TT paradigm. In [PS89], SEND-RECEIVE-REPLY and QUERY-RESPONSE primitives are used for communication between PNs.

The algorithm works under the following assumptions:

1. Each task consists of one or more task modules.
2. All modules of the same task are allocated to the same PN.
3. Precedence constraints may exist between modules of the same or different tasks.

[4] [PS89] is the conference paper presenting the algorithm. [PSA97] is a journal paper and is an extension of the [PS89]. [PSA97] also presents the evaluation of the algorithm.

4. A task module can be of one of the two types: computation module or communication module.
5. Execution time of a communication module depends on the PN it executes and on whether the communication module it communicates with is on the same PN or on a different PN.

The algorithm is based on the Branch and Bound technique and aims at minimizing the performance measure *maximum normalized response time* (MNTRT), where, MNTRT is the maximum value of normalized task response time amongst all possible tasks in the system. The algorithm uses a tree-based search based on the B&B technique. At each level in the search tree, it considers an unallocated task for allocation. The branching function is used to select the vertex with the minimum vertex cost for expansion, where vertex cost is the lower bound value of MNTRT. The bounding function is used to prune the vertices that are unlikely to result in a better solution compared to the one already found so far. This is done by calculating the vertex cost for all non-terminal vertices. The complexity of the calculation of vertex cost $O(nM^2)$, where n is number of PNs and M is number of modules. Those vertices that have MNTRT value less than the already found solution (feasible offline schedule) are pruned, thereby reducing the search space. All terminal vertices resulting in the value of MNTRT being less than or equal to 1, represent feasible task schedules. A modified version of algorithm 'algorithm A' [PS93] is used to generate the offline task schedule for each PN.

The algorithm is evaluated for a maximum of 14 tasks with a total modules 140 and 4 PNs. In such a case, the percentage of vertices expanded in the search tree is not more than 0.0000004% of the entire search space. However, when the number of PNs increases to 6 (and the number of modules are reduced to 80), a maximum of 0.004% of the vertices are expanded. This shows that the bounding function is effective in reducing the number of expanded vertices. Also, the authors observed that the algorithm is considered to function best when the execution time of modules is comparable to the communication time i.e. for heavily communicating tasks.

9.2.2 Clustering Algorithm based Search

Faucou et al. [FDP00] consider the problem of task allocation, task scheduling and network scheduling. The task model comprises of periodic tasks with precedence constraints and communication constraints. The system model considers non-preemptive scheduling policy with homogeneous PNs. The scheduling of tasks is based on TT paradigm. However, for communication between PNs, the authors consider in separate experiments, TT communication (TDMA based) and ET communication (CAN Bus), in order to understand if this affects schedulability of the task sets.

The authors make the following assumptions:

1. All PNs are identical.
2. Each task is composed of one or more modules.
3. Behavior of modules follow ``data flow'' model: when activated (after the completion of all its predecessors), a module starts by reading input data; when it computes results from them, finally it sends those outputs to its successors.
4. If two communicating modules are added to the same PN, the communication uses shared memory (and the communication delay is negligible). If they are allocated to different PNs, the shared broadcast bus is used to achieve the communication. To handle this, there is a network adaptor on each PN.
5. In case of carrier sense multiple access with collision avoidance (CSMA/CA), a frame contains only a single data and data is contained in only one frame (this imposes limit on the size of data field in the frame = 8 bytes).
6. In case of TDMA-based network, every PN is assigned exactly one TT slot in the time division multiple access (TDMA) round which is fixed
7. All tasks are independent.

The algorithm is divided into two parts: Clustering and Mapping & Scheduling. In the first part, it aims to form clusters of communicating modules based on the parameter *communication factor* (CF). The system designer initially sets CF to 0. Next, the network utilization (NU_{ij}) of each communicating modules pair (i,j) is computed. NU_{ij} is the ratio of the communication time needed by the communicating module pair C_{ij} to the period of the sending module T_i. If this computed value is greater than the product of the communication factor and that of the maximum network utilization, then the communicating module pair (i,j) is clustered. Since, CF is initially set to 0, this means that clusters of all communicating modules are formed. In the second part, a tree-based search guided by latest start times is used for mapping and scheduling of tasks with the aim to find a feasible schedule. Communicating modules in a cluster are allocated to the same PN. If a feasible schedule is not found, then the value of CF (in the first part) is incremented by 0.1 and the algorithm iterates until either value of CF is 1 or a feasible schedule is found.

The proposed algorithm is evaluated for a maximum of 5 tasks, where each task may have 6 to 10 modules. The number of PNs ranges from 3-5. When compared against genetic algorithm (GA) based approach (presented in section 9.2.4 [MD98]), the success ratio of the presented approach is better than GA based approach, in both cases when either TDMA bus or CAN bus is used for communication.

9.2.3 Clustering + Branch and Bound based Search

Ramamritham [Ram90] considers the problem of task allocation, task scheduling and network scheduling. The task model considers periodic tasks with precedence constraints and communication constraints. The system model comprises of non-preemptive scheduling policy, non-homogeneous PNs and TT paradigm.

The algorithm works under the following assumptions:

1. Each PN has one processing element and a given set of passive resources.
2. Each task is composed of a set of modules.
3. Modules of a task can be allocated to different PNs.
4. Tasks are independent i.e. no precedence constraints exist between different periodic tasks.
5. Each module executes sequential code, and communication occurs when one completes execution and sends its result to the other, before the latter begins execution.
6. The PNs are connected by multiple-access network.
7. Communication between modules on the same PN incurs zero delay.
8. The module throughout its execution needs all the specified resources and the resources allocated to module are only released at the end of its execution.

The algorithm aims to find a feasible task and message schedule and is divided into two parts: Clustering and Mapping & Scheduling. The first part forms clusters of communicating modules in order to minimize the inter-PN communication. Then, in the second part of the algorithm, modules are mapped to PNs using a tree-based heuristics search based on branch and bound technique and then scheduled non-preemptively. Here, the clustered communicating modules are considered for allocation to the same PN.

For the evaluation of the algorithm, the authors consider a maximum of 3 periodic tasks having 4, 8 and 12 modules respectively and a maximum of 6 PNs. A key observation by the authors is that if a feasible schedule exists, the algorithm is likely to find it in the initial chosen path itself. If not, it is highly likely that the task set is infeasible. In addition, it is observed that backtracking only results in marginal improvement of less than 3.5% in cases of low laxity value when a maximum of 100 backtracks are allowed. This however, results in 30 times increase in cost when a feasible schedule is

not found. In cases when a feasible schedule is found, the observed increase in costs is four times compared to the case when backtracking is not allowed.

9.2.4 Genetic Algorithm based Search

Monnier et al. [MD98] consider the problem of task allocation, task scheduling and network scheduling. The task model considers a set of periodic tasks with precedence constraints and communication constraints. The system model is based on non-preemptive scheduling policy, homogeneous PNs and TT paradigm.

The algorithm makes the following assumptions:

1. The periodic tasks are independent of each other and their deadlines are equal to their respective periods.
2. Each task may have one or more modules.
3. Each module is non-preemptable.
4. All PNs are connected by a bus, where only one communication at a time is allowed.
5. The contention-free communication delay does not depend on the distance between the PNs, but on the amount of data exchanged.
6. Communication between two communicating modules assigned to the same PN is negligible and is set to zero.
7. All tasks are independent.

The proposed algorithm to find a feasible task and message schedule is based on the genetic algorithm search heuristics with roulette wheel selection strategy. The algorithm aims to minimize schedule tardiness, where schedule tardiness is simply the sum of tardiness of all tasks. The algorithm tries to determine the allocation of modules and a priority order of modules such that precedence constraints and temporal constraints are met. The initial population is generated randomly and then the algorithm continues generating new children until a zero tardiness schedule is found or the maximum allowed number of individuals that could be produced is reached.

The evaluation of the algorithm is shown for a maximum of 6 tasks having 6-10 modules and 6 PNs. The authors compare the success ratio of their algorithm against two other approaches: *simulated annealing* (SA) based approach and clustering (CA) based approach. When the number of PNs is less than or equal to 2, the *success ratio* (SR) of the presented algorithm is 100%. As the number of PNs increases to 5, the SR of the presented algorithm is better than the other two approaches. But, when the number of PNs is increased to 6, the success ratio of the presented algorithm is worse than that of the other two approaches. . Also, in this case, even when the number of generations of the presented algorithm is doubled, no improvement in performance is observed. The performance could be improved by considering problem specific genetic operators in the algorithm. The authors also point out that the main advantage of their algorithm when compared against SA based approach is that during optimization, the likelihood of their algorithm getting stuck at local minima is reduced since it evaluates multiple schedules at the same time.

9.2.5 List Scheduling based Search

Girault et al. [GKM03] consider the problem of task allocation, task scheduling and network scheduling. The authors consider precedence constraints, communication constraints and fault-tolerant constraints. However, they make no mention of the frequency task activations i.e. whether it is periodic, aperiodic or sporadic. The system model considers is non-preemptive scheduling policy, heterogeneous PNs and TT paradigm.

The authors make the following assumptions:

1. The distributed system consists of PNs and network where the network can have many communications links.
2. Communication operation is send/receive where the send operation is non-blocking and the receive operation blocks in the absence of data.
3. All values, returned by the replicas of the same operation in the same iteration, are identical.
4. At each input event from the sensors, algorithm is executed repeatedly to compute the output events.
5. There are three kinds of operations: computation, memory and external input/output operation. (Note that these operations are referred as tasks below for clarity of discussion)
6. Deadline is specified only for the complete schedule i.e. the given set of tasks must finish by the time specified as deadline.

The algorithm is based on list scheduling heuristics and uses *schedule pressure* as the performance measure. *Schedule Pressure* is a cost function that provides information about how the scheduling of a task on a certain processor increases the critical path length. The algorithm aims to reduce the critical path length by minimizing the *schedule pressure*. This is done by first determining the task with the maximum schedule pressure and then trying to minimize its earliest start time by scheduling its latest immediate predecessors earlier. If the length of the determined schedule is less than the specified deadline, then the schedule is feasible. The algorithm is evaluated for a maximum of 80 tasks and 4 PNs.

9.2.6 Simulated Annealing based Search

Cheng et al. [CSA95] consider the problem of task allocation, task scheduling and network scheduling. The task model considers periodic tasks having precedence constraints and communication constraints. The system model considers non-preemptive scheduling policy, non-homogeneous PNs and TT paradigm.

The authors make the following assumptions:

1. Each task is composed of one or more modules.
2. Modules of tasks communicate with each other using inter-module communication (IMCs).
3. Each module may have more than one replica and a module may start its execution after receiving necessary data from a replica of each of its predecessors.
4. The offset of the periodic task is 0.
5. Each IPC occurs at pre-specified time according to the schedule.
6. At most, one communication can occur at any time on the network. All the communications are via message passing mechanism.
7. There is no shared memory in the system.
8. Time required by a local IMC is negligible compared to the transit time needed for an inter-processor communication (IPC).
9. Modules of a task can be assigned to different processors.
10. The network is associated with the nominal delay (ND) for transmitting one unit of data from one processor to another.

The algorithm, based on simulated annealing based search with replication (SA/R), aims to find a feasible such that the value of the performance measure *energy function* (E) is 0. *Energy function* E depends on the module and message completion times and their respective deadlines. If its value is zero, it represents a feasible schedule. On the other hand, if the value is greater than zero, the computed schedule is not feasible. If the IPC associated with a module leads to infeasibility of the task set, the algorithm considers replication of modules on multiple PNs, thereby making a tradeoff between communication time and computation time i.e. replicate a module on another PN.

Now, we briefly describe the working of the algorithm. In the first step, the algorithm, as its starting point, generates an initial schedule by assigning modules directly to PNs and determining task and message schedules. In the second step, the energy function value of the generated task schedule is calculated. If the value of the energy function is zero, the generated schedules are feasible and the algorithm terminates. If not, then, here in third step, the algorithm identifies the IPC's resulting in bottleneck using dynamic programming and replicates the sender modules of these bottleneck IPCs. The complexity of the replication step is $O(K^2N)$ where, K represents the number of IPCs and N is number of modules. In the fourth step, the value of the *energy function* is recomputed. In case, this leads to a feasible schedule, the algorithm terminates. If not, in the fifth step, a new schedule is obtained by using one of the following operators on the schedule: swap operator[5], random operator, merge operator. The algorithm then jumps to the second step and iterates until either a feasible schedule is found or the maximum time allotted for the search is reached.

The authors evaluate the algorithm for a maximum 55 modules and 7 PNs. They also compare the presented algorithm SA/R against the classical simulated annealing approach (without replication). The observation from the results is that SA/R needs comparatively less iterations to find a feasible schedule as compared to the classical simulated annealing approach thereby increasing the schedulability of the task set.

9.2.7 Other Heuristics based Search

9.2.7.1 IDA based Search*

Fohler and Koza [FK90] consider the problem of task scheduling for periodic tasks having precedence constraints and communication constraints. The system model considers preemptive scheduling policy with homogeneous PNs and TT paradigm. The authors assume that the tasks are pre-allocated to PNs. The algorithm uses a tree-based search based on iterative deepening A* (IDA*) heuristics, where each level in the search tree corresponds to one unit of execution time. It aims to determine a schedule for the given set of tasks such that the length of the schedule is less than or equal to the specified *time until response* (TUR), which is essentially a deadline. The algorithm, at each vertex in the search tree, calculates the cost of the path from root vertex to the current vertex and estimates the cost from the current vertex to the terminal vertex. Based on this combined cost, the algorithm chooses a vertex from the possible vertices for expansion. Heuristic function based on TUR guides the search towards the solution and effectively prunes paths unlikely to result in a feasible schedule. The key advantage of choosing IDA* is the small memory footprint as information related to the current vertex only needs to be stored.

9.2.7.2 Stepwise Enlargement of Schedule

Verhoosel et al. [VLH91] consider the problem of task allocation and task scheduling for periodic tasks having precedence constraints, resource constraints and communication constraints. The system model considers non-preemptive scheduling policy, homogeneous PNs and TT paradigm

The authors make the following assumptions:

1. Each periodic task consists of infinite sequence of jobs.
2. Jobs are divided into non-preemptable blocks and all job characteristics are known a priori.
3. A PN consists of a number of resources of different types, such as processors, sensors, actuators, disks etc. Resources of the same type are homogeneous. e.g. functionality and speed of all processors are equal.
4. Communication delay within a PN is insignificant (considered 0 in the algorithm) compared to communication delay between the PNs (considered fixed).
5. Communication delay is constant and equal to the maximum delay.

[5] In [SSA 1995], what we call here 'operator' is referred as 'mode'. However, we avoid using 'mode' for in context of offline scheduling in RTS, it has a special meaning as defined in terminology.

6. Different instances of a task can execute on different PNs.
7. A resource can be assigned to only one task at a time and a task can only start if all resources required for its execution are available.
8. Tasks of the same job need access to common data and must be scheduled on the same PN.
9. Resource requirements of a task may be specific or non-specific.
10. Each block has a deadline assigned to it.

The algorithm uses a tree-based search considering three different heuristic functions: job ordering heuristics function (JOH), PN selection heuristics function (PSH), and backtracking heuristics (BH). Each level in the search tree represents a job allocation to a PN. In the first step, the algorithm assigns indices to the jobs based on increasing value of JOH. Jobs having lower JOH value, have tighter constraints compared to other jobs with comparatively higher JOH value. Jobs are considered for allocation in ascending order based on their index value. The number of possible vertices to be considered for expansion at each vertex equals to the number of PNs that satisfy resource requirements. The algorithms selects the vertex for expansion i.e. allocation of job, based on the value of the PSH function. Out of the valid vertices for expansion, vertex with the highest PSH is chosen for expansion. The algorithm then checks if the current allocation leads to infeasible schedule. If yes, it backtracks based on the backtracking heuristics and keep track of times each job lead to infeasible schedule (will be used later to alter job priorities). If not continue with the allocation and scheduling, until all tasks are schedules and a feasible schedule is found. The authors evaluate the algorithm for a maximum of 5 PNs = 5 where each PN has 4 different kinds of resources with different resource capacities.

9.2.8 Handling Periodic Tasks with Deadlines greater than Periods

Fohler and Ramamritham [FR97] consider the problem of task scheduling for periodic tasks. The major difference between their works and all other algorithms presented before in section 9.2 is that it relaxes the condition that the periodic tasks must have deadlines less than or equal to their respective periods. The authors refer the algorithm by meta-algorithm for it works over the already discussed offline scheduling algorithms like [Ram 1990]. The meta-algorithm only takes control at the hyperperiod boundary, L, where L represents the lowest common multiple of all periodic tasks in the task set. The meta-algorithm is based on TT paradigm. The scheduling policy (preemptive or non-preemptive) and the constraints considered by the meta-algorithm depend on the algorithm over which it works.

The authors make the following assumptions:

1. Each task is composed of one or more modules.
2. The network is a multiple-access network.
3. Communication media and protocols that have predictable communication delays such that knowing the arrival times and characteristics of the message, the time when the message is delivered can be predicted.
4. There may be other assumptions depending on the offline scheduling algorithm chosen above which this meta-algorithm works.

The algorithm is used for scheduling of periodic tasks having deadlines greater then periods. One of the already presented offline scheduling algorithm is schedules the modules until the hyper period boundary. At the hyper period boundary, as mentioned before, the meta-algorithm takes control to check the occurrence of the following three conditions:

A. all modules of the task set have been scheduled in the interval [i*L, j*L) where, ‘i’ and ‘j’ are some non-negative integers such that ‘i’ is greater than ‘j’.

B. if set of sub-tasks that have not been scheduled at the current hyperperiod boundary (j*L), are a subset of sub-tasks that not been scheduled at any other previous hyperperiod boundary (i*L), or
C. if the overlaps for the current hyperperiod boundary (j*L), have remaining execution time smaller than or equal to their corresponding sub-tasks at some previous hyperperiod boundary (i*L).

It may happen that both conditions B and C occur together. Either way, the meta-algorithm deals with each of the three conditions and when B & C occur together in special ways to generate a feasible schedule. In case a non-preemptive scheduling policy is chosen, the meta-algorithm forces the overlaps to execute non-preemptively at hyperperiod boundary by modifying their start time and deadline. Overlaps are modules that started their execution before the hyper period boundary and still need to execute for their left execution time beyond the hyper period boundary to meet their deadline.

The meta-algorithm is evaluated for a maximum of 10 PNs and 5 communication links. The farthest schedule found by it lies between 7*L and 11*L, where L is the hyper period. The authors also mention that the algorithm can be used with mode-changes.

9.3 Offline Scheduling Table Construction with Flexibility

Offline scheduling, in order to provide determinism, assumes that all information about the system is known a priori. For example, the arrival times, deadlines, WCET for the tasks and the frequency of activations of the tasks. This leads us towards the main limitation of offline scheduling that it provides no flexibility at runtime, as the execution TT-slots of tasks are pre-determined. And at runtime, the online scheduler just executes the decisions taken offline through the use of scheduling table. In this section, we present algorithms that provide runtime flexibility with determinism.

9.3.1 Slot Shifting: Handling Aperiodic Tasks

Fohler [Foh95] considers the problem of task scheduling combining offline and online scheduling to provide determinism and runtime flexibility. The task model considers periodic tasks, firm aperiodic tasks and soft aperiodic tasks. The periodic tasks are allowed to have precedence constraints and communication constraints. The system model comprises of preemptive scheduling policy and TT paradigm while integrating event-triggered (ET) paradigm.

The assumptions considered are as follows:

1. Scheduling blocks are formed at each PN combining modules that have same start module and same end module in precedence graph. This is done so as to reduce the number of modules to be considered. End modules can be either exit modules in a precedence graph or modules sending inter-PN messages. Start modules can be either entry modules in the precedence graph or tasks receiving inter-PN messages [Foh94]
2. Henceforth, to keep it simple, scheduling blocks are referred as tasks.
3. Tasks communicate with each other with data read at the beginning and written at the end.
4. Aperiodic tasks are ready to run at the time of their arrival.
5. The time model is discrete.
6. Communication medium is slotted and pre-scheduled.
7. Protocols with bounded transmission times like e.g., TDMA or token ring are applicable i.e. message transmission times are fixed.

The algorithm, referred as ‘slot shifting’ in the literature, integrates both, offline and online scheduling, to provide determinism and flexibility at runtime. The reclaiming of unused resources is performed at runtime to serve firm and soft aperiodic tasks. The unused resources comprise of free

TT slots in the offline scheduling table and the TT slots freed when guaranteed tasks finish earlier than their specified computation time.

Slot shifting comprises of two phases: offline phase, and online phase. In the offline phase, the first step comprises of construction of offline scheduling table for periodic tasks using a tree-based search based on IDA* heuristics (as presented in section 9.2.7.1). In the second step, the computed scheduling table is used to form disjoint execution intervals based on task deadlines. In the last step of offline phase, the algorithm determines the *spare capacities* for all execution intervals. *Spare capacity* (sc) of an interval is the maximum amount of idle time in an interval by which tasks belonging to this interval can be postponed such that all other tasks in the system meet their respective deadlines.

In the online phase, the online scheduler decides which task to schedule at the start of TT slot. It schedules a soft aperiodic task if the spare capacity of the current interval ($sc(I_c)$) is greater than zero and a soft aperiodic task arrives in the system. In addition, the spare capacity is decremented by one. When there is no soft aperiodic task in the system, and the spare capacity of the current interval is greater than zero, then EDF-based scheduling is performed. However, if the spare capacity of the current interval is zero, then a guaranteed task (could be offline guaranteed task or firm aperiodic) is scheduled. Note, however, that a firm aperiodic task can only be guaranteed on its arrival if and only if, the spare capacity of all intervals until the deadline of the firm aperiodic task is greater than or equal to the computation time required the task. If guaranteed, the task is accepted and spare capacities are updated. If no task is scheduled at some TT slot, then the spare capacity is decremented by one.

The algorithm is evaluated for a maximum of 4 PNs using 1600 task sets., based on the metric *guarantee ratio* (GR). *Guarantee ratio* represents ratio of the number of firm aperiodic tasks guaranteed to the total number of firm aperiodic tasks released in the system. When the combined periodic and aperiodic load is 100%, GR is equal to 70% that shows the effectiveness of algorithm in reclaiming unused resources at runtime to provide flexibility.

9.3.2 Handling Aperiodic and Sporadic Tasks

Isovic and Fohler [IF09] consider the same problem of task scheduling while providing flexibility at runtime. The major difference between this work and that of [Foh95] is in the task model. In this work, task model considers periodic, sporadic and aperiodic tasks. The periodic tasks may have precedence constraints and communication constraints. The system model comprises of preemptive scheduling policy considering TT paradigm while integrating event-triggered (ET) paradigm.. The assumptions for the presented algorithm are same as that of slot shifting (section 9.3.1). In addition, the algorithm also assumes that migration of tasks is not allowed at runtime.

The algorithm, like slot shifting, integrates offline and online scheduling to provide both determinism and flexibility at runtime. It is the only algorithm till date that considers handling of periodic, sporadic and aperiodic tasks. The algorithm is divided into two phases: offline phase, and online phase. In the offline phase, the initial steps of computation of offline scheduling table for periodic tasks, formation of disjoint execution intervals and calculation of spare capacities are same as for slot shifting (section 9.3.1). After these steps, the next step involves transformation of sporadic tasks into pseudo-periodic tasks. Then the algorithm considers one sporadic task at a time and tries to guarantee all of its instances over the already scheduled periodic tasks. The sporadic tasks need to be guaranteed only at specific instants in time in each interval (until hyper period) called critical slot. No reservation of resources is made in the offline phase for sporadic tasks.

In the online phase, a two priority level EDF-based scheduling is used for scheduling of tasks at each TT slot. If the spare capacity of an the current interval is greater than zero, then it reflects that the priority level is ‘1’ and classic EDF scheduling is used. However, if the spare capacity is zero, then it means the priority level is ‘2’ and an offline guaranteed task needs to be scheduled. Acceptance and guarantee test are performed for soft and firm aperiodic tasks on their arrival as described previously in section 9.3.1. At runtime, the online scheduler reclaims unused resources i.e. free TT

slots in offline scheduling table, freed TT slots when tasks finish earlier than their specified computation time to service sporadic tasks and firm and soft aperiodic tasks.

The algorithm is evaluated for 800,000 different interactions of periodic, sporadic and aperiodic tasks. Even when the combined utilization of periodic, sporadic and aperiodic load is 100% and the deadline of aperiodic tasks is twice of their execution times, the *guarantee ratio* (GR) is greater than 55% in the case when the algorithm keeps track of sporadic task arrivals at runtime). This shows that the algorithm is effective in providing runtime flexibility by guaranteeing (in this case) 55% of the firm aperiodic tasks released in the system. However, when the algorithm does not keep track of sporadic tasks arrivals, the GR drops to approximately 20%.

9.3.3 Mode Change Algorithm

Fohler [Foh93] considers the problem of task scheduling and network scheduling with mode changes. The task model comprises of periodic tasks having precedence constraints and communication constraints. The system model comprises of preemptive scheduling policy and TT paradigm.

In TT systems, a mode refers to a phase of operation of a real-time system. For example, an aircraft goes through various phases during its flight: take-off, normal-flight and landing phases. The tasks performed in each of these phases change. So, a mode best captures a particular phase of operation of a real-time system, through a corresponding scheduling table. Each mode has its own precedence graph represented as directed acyclic graphs. Now, a mode-change refers to the deterministic switching among a number of modes (essentially scheduling tables) such that the offline-scheduled real-time system is able to adapt to changing environmental situations. When compared to slot shifting based approaches to provide runtime flexibility (sections 9.3.1, 9.3.2) mode change results in a system wide scheduling table change. In the discussed work, the authors consider two kinds of modes: continuous mode and transition mode. All continuous modes represent phase of operations of a real-time system. A transition mode on the other hand, is the phase of operation when switching to a continuous mode. The transition condition for a mode change needs to be specified. Mode change can even be requested during transition mode: handled in same way as occurring in the continuous mode. Three possible semantics exist for mode changes [JLM 1988]. In immediate change semantics, mode change is instantaneous as all current tasks are aborted. In change at the end of current scheduling table semantics, a mode change can only occur once all tasks in the current scheduling table are completed .In change after completing certain tasks semantics, a mode change can only occur once specific tasks have been completed from the current scheduling table. The presented algorithm, based on IDA * heuristics (section 9.2.7.1) constructs concurrently scheduling tables for all modes considering “immediate change” and switch through property. If the mode change considers switch though property while construction of scheduling table, it means that switching between modes is possible at each TT slot.

9.3.4 TT Scheduling of Mixed Criticality Systems

Baruah and Fohler [BF11] consider the problem of job scheduling in mixed criticality systems (MCS) for a single PN. The system model comprises of preemptive scheduling policy with TT paradigm. The mixed-criticality (MC) model is based on the Vestal model [Ves07] considering two criticality levels, HI and LO. HI mode corresponds to Certification Authority (CA) assumptions while LO mode considers the less pessimistic system designer assumptions. Each job belongs to only one of the two criticality levels, HI or LO, and the computation time of the HI criticality job in HI criticality mode is greater than or equal to its computation time in LO criticality mode.

The authors make the following assumptions:

1. For each job, it’s criticality level, arrival times, deadline, computation times for different criticality levels are known before runtime.
2. The system is restricted to two criticality levels – HI and LO.

3. For all HI criticality jobs J_i, computation time needed by J_i in HI mode $C_i(HI)$ is always greater than or less than the computation time needed in LO mode $C_i(LO)$
4. For LO criticality jobs, computation time needed by them in HI and LO modes is same.

The proposed algorithm aims to determine two scheduling tables, S_{LO} and S_{HI}, such that *switch through property* is followed i.e. switching from scheduling table S_{LO} to S_{HI} is possible at each instant in time using a mode-change. This is important for, as mentioned before, LO mode corresponds to system designer's assumptions that are normally less pessimistic than the CAs pessimistic assumptions. This difference in pessimism results in some tasks having two different computations times, $C_i(LO)$ based on system designer assumptions and the other $C_i(HI)$ based on CAs assumptions, where $C_i(LO) <= C_i(HI)$. At runtime, the system starts execution in LO mode. If at any point in time, a HI criticality job J_i exhausts its computation time in LO mode $C_i(LO)$ but still has not completed execution, then a mode change results in transition from LO to HI mode, where job J_i is granted additional computation time equal to the difference ($C_i(HI)$ - $C_i(LO)$) to finish its execution by its deadline.

The algorithm involves three main steps. In the first step, the algorithm determines a priority order based on Audsleys' fixed priority scheduling approach for all jobs such that when a HI criticality job exceeds its LO criticality WCET, mode change to HI mode is made and HI criticality job is provided additional execution time based on CA assumptions such that it still meets its deadline. The second step constructs the scheduling table S_{LO} based on the job priority order determined in step 1 using preemptive fixed priority-based scheduling. Finally, the last step involves construction of scheduling table S_{HI} based on the job priority order determined in step 1, again using preemptive fixed priority-based scheduling

The major theoretical results in the work is that if the given job set is clairvoyant schedulable on a given processor, then it is schedulable by the presented approach on a $(\sqrt{5} + 1)/2 \approx 1.62$ times as fast processor. The authors also mentions the presented approach is extendible to multiple criticality levels (more than 2) and can also deal with periodic tasks by determining all jobs of the periodic tasks until the hyperperiod and then applying the discussed approach.

9.3.5 Mixed Criticality and TT Legacy Systems

Theis et al. [TFB13] consider the problem of mixed criticality task scheduling in TT legacy systems without modifying the legacy TT scheduling tables. The task model comprises of periodic and aperiodic tasks. The constraints considered depend on the offline scheduling algorithm initially used to construct scheduling tables. The system model comprises of preemptive scheduling policy considering TT paradigm while integrating event-triggered (ET) paradigm for task activation. The authors make the same assumptions as considered by Baruah and Fohler [BF11] in (Section 9.3.4). In addition, the authors assume that the scheduling table for a PN is already given.

The algorithm is based on slot shifting [Foh95] (section 9.3.1) and considers scheduling of mixed criticality tasks of the given scheduling table TT legacy scheduling table is not modified. Like slot shifting, the algorithm comprises of two phases: offline phase and the online phase. In the offline phase, the steps to determine the offline scheduling table and construct disjoint execution intervals based on task deadlines are similar to slot shifting (section 9.3.1). A minor change is made in the step that calculates the spare capacities for each interval. Now, the algorithm calculates two spare capacities for each interval, one for each criticality levels (HI and LO). As the algorithm considers that modification of the legacy scheduling table is not allowed, it deals with mixed criticality scheduling at runtime. In the online phase, the online scheduler decides which task to schedule depending on the task deadline, criticality and spare capacity available (at different criticality levels). Maintenance of spare capacities, acceptance test and guarantee test are quite similar to slot shifting (section 9.3.1). It is important to note, that the authors allow the transition from both LO to HI mode and vice-versa, depending on the system state.

9.3.6 Mixed Criticality and TT Scheduling Table Construction

Theis and Fohler [TF13] consider the problem of job scheduling in mixed criticality systems. The system model comprises of preemptive scheduling policy and TT paradigm. The authors make the same assumptions as made by Baruah and Fohler [BF11] in section 9.3.4. In additions, the authors assume the allocation of jobs to PNs is predefined.

The authors present an algorithm that aims to construct two scheduling tables, one for each mode (LO and HI) using search tree-based on iterative deepening heuristics [Kor85] such that it is possible to switch from LO to HI mode at any point in time (switch through property). Since, the algorithm considers Vestal model for mixed criticality, it splits all HI criticality jobs J_i into J_i^{LO} and J_i^{Δ}, where, J_i^{LO} is scheduled in both LO and HI mode and J_i^{Δ} is scheduled only in HI mode. Two scheduling tables (one for each criticality level) are constructed concurrently. First, the algorithm selects a job based on EDF policy and then schedules the TT slot in LO table. The algorithm then calculates the value of 'leeway' for the current TT slot in which a job is scheduled in the LO table. A non-negative leeway for a TT slot represents that HI criticality jobs J_i^{Δ} can meet their deadlines in the HI criticality mode. The algorithm then schedules the same TT slot in HI table depending on the criticality of the job scheduled in the corresponding TT slot in LO table. Backtracking heuristic is used to save from exhaustive search. If for a job J_i the value of leeway is negative, the algorithm finds a swap TT slot such that leeway in swap TT slot is greater than or equal to current TT slot. On finding such a swap TT slot, the algorithm swaps the jobs selected for scheduling in swap and current TT slots. The time of the swapping TT slot must be after the release time of job J_i. Once all jobs are schedules in the LO table and all HI criticality in in the HI table, the algorithm terminates. It should be noted, that the transition from HI to LO mode is not defined/allowed.

The authors evaluate their approach for mixed-criticality schedule generation against the fixed-priority scheduling (FPS) approach of Baruah and Fohler [BF11]. When utilization of the LO criticality jobs is less than or equal to 70%, the presented approach has equal or better success ratio than the fixed priority approach. As the utilization of LO criticality jobs is decreased to less than 60%, the presented approach has better success ratio in most cases.

9.4 Challenges/Shortcomings w.r.t. MCS

9.4.1 Requirements Related to Scheduling in DREAMS

The requirements related to scheduling in DREAMS include:

- The processing nodes in the distributed system architecture considered in DREAMS could be many/multi-core chips.
- Task model: consider periodic tasks, aperiodic tasks and sporadic tasks (D1.1.1 R1.10.3)
- Mixed-criticality model: allow for consideration of different assurance levels (e.g. DAL A-E levels in RTCA DO-178B, SIL1-4 in EN ISO/IEC 61508) to interact and co-exists on same-networked distributed computational platform.
- Inter-partition Scheduling: support cyclic scheduling policy and preemptive fixed priority scheduling policy (D1.1.1 R2.8.1)
- Intra-partition scheduling: upto the GuestOS/partition
- Restrictions imposed by the Xtratum
    - Partition slots are fixed in time.
    - Mode change semantics: It only allows change of mode at the end of the current partition scheduling table.

9.4.2 Shortcomings of the Current Solutions

The requirements presented in the previous section are used to determine the shortcomings of the various solutions related to scheduling presented in sections 9.2 and 9.3. They are listed as follows:

- Solutions presented in section 9.2:
    - Only consider periodic task model

  - Do not consider criticality levels
  - Do not consider scheduling of partitions
  - Do not consider scheduling for many/multi-cores.
- Solutions presented in section 9.3:
  - [Foh95] [IF09] do not consider criticality levels.
  - [TF13] considers periodic and aperiodic tasks and not sporadic tasks.
  - [TF13] [TFB13] [BF11]:
    - Consider only dual criticality level based on Vestal model. However no consideration of assurance levels as considered is DREAMS is made.
    - Mode change semantics considered “immediate change” with switch through property
  - [IF09] considers all periodic, sporadic and aperiodic tasks. However, it does not consider different criticality levels.

9.4.3 Challenges

Considering the DREAMS requirements and the shortcomings of the current solutions, the following challenges lie ahead of us related to construction of offline scheduling tables in DREAMS:

- Extend/develop an offline scheduler considering
  - multiple criticality levels based on assurance levels,
  - for distributed systems where processing nodes could be many/multi-cores platforms,
  - partitions scheduled in fixed partition slots non-preemptively
  - mode-change is only allowed at the end of current partition scheduling table.

We also identified certain open questions in context of offline scheduling in DREAMS:

- Open questions:
  - In context of avionics DAL A-E, since all tasks are scheduled (inside partitions) in the same scheduling table:
    - What does a mode change mean? (Considering Vestal model, it means a HI criticality tasks meets its deadline)
    - What are the conditions for a mode change? Who decides it? (Again considering Vestal mode, in LO criticality mode when a HI criticality job needs more computation time that its $C_i(LO)$, it results in a mode change to HI criticality mode.)
    - Except DAL A, tasks belonging to which other criticality levels (DAL B-E or only DAL B or DAL B-C or DAL B-D) need to be guaranteed in all cases?
    - In MultiPartes, a partition could only have tasks belonging to a single criticality level. Considering ARINC 653, unused resources (TT slot) of one partition cannot be used by other partitions. Is the way to take in DREAMS related to scheduling?

9.5 References

[HS92] C. J. Hou and K. G. Shin, "Allocation of periodic task modules with precedence and deadline constraints in distributed real-time systems," in *Real-Time Systems Symposium, 1992*, 1992, pp. 146-155.

[HS94] C.-J. Hou and K. G. Shin, "Replication and allocation of task modules in distributed real-time systems," in *Fault-Tolerant Computing, 1994. FTCS-24. Digest of Papers., Twenty-Fourth International Symposium on*, 1994, pp. 26-35..

[PS93] D. T. Peng and K. G. Shin, "Optimal scheduling of cooperative tasks in a distributed system using an enumerative method," *Software Engineering, IEEE Transactions on,* vol. 19, pp. 253-267, 1993.

[AS99] T. F. Abdelzaher and K. G. Shin, "Combined task and message scheduling in distributed real-time systems," *Parallel and Distributed Systems, IEEE Transactions on,* vol. 10, pp. 1179-1191, 1999.

[PS89] D. T. Peng and K. G. Shin, "Static allocation of periodic tasks with precedence constraints in distributed real-time systems," in *Distributed Computing Systems, 1989., 9th International Conference on*, 1989, pp. 190-198.

[PSA97] D. T. Peng, K. G. Shin, and T. F. Abdelzaher, "Assignment and scheduling communicating periodic tasks in distributed real-time systems," *Software Engineering, IEEE Transactions on,* vol. 23, pp. 745-758, 1997.

[FDP00] S. Faucou, A. Deplanche, and J.-P. Beauvais, "Heuristic techniques for allocating and scheduling communicating periodic tasks in distributed real-time systems," in *Factory Communication Systems, 2000. Proceedings. 2000 IEEE International Workshop on*, 2000, pp. 257-265.

[Ram90] K. Ramamritham, "Allocation and scheduling of complex periodic tasks," in *Distributed Computing Systems, 1990. Proceedings., 10th International Conference on*, 1990, pp. 108-115.

[MD98] Y. Monnier, J.-P. Beauvais, and A.-M. Deplanche, "A genetic algorithm for scheduling tasks in a real-time distributed system," in *Euromicro Conference, 1998. Proceedings. 24th*, 1998, pp. 708-714 vol.2.

[GKM03] Girault, H. Kalla, M. Sighireanu, and Y. Sorel, "An algorithm for automatically obtaining distributed and fault-tolerant static schedules," in *Dependable Systems and Networks, 2003. Proceedings. 2003 International Conference on*, 2003, pp. 159-168.

[CSA95] C. Sheng-Tzong, H. Shyh-In, and A. K. Agrawala, "Schedulability-oriented replication of periodic tasks in distributed real-time systems," in *Distributed Computing Systems, 1995., Proceedings of the 15th International Conference on*, 1995, pp. 196-203.

[FK90] G. Fohler and C. Koza, "Heuristic Scheduling for Distributed Hard Real-Time Systems," 1990.

[VLH91] J. P. C. Verhoosel, E. J. Luit, D. K. Hammer, and E. Jansen, "A static scheduling algorithm for distributed hard real-time systems," *Real-Time Systems,* vol. 3, pp. 227-246, 1991/09/01 1991.

[FR97] G. Fohler and K. Ramamritham, "Static scheduling of pipelined periodic tasks in distributed real-time systems," in *Real-Time Systems, 1997. Proceedings., Ninth Euromicro Workshop on*, 1997, pp. 128-135.

[TF13] Theis and G. Fohler, "Mixed Criticality Scheduling in Time-Triggered Legacy Systems," in *1st Workshop on Mixed Criticality Systems, IEEE Real-Time Systems Symposium*, 2013

[Foh95] G. Fohler, "Joint scheduling of distributed complex periodic and hard aperiodic tasks in statically scheduled systems," in *Real-Time Systems Symposium, 1995. Proceedings., 16th IEEE*, 1995, pp. 152-161.

[Foh94] G. Fohler, "Flexibility in Statically Scheduled Real-Time Systems," Ph.D., TNF, Wien, Österreich, 1994.

[IF09] D. Isovic and G. Fohler, "Handling mixed sets of tasks in combined offline and online scheduled real-time systems," *Real-Time Syst.,* vol. 43, pp. 296-325, 2009.

[Foh93] G. Fohler, "Changing Operational Modes in the Context of Pre Run-Time Scheduling," *IEICE Transactions on Information and Systems,* vol. Special Issue on Responsive Computer System, 1993.

[JLM88] F. Jahanian, R. Lee, and A. K. Mok, "Semantics of Modechart in real time logic," presented at the Proceedings of the Twenty-First Annual Hawaii International Conference on Software Track, Kailua-Kona, Hawaii, USA, 1988.

[TF13] Theis and G. Fohler, "Mixed Criticality Scheduling in Time-Triggered Legacy Systems," in *1st Workshop on Mixed Criticality Systems, IEEE Real-Time Systems Symposium*, 2013.

[BF11] S. Baruah and G. Fohler, "Certification-Cognizant Time-Triggered Scheduling of Mixed-Criticality Systems," in *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*, 2011, pp. 3-12.

[Aud91] N. C. Audsley, "Optimal Priority Assignment and Feasibility of Static Priority Tasks With Arbitrary Start Times," The University of York1991.

[Aud93] N. C. Audsley, "Flexible Scheduling in Hard-Real-Time Systems," Ph.D., Department of Computer Science,, University of York, 1993.

[TFB13] J. Theis, G. Fohler, and S. Baruah, "Schedule Table Generation for Time-Triggered Mixed Criticality Systems," in *1st Workshop on Mixed Criticality Systems, IEEE Real-Time Systems Symposium*, 2013.

[Kor85] R. E. Korf, "Depth-first iterative-deepening: an optimal admissible tree search," *Artif. Intell.,* vol. 27, pp. 97-109, 1985.

[Foh12] G. Fohler, "Predictably Flexible Real-Time Scheduling," in *Advances in Real-Time Systems*, S. Chakraborty and J. Eberspächer, Eds., ed: Springer Berlin Heidelberg, 2012, pp. 207-221.

[DRS89] A Damm, J Reisinger, W Schwabl, and H Kopetz. The real-time operating system of MARS. SIGOPS Oper. Syst. Rev., 23:141–157, 1989.

10 Offline and Incremental Network Scheduling

10.1 Network Message Scheduling

Fully scheduled networks guarantee the deterministic delivery of periodic messages at predefined points in time, enabling data communication with fixed latency and bounded jitter. Other types of traffic may be enabled without interfering with the guaranteed times of scheduled messages (e.g. best-effort traffic). In order to follow a coordinated conflict-free scheme a global schedule including data forwarding needs to be enforced on each of the participating devices. An example of a scheduled network can be found in Figure 10-1.

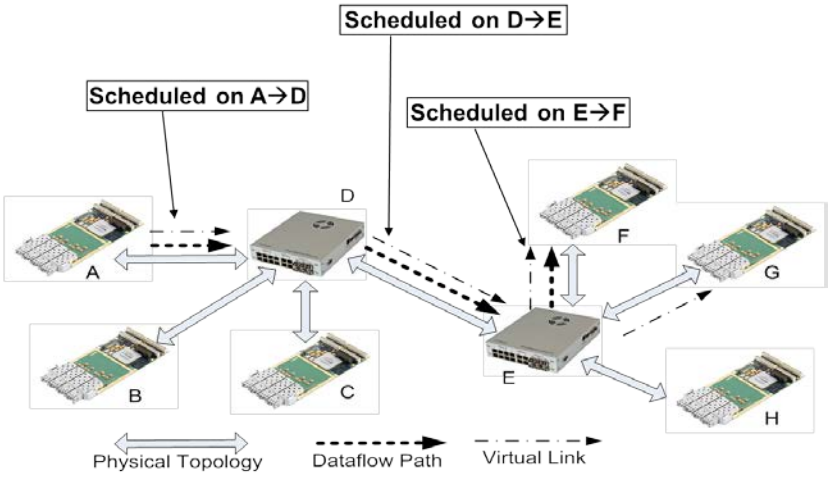

Figure 10-1 Example of a scheduled network.

Each device in the network must be aware of the incoming and outgoing time-windows of each end-to-end communication in which they participate. In the particular case of time-triggered networks, this scheme implies knowledge regarding the arrival of frames at an incoming port or outgoing ports for end systems, as well as the due time of forwarding at one or several outgoing ports in the case of switches. The timeliness of this operation must be precise, within the precision allowed by a global time synchronization service. Without this crucial service, the collision-free coordination between time-triggered network devices would not be possible. Figure 10-2 depicts an example of an end-to-end communication between two nodes (node A and node C) following a time-triggered scheme.

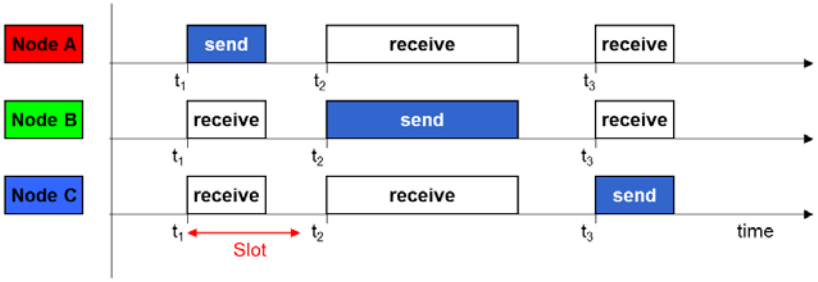

Figure 10-2 Example of Time-Triggered communication scheme.

10.2 Message Scheduling Constraints

Scheduling a time-triggered network is a known problem from the NP-complete complexity class. Hence, it has exponential complexity in terms of the size of the network and the number of messages that have to be scheduled. For this reason, offline network scheduling is still an area of research through different approaches.

10.2.1 Generic Constraints

A network schedule (tt-network-schedule) must satisfy a number of constraints in order to be feasible. An example of such constraints can be found in [Ste10]. For example, the scheduling window of any frame instance scheduled on a network link must be positioned within the period instance. Another essential constraint is that there is no overlap between any two scheduling windows associated with two frame instances scheduled on the same network link.
We can say that finding a feasible schedule for network messages reduces to finding a solution to a set of logical constraints.

10.2.1.1 Frame constraints

One essential constraint, mentioned above, is that any frame instance scheduled on a network link has a scheduling window positioned inside a period instance of the message, i.e., the frame instance offset is between the start of the period instance and the end of the period instance minus the length of the frame instance on the respective link.

10.2.1.2 Link constraints

Another important constraint for the correctness of a time-triggered network schedule is that no two scheduling windows on the same link overlap in time. This is similar as in CPU scheduling, in which no two tasks running on the same CPU may execute at the same time.

10.2.1.3 Communication constraints

Communication constraints deal with the correctness of the sequence of frame instances on different network links. The scheduling windows of frame instances of sequential links in the communication path has to follow the correct order, i.e., the end of the scheduling window on one link has to precede the start of the scheduling window on the next link.

10.2.1.4 End-to-end latency constraints

The difference between the arrival of a frame on the last link and the sending of the frame on the first link of the communication path has to be less than a user-defined maximum, also called the end-to-end latency. Here we differentiate two cases, namely, the end-to-end latency is less than or equal to the period length and the end-to-end latency is larger than the period length. Each case needs to be addressed in different logical constraints.

10.2.1.5 Resource constraints

Additional constraints may be needed in order to guarantee feasibility in terms of resource utilization. An example of such constraints is the utilization of a maximum number of frame buffers allocated in the internal memory of switch devices. In this case, we can take advantage of the properties of the time-triggered paradigm and define the memory constraints as the time a frame can remain in the internal buffers of a switch on the sending node. Usually, in TTEthernet this time is restricted to one message period.

10.2.2 Incremental Scheduling Constraints

We envision three different options to specify incremental constraints. The most straightforward one is to maintain the existing schedule *as-is* and try to fit the new messages in the remaining time slots of the schedule. This method is easy to implement and, due to its low computational complexity, might even be suitable to run on embedded nodes.

The second option consists of freezing the schedule on the end-nodes yet allowing for relocation of intermediate hops. Through this, more freedom is given to the scheduler to find a placement for the

new virtual links and their frame instances, yet maintaining the end-to-end properties of the already scheduled communication.

Finally, we envision a method where all existing frame instances on any link can be relocated and replaced anywhere in the schedule to accommodate new frames. This method guarantees the maximum flexibility towards new frame instances but will require a more computationally complex algorithm (either one which is heuristic-driven or one that uses some ILP or SMT solver). The trade-offs of one method over another need to be analyzed based on the system and communication requirements.

10.3 Message Scheduling Strategies

Several approaches for the elaboration of distributed time-triggered can be found in literature. For example, [Foh94] studies methods for static schedule generation of communication under the TDMA or Time-Triggered Protocol (TTP) paradigm. The authors use the iterative deepening algorithm with a heuristic function to schedule periodic tasks in a non-optimal approach. Follow-up work [IF09], extends the above method to include aperiodic tasks by combining the offline approach with a dynamic online mechanism.

Studying the scheduling problem as a constrained optimization formulation has been a hot topic in recent years. An example is [AS99], where an optimal task schedule for communicating tasks is generated using the branch-and-bound algorithm under the optimization criteria of minimizing the maximum task lateness. A similar approach but with a different optimization criteria is presented in [PSA97]. In [Ste10] the author evaluated the generation of offline schedules for time-triggered multi-hop networks by means of the state-the-art SMT solver YICES [SRI14]. The author formally describes the network communication as well as the communication constraints in terms of contention, end-to-end latency, and hop sequence dependencies. Furthermore, additional constraints with respect to resource utilization and multi-path dispatching instances are considered. Based on a given network topology and a set of communication requirements (virtual links), the author provides an evaluation of the YICES SMT solver, which proves to be suitable even for large networks like those typical of industrial use-cases. The evaluation presented in the paper shows that for significantly large networks (i.e. some 1000 virtual links) the SMT solver is able to find a schedule in about 30 min. In [Ste11] the author extends the previous work to introduce blank intervals within the schedule, which are used to accommodate rate-constrained communication traffic. This is done by means of a mixed pre- and post- processing of the schedule generated by the SMT-solver. While the final schedule maintains the schedulability constraints for TT-traffic, this approach enforces a certain ``porosity'' in the schedule, preventing all TT messages to be scheduled ``back-to-back'' and hence inducing starvation of RC messages.

In [ZGSC14] the authors approach a combined task and network schedule formulated as a Mixed Integer Programming (MIP) model, which is solved with different optimization criteria. In their work, the authors assume the end-systems to follow a non-preemptive time-triggered scheme, hence reducing significantly the model complexity for which a combined --tasks and network-- schedule is searched.

10.4 Network Scheduling for TTEthernet

TTEthernet [Ste08] (SAE AS6802) enables the coexistence of time-triggered, safety-critical communication together with non-real time traffic over standard IEEE 802.3 Ethernet. The main application domain of TTEthernet is in aerospace and industrial domains which require high-integrity and safe-critical real-time functionality while also permitting non-critical traffic without latency and throughput constraints. TTEthernet is a scalable, open real-time Ethernet platform used for safety-related applications primarily in transportation industries and industrial automation. It is compatible to IEEE 802.3 Ethernet and integrates transparently with Ethernet network components.

The three classes of communication in TTEthernet are time-triggered, rate constrained, and best effort. Time-triggered communication follows a static schedule where each message is transmitted and received between switches and end-systems at precise and predetermined instants in time.

In order to enable a network-wide time-triggered schedule without collisions, each participating node must have a common notion of time. This is achieved through a network-wide fault tolerant time synchronization protocol that has sub-microsecond precision [SD11, Kop97]. Rate constrained messages have a maximum bandwidth reservation that they can use. Best-effort messages are defined as regular Ethernet messages which are sent whenever there is time in the time-triggered schedule, i.e., whenever TT-messages are not scheduled.

10.4.1 The TTE-Toolchain

The generation of tt-network-schedules and configuration artifacts for (off-chip) TTEthernet nodes is carried out by the TTE-Toolchain. Figure 10-3 and Figure 10-4 show respectively the tool workflow and the related dataflow on a typical toolchain run.

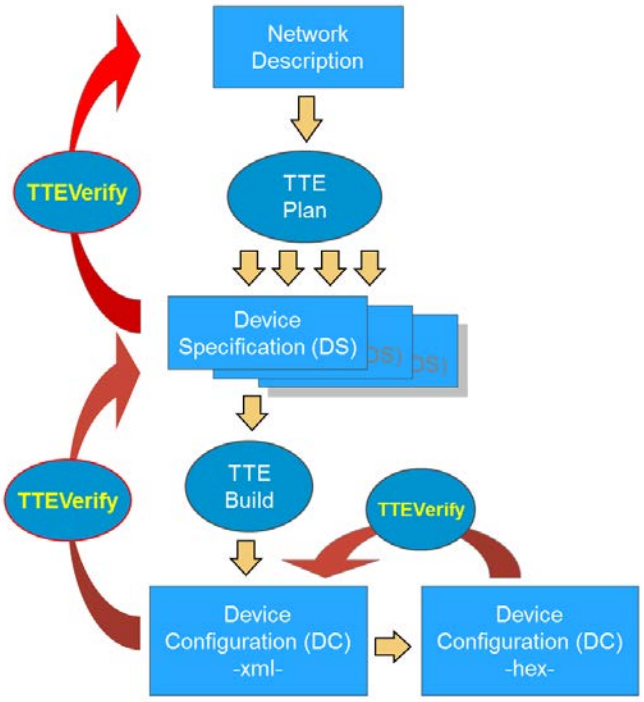

Figure 10-3 TTE-Toolchain workflow.

10.4.2 Constraints to TTE-Plan

TTE-Plan is used to generate network configurations for TTEthernet systems. It helps the user to model the topology of a TTEthernet network, and to generate a communication schedule for that network (i.e. tt-network-schedule).

10.4.2.1 Overview

TTE-Plan is a command-line tool that takes a network description XML file as input and generates a network configuration, which consists of a main *<.network_config>* file and additional files referenced by this file. In the network description file, the user configures aspects of the network such as topology, virtual links, and synchronization parameters. The network configuration file can be used to generate device configuration *HEX* files for each device in the network using TTE-Build, and these file s can be downloaded to the switches using TTE-Load, and to the end systems using the

application. When the network is in operation, the data traffic can be viewed using either an oscilloscope or TTE-View, a *Wireshark* distribution with a dissector plug-in for TTEthernet frames.

An overview of this tool chain showing input and output files is shown in Figure 10-4.

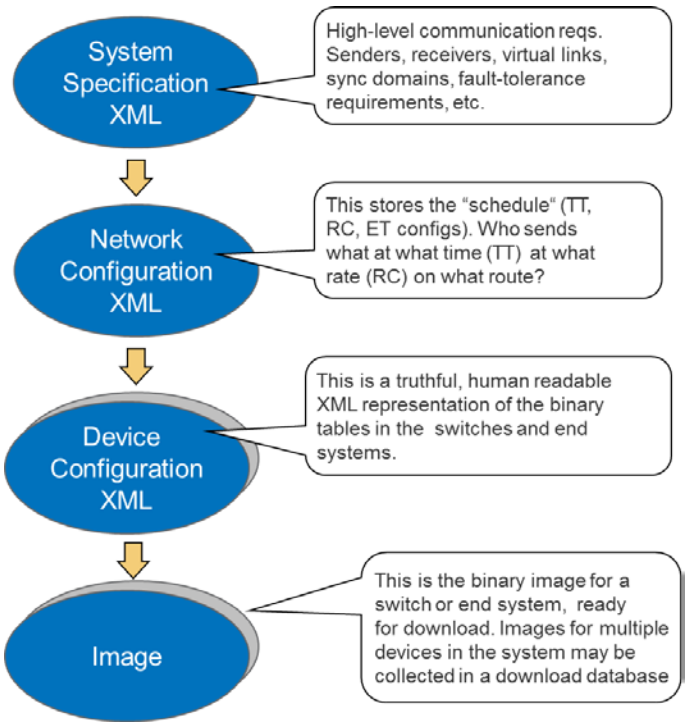

Figure 10-4 TTE-Toolchain dataflow.

10.4.2.2 Configuring a TTEthernet Network

TTE-Plan expects a network description XML file corresponding to the network topology and preferences. This file is used as input to TTE-Plan so it can generate the corresponding network configuration files, which constitute the input to TTE-Build and the following tools.

A complete network description contains several sections specifying the necessary TTEthernet preferences. Namely,

- timing parameters, including redundancy, global speed, and network periods;
- network synchronization configuration;
- network topology, specifying switches, end systems and their physical connections;
- time-triggered and rate-constrained virtual links, as well as best-effort traffic;
- frame buffers;
- specific constraints and scheduling guidance parameters.

10.5 Challenges/Shortcomings w.r.t. MCS

The synthesis of network-schedules is an NP complete problem and the computation time scales drastically for medium to large networks, which makes it untreatable even for offline methods. Methods based on heuristics provide reasonable run-time at the expenses of losing completeness (i.e. feasible schedules may be missed by the heuristic search). Other approaches may reduce the complexity by partitioning the problem to treat into several smaller problems. The trade-off between run-time and complexity is a clear challenge that needs to be evaluated.

In addition to the inherent complexity of finding feasible schedules, optimization metrics add up to the run-time and overall complexity. Simple or multi-objective optimization based one quality metrics provide guidance to the scheduler towards preferred allocation of transmission windows

improving selected properties (e.g. minimize memory utilization, minimize end-to-end latency, etc…).

In Section 10.2.2 we enumerate a number of possibilities to construct incremental schedules and their implications with respect to run-time and the already existing schedule. The pros and cons of each method and their suitability to particular use cases need to be explored in detail.

An additional remark remains with respect to the combination of dependent network and task schedules. In fact, producing and consuming messages transmitted through the network is performed by tasks running on the end-system nodes in the network. The end-to-end latency requirements thus extend to the task layer governing the complete communication path from the production of a message to its consumption inside a time-triggered task. The composition of the two scheduling domains is a non-trivial extension of the network-only incremental scheduling approach.

10.6 References

[PSA97] Peng, D.-T., Shin, K., and Abdelzaher, T. Assignment and scheduling communicating periodic tasks in distributed real-time systems. IEEE Trans. Softw. Eng. 23, 12 (1997), 745-758.

[AS99] Abdelzaher, T. F., and Shin, K. G. Combined task and message scheduling in distributed real-time systems. IEEE Trans. Parallel Distrib. Syst. 10, 11 (1999), 1179-1191.

[IF09] Isovic, D., and Fohler, G. Handling mixed sets of tasks in combined offline and online scheduled real-time systems. Real-Time Syst. 43, 3 (2009), 296-325.

[Foh94] Fohler, G. Flexibility in Statically Scheduled Real-Time Systems. PhD thesis, TNF, April 1994.

[Ste08] W. Steiner, “TTEthernet specification,” TTA Group, 2008. [Online]. Available: http://www.ttagroup.org

[SD11] W. Steiner and B. Dutertre, “Automated formal verification of the TTEthernet synchronization quality,” in NASA Formal Methods, ser. Lecture Notes in Computer Science. Springer, 2011, vol. 6617.

[SRI14] SRI. The Yices SMT Solver. Computer Science Laboratory { SRI Interna-tional, http://yices.csl.sri.com/, accessed 27-Mar-2014].

[Ste10] Steiner, W. An evaluation of smt-based schedule synthesis for time-triggered multi-hop networks. In RTSS (2010), pp. 375-384.

[Ste11] Steiner, W. Synthesis of static communication schedules for mixed-criticality systems. In 1st IEEE Workshop on Architectures and Applications for Mixed-Criticality Systems (AMICS 2011) (2011), IEEE Computer Society.

[ZGSC14] Zhang, L., Goswami, D., Schneider, R., and Chakraborty, S. Task-and network-level schedule co-synthesis of ethernet-based time-triggered systems. In Design Automation Conference (ASP-DAC), 2014 19th Asia and South Pacic (Jan 2014), pp. 119-124.

[Kop97] Kopetz, H. Real-Time Systems: Design Principles for Distributed Embedded Applications. Kluwer Academic Publishers, 1997

11 Optimization Techniques for Architectural Exploration

11.1 MOEA-based Architectural Exploration

Fault-tolerance techniques applied at the system-level (e.g., active redundancy) are a promising way to enhance the system reliability for safety-critical applications with moderate overhead. However,

the efficient application of active redundancy techniques during the design of real-time systems requires the joint solution of two challenging problems, namely the computation of an optimal redundancy configuration, and the deployment (i.e., mapping and scheduling) of a fault-tolerant variant of an application to the platform under reliability, real-time and possibly further constraints.

In the following, a MOEA-based design-space exploration (DSE) approach to the problem of synthesizing fault-tolerant embedded systems will be summarized [HRB12, HBR14]. As pointed out in Section 0, the DREAMS development process distinguishes the phases: modeling of product-lines, variability binding, offline adaptation strategies, and finally the generation of implementation artifacts. The approach summarized in this section contributes to the offline adaptation step (blue circle in Figure 2-1). Figure 11-1 provides a more detailed overview of the approach: First, a model of the application, and a model of the execution platform is either created by the system designer, or obtained by the preceding variability binding step. Additionally, extra-functional properties such as reliability of processors and temporal properties, e.g. the worst-case execution times of application tasks are specified. Now, the DSE process uses the system model as input and aims at finding an optimal deployment of the application under user-specified design constraints (e.g., end-to-end latency and reliability).

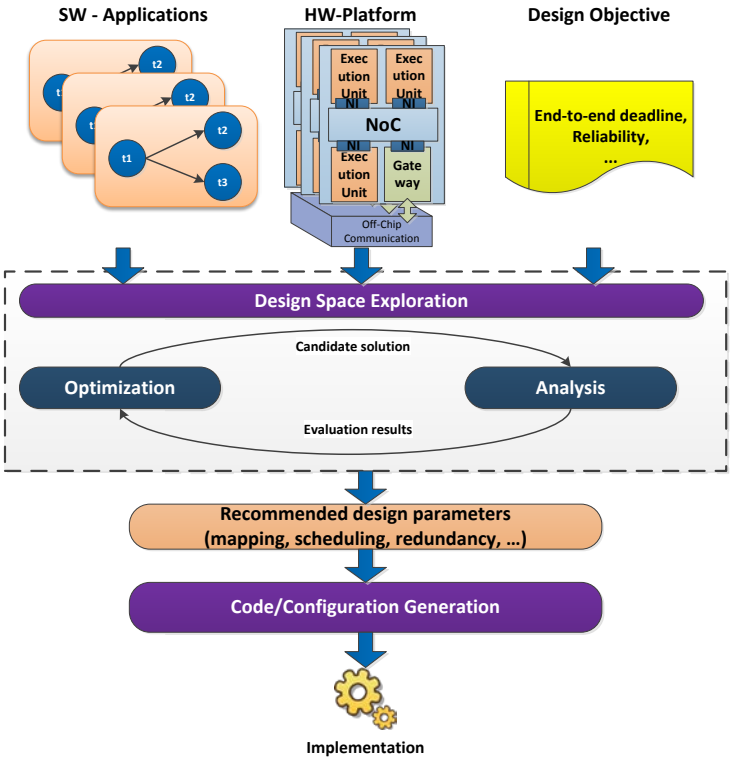

Figure 11-1 MOEA-based offline adaptation strategy

11.1.1 Optimization Procedure

As pointed out above, a multi-objective evolutionary algorithm (MOEA) is used as a generic optimization engine. [VL00] survey a number of algorithms with different optimization and implementation characteristics, including the well-known Strength Pareto Evolutionary Algorithm (SPEA) [ZT99]. The review of [ZQL11] provides an update that discusses more recent algorithms. There exist a number of software libraries providing reusable implementations of MOEA algorithms. For instance, the PISA toolset defines an interface abstracting the different implementations [BLT03]. The implementation of the optimization approach summarized in this section is based on the Opt4J library [LGR11] which is a Java-based framework providing several MOEA, including SPEA2 [ZLT02], an improved version of the aforementioned SPEA approach.

In MOEA, candidate solutions are encoded in dedicated data structure called chromosome. MOEA is an iterative optimization procedure where an optimizer maintains a set of solutions for the particular problem – the so-called population. In each of the iterations, the optimizer selects a subset of solutions from the population, which are used to derive new candidate solutions. Hence, the selected solutions from the population are referred to as parent solutions, whereas the newly created solutions are called offspring. The creation of offspring from the selected parent solution is performed using crossover and/or mutation operators. While cross-over operators combine one or more of the parent solutions to obtain a new candidate solution, mutation operators introduce artificial disturbances into a parent solution, in order to explore otherwise unreachable regions of the solution space.

In MOEA, both generic and problem-specific operators can be distinguished. While the former ones have the advantage that they can universally be applied to different optimization problems, they can result in semantically invalid chromosomes. Problem-specific operators avoid this problem, and potentially use heuristics to obtain promising offspring by exploiting specific properties of the particular optimization task. The approach summarized in this section relies on a number of these custom operators, which are presented in the next section.

During the optimization, new solutions are evaluated by a user-defined measure, so-called fitness functions, such that high-quality solutions will replace low quality ones in the population. This optimization process terminates when either a candidate of sufficient quality is found (subject to the specified fitness functions) or a maximum number of iterations is reached.

As pointed out above, the computation of optimal fault-tolerant system configurations requires the joint optimization with respect to different fitness functions. The approach summarized in this section considers two time-triggered scheduling policies that can be used to provide real-time guarantees for the resulting system design. On the one hand, the approach supports a hierarchical combination of Time-Triggered and Static Priority scheduling (TT-SP) [IPE05]. On the other hand, Time-Triggered scheduling with Flexible Slack (TT-FS) [HBR11] is considered.

11.1.2 Schedule Reconstruction

It is necessary to consider information about all time slots, such as start/finish times and task assignments, in order to optimize with respect to each of the above scheduling policies. Hence, a direct encoding of such schedules would result into a very large chromosome, not only growing with the number of tasks, but also in the length of the schedule.

To overcome the problem pointed out above, the approach summarized in this section employs a two-step encoding that has initially been introduced in [LGH07]. Here, the basic idea is to minimize the size of the chromosome by only storing the task mapping and the redundancy configuration in it. If the full schedule is required in the fitness evaluation of a candidate solution (e.g., in particular for the reliability analysis), a scheduler is used to rebuild the schedule from the information contained in the chromosome.

As illustrated in Figure 11-2, the chromosome contains one gene per task encoding a unique (integer) index for the task, as well as the (integer) indices of the processing elements where the task is mapped to. Here, mappings of a particular task to different processing elements represent spatial replication (hardware redundancy, see Section 6.3.2.1), whereas multiple mappings of the same task represent re-execution slots (i.e., software redundancy).

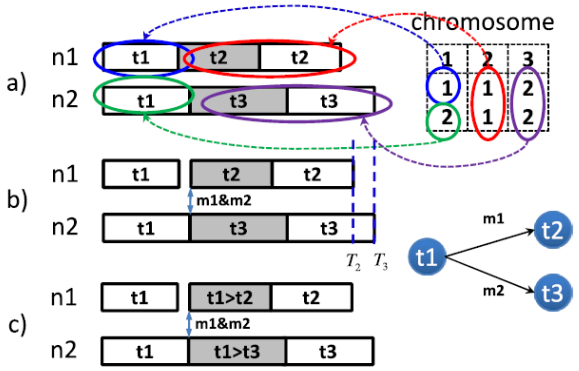

Figure 11-2 Encoding of mapping and schedule reconstruction

In order to improve the performance of the exploration process, a number of custom crossover and mutation operators have been defined. All operations do not have further effects on the remainder of the chromosome.

- The *task implementation crossover* operator randomly selects a task and swaps its entire implementation between two chromosomes, including its replica configuration (amount of spatial/temporal redundancy and mapping).
- The *task mapping crossover* operator is applied to the mapping of an individual replica of tasks as follows: Given two chromosomes, at first the set of tasks to be manipulated is randomly selected. Then, for each task, the mapping of one randomly selected replica is swapped between the chromosomes (i.e., the i^{th} replica of both tasks is modified).
- The *increment (decrement) redundancy mutation* operator inserts (removes) a new mapping entry for a randomly selected task. The insertion of the new mapping x to task t might result in a slack slot, if the chromosome already contains a mapping of t to x, or a spatial replication, if t has not been mapped to x. The decrement operation ensures that for each task at least one mapping remains.

The problem of reconstructing a schedule from a chromosome corresponds to the problem of task scheduling with a known mapping and redundancy configuration. The reliability analysis discussed in Section 11.1.4.1 can be combined with different schedulers. Here, the actual selection, and therefore the properties of the resulting schedules (e.g., generic vs. strict schedules), will have an influence on the efficiency of the analysis.

Before summarizing schedulers for the policies introduced above, the underlying system model will be briefly introduced.

- An application A is comprised of a set of independent application subsystems (jobs) that is defined using a DAG modeling the data dependencies between tasks. The tasks of A are assumed to be periodically activated and to share the same period. If needed, the task graph needs to be transformed into a hyper-period-based graph that satisfies this condition (LCM of all periods).
- Furthermore, the approach assumes the availability of time-triggered communication between the processing elements of the platform. The corresponding message schedule can be described by a set of message slots, where each slot models the start and end time of each message, as well as its sender and receiver.

The presented approach depends on the selected scheduling policy. For TT-SP, a scheduling slot of length equal to its execution time is instantiated for each mapping entry of a task. Then, the set of slots is scheduled using a list scheduler. Here, a heuristic is used to determine the task priority (by applying an EDF-scheme at the job-level and by considering the data-dependencies for tasks within the same job). After a priority has been assigned to all tasks, messages are scheduled based on the transparent recovery approach [KHM03]. This implies that messages should only be scheduled after

the last possible re-execution of a task in order to allow faults to be masked by the replica. In the last step, re-execution slots are selected for slack sharing in a greedy fashion. Here, a slot is shared with all tasks that become ready before the slot's start time and whose execution time fits into the slot size.

For TT-FS, both the encoding in the chromosome and the scheduling approach are altered as follows: In the chromosome, the task ID 0 is reserved for slack slots (IDs > 0 denote regular slots). Then, the same list-based scheduler as described above is used to obtain an initial schedule. Slack slots are initially placed right after the regular slots of the same tasks. In a second step, a greedy slack sharing strategy is applied. Here, the schedule is separated into segments, where segments are separated by one or several consecutive slack slots. Then, each slack slot is allowed to be shared by all tasks in the segment just before itself. The size of slack slots is set to the largest execution time of all tasks sharing the slot. Also, the placement of messages must consider the fact that normal slots might be delayed due to out-of-order execution of slack slots. Hence, the message scheduling has to take into account the worst-case scenario in order to ensure that faults on one processor are actually masked to other processors even if the task is delayed.

11.1.3 Objective Specification

In addition to a model of the system onto which the MOEA-based exploration approach summarized in the last section should be applied, the specification of the optimization objective is the second input to be provided by a user of the approach. In the following, a meta-model that can be used to specify the optimization goals will be briefly described. On the one hand, the object specification meta-model allows the configuration of general parameters of the MOEA engine:

- *Scheduling model*: Selection of the execution paradigm of the application (see above)
- *Optimizer*: Configuration of parameters of the MOEA engine:
    - Alpha: the size of population
    - Generations: number of iterations of the EA
    - Spea2 Tournament: the tournament value in the SPEA2 algorithm

On the other hand, the meta-model provides means for the specification of optimization objectives and constraints.

11.1.3.1 Reliability Objectives

The reliability meta-model consists of two main parts: The first part is used to annotate system-level reliability goals. The metric of reliability depends on the selected fault model. For the consideration of permanent faults, the most commonly used metric is the *Mean-Time-To-Failure* (MTTF). If transient faults (or soft errors) are considered (see Section 6.2), the system level reliability is typically described by the failure probability or *Failure-In-Time* (FIT). In the following, the specializations of the class *ReliabilityGoal*, the root class for the specification of the system-level reliability goals, will be described:

- *Failure in time*: Goal specification to maximize the reliability of the selected application, considering Failure-In-Time (FIT) of both detectable and undetectable faults. It contains the following configuration parameters:
    - *MaxReplication*: maximum number of replications for each task.
    - *MinReplication*: minimum number of replications for each task.
    - *Mode*: fail-safe or fail-operational. For fail-safe systems, faults that are only detected but not corrected are considered as harm-free, since the system can execute a safe shut-down. In contrast, both detectable and undetectable faults are considered as failure in fail-operational systems.
    - *Sink* Component: the sink task used to identify the job for which the reliability is to be optimized (an application may contain multiple jobs).
- *Reliability objective DUF fit*: A reliability objective that only considers Detectable Unrecoverable Faults (DUFs). The configuration parameters are the same as for the *Failure in time* objective, except that the *Mode* selection is not necessary.

- *Reliability objective SDC fit*: A reliability objective that only considers Silent Data Corruption (SDC). The configuration parameters are the same as above, except that the *Mode* selection is not necessary.
- *SystemSoftErrorRateGoal*: This class is used to specify the SER goal of the entire system.
- *MTTF*: This class is used to annotate the system-level reliability requirement concerning permanent faults.

The second part of the meta-model is used to describe the component-level reliability. In the following, specializations of the class *ReliabilityAnnotation*, the root class of all classes for annotating component-level reliability information, will be described. For the consideration of transient faults, such information may be the component-level *soft error rate* (SER). With respect to permanent faults, reliability is typically described using reliability functions.

- *ComponentSoftErrorRate*: This class is used to annotate SER of a certain hardware component. The standard assumption in literature is that all hardware components suffer from transient faults according to a Poisson distribution with a constant failure rate.
- *TaskSoftErrorRate*: This class is used to annotate the failure probability of tasks. According to the classical reliability model, the failure probability of a task T is computed as $\Pr(T) = 1 - e^{-\lambda t}$, where λ is the SER of the hardware resource and T is executed on and t is the execution time of T.

11.1.3.2 Temporal Constraints

As temporal constraint, a *deadline objective* can be specified as optimization goal which demands the minimization of the end-to-end latency between selected tasks. This constraint specification object bundles sub-constraints in order to define a number of end-to-end deadlines for joint optimization. The fitness value of the deadline objective is evaluated as follows: For a given design, every deadline specification is evaluated. If its particular deadline is met, it has no contribution to the fitness value. Otherwise, the gap between the actual end-to-end latency and the deadline is used as penalty. A fitness value of 0 for the deadline objective means that all end-to-end deadlines have been met.

- *Deadline*: end to end deadline value
- *Unit*: time unit
- *Source*: source task
- *Sink*: sink task

11.1.4 Fitness Evaluation

In this section, different methods that can be used to rate the fitness of the current population in the optimization process will be introduced.

11.1.4.1 Reliability Analysis

As mentioned above, system reliability is one of the optimization criteria in the architectural exploration process summarized in this chapter. Here, the goal is to explore the application of fault-tolerance techniques onto a given (non-fault-tolerant) system specification in order to guarantee a required level of reliability. Here, the reliability analysis typically aggregates its estimation from individual components up to the system-level. A reliability analysis that is suitable for the exploration process and the scheduling models introduced is presented in the context of the recovery strategies in Section 6.3.

11.1.4.2 Application-specific Constraints

The analysis summarized in the last section can be used to jointly explore different design alternatives that affect the reliability and certain temporal properties of the system under design. In addition to that, the approach can also be used to consider additional constraints.

The two-step encoding pointed out above, where only the mapping of tasks to processing elements is directly encoded in the chromosome, not only increases the efficiency of the exploration approach, but also enables to encode additional *application-specific constraints* during the construction of the chromosome. This type of constraint is used to enforce or prevent the mapping of tasks to specific processing elements in order to satisfy application-specific requirements. For instance, a task that requires access to I/O signals must be mapped to a processor core that can access the corresponding peripheral. Similarly, separation constraints can be expressed that enforce the spatial separation of critical tasks.

In order to allow for the specification of application-specific constraints, the objective model introduced in Section 11.1.3 is extended as follows:

- *Fixed Deployment Constraint*: fixed mapping of selected task to a certain processing element in the platform model
  - *Task*: task for which a fixed deployment is desired
  - *Replication*: desired number of replications for the selected task
  - *Resource*: processing element to which the task is to be deployed
- *Exclude Deployment Constraint*: counterpart of the above, i.e. it prevents a task from being mapped to particular processing elements.
  - *Task*: task for which the deployment exclusion constraint is to be specified
  - *Resources*: list of the processing element to which the task should not be mapped to.

11.1.4.3 Energy Consumption

The cost functions introduced so far rate the different variations of the original system design based on their estimated reliability, temporal conditions (such as end-to-end latencies) and additional task separation or fixed mapping constraints. However, an optimization that is exclusively based on these measures has the tendency to prefer solutions that require a significant use of resources (e.g., due to excessive use of hardware redundancy). In order to balance the measures used to satisfy the above requirements with the cost of design, additional measures that consider the cost of the candidate solutions such are required.

The current implementation of the exploration relies on a linear energy model that considers the resource consumption of tasks (CPU time) in order to provide a very rough estimation of their energy consumption. While this over-simplified model abstracts many details, it is an effective initial approach to the above problem. In order to provide a more realistic estimation, more fine-grained energy models such as the one suggested by [ZA09] should be considered that also takes into account the impact of energy management mechanisms on the system reliability.

11.2 Product-line Testing Technology

Product Lines (or product families) are groups of products whose commonalities were anticipated and leveraged in order to reduce time to market, increase productivity and reduce production costs. An intuitive example of product lines is the car production domain, where all cars share a common structure (including wheels, engine, windshield, etc.) but where such parts can vary (electric/gas engine, summer/winter tires, etc.) from one car another. Product line engineering (PLE) is thus concerned with the early identification of variability (resp. commonality) in a product family and the construction of the platform that supports the development of any related product (so called "derived" products).

Software Product Lines (SPL) are product lines where products are software systems. Among others, example of large scale SPLs includes the Linux Kernel, which is made of a set of carefully selected modules, or the Eclipse IDE that can be tailored to specific tasks (e.g., C/C++ or Java deployment, testing, modelling) by adding (resp. removing) some of the underlying plugins. In the DREAMS project, we intend to use SPL to capture the variability of both the DREAMS application and the related execution platform. Figure 11-3 below illustrates some of the variability that could be

relevant in the DREAMS setting: various types of CPU, various types of hypervisors, various types networking technologies, etc. As shown below, SPL focus on capturing the set of discrete choices that drive the design of specific products. Continuous variability is though not necessarily incompatible with the idea of SPL, but requires different techniques to be properly leveraged.

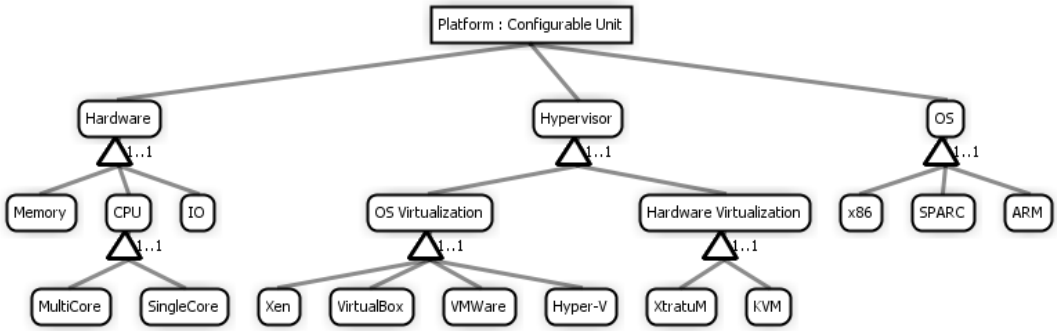

Figure 11-3 A simplified BVR model describing some variability of hardware execution platforms

Feature diagrams capture the variability of software systems using a tree to structure the variation points and their associated variants. For instance, in Figure 11-3, the hypervisor technology might either a bare-metal hypervisor which runs directly on the host hardware or a hosted hypervisor, which runs on top of the operating system of the host. In addition, logical constraints are used to filter out configurations (i.e., products) that do not make sense in the domain of interest (i.e., Eclipse IDE). In Figure 11-3 for instance, selecting Hyper-V implies an x86 underlying systems.

11.2.1 Modelling Product Lines with BVR (CVL)

BVR (Base Variability Resolution models) is a language built on the *Common Variability Language* (CVL) technology, but enhanced due to needs of the industrial partners of the VARIES project[6], in particular Autronica. BVR is built on CVL, but CVL is not a subset of BVR. In BVR we have removed some of the mechanisms of CVL that we are not using in our industrial demo cases that apply BVR. We have also made improvements to what CVL had originally. For the purpose of DREAMS we may just say that BVR is a continuation of the CVL language with associated tooling.

CVL is the language that is now a Revised Submission in the OMG [HW13] defining variability modelling and the means to generate product models. CVL is in the tradition of modelling variability as an orthogonal, separate model such as OVM [PB05] and the MoSiS CVL [MoSIS-D2.1.4] which formed one of the starting points of the OMG CVL. The principles of separate variability model and how to generate product models are depicted in Figure 11-4.

[6] http://www.varies.eu

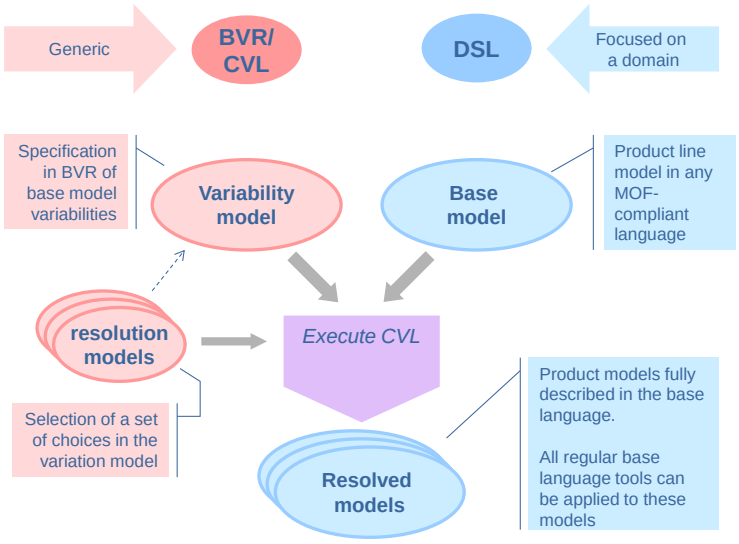

Figure 11-4 BVR/CVL principles

The CVL architecture is described in Figure 11-5. It consists of different inter-related models. The variability abstraction consists of a VSpec model supplemented with constraints, and a corresponding resolution model defining the product selections.

The variability realization contains the variation points representing the mapping between the variability abstraction and the base model such that the selected products can be automatically generated. The configurable units define a layer intended for module structuring and exchange. In this paper we have not gone into that layer.

The VSpec model is an evolution of the FODA [KC90] feature models, but the main purpose of CVL has been to provide a complete definition such that product models can be generated automatically from the VSpec model, the resolution model and the realization model.

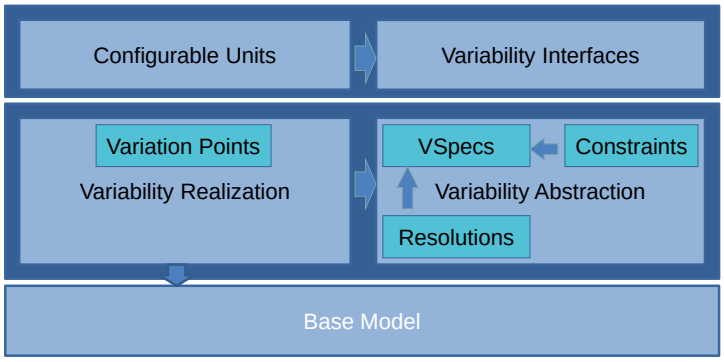

Figure 11-5 CVL/BVR language architecture

BVR is an evolution from CVL where some constructs have been removed for improved simplicity and some new constructs have been added for better and more suited expressiveness.

The main modelling vehicle for describing the choices of the product line is the VSpec tree (called feature models in other notations) as you can see in Figure 11-3. The notations for resolution model and realization models are tool specific.

11.2.2 Testing Software Product Lines

As any other software pieces, derived products must exhibit predefined quality of service and are subject to verification and validation (V&V) using testing tools among others. However, as the

number of possible products grows exponentially with the number of variation points, testing them separately is practically not feasible in real life settings. As shown by M. F. Johansen in [Jo13], product line testing therefore relies on three main approaches, namely: model based testing, regression testing and subset heuristics.

Martin Johansen developed in SINTEF the ICPL toolset, which foster testing software product lines. The main contributions of this tool are summarized in Figure 11-6. Testing SPL requires three main inputs:

- A software system and its implementation artefacts
- The feature model capture the inherent variability, and in turn the set of possible variants, which can be derived from the given software artefacts.
- A set of test cases used to validate products derived from the systems.

The first step consists in *sampling* the space of possible products, in order to cover the possible *t*-wise interactions between features (i.e., 1-wise ensures that all features are selected at least once, whereas 2-wise coverage ensures that each couple of feature is selected at least once). The resulting products can thus be automatically built by assembling existing software artefacts, and tested using the provided test cases.

11.2.3 Product Line Engineering to Build Self-Adaptive Systems

11.2.3.1 Self-Adaptive Systems

Software components are expected to maintain their quality of service (QoS) continuously under various execution conditions. One promising solution, so called self-adaptation, is to let the system adjust its internal configuration while its environment evolves so as it keeps delivering the expected QoS. The development of these self-adaptive systems is at the crossroad of several engineering discipline including Artificial Intelligence, Control Systems and Software Engineering among others.

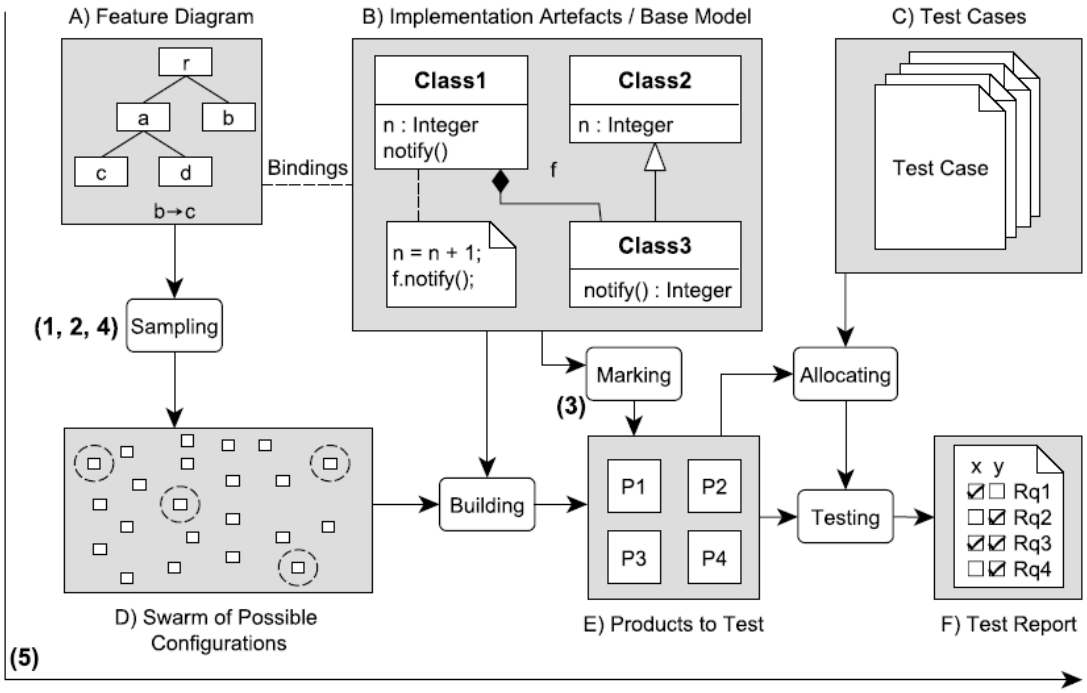

Figure 11-6 The ICPL tool to efficiently test product lines (borrowed from [Jo13])

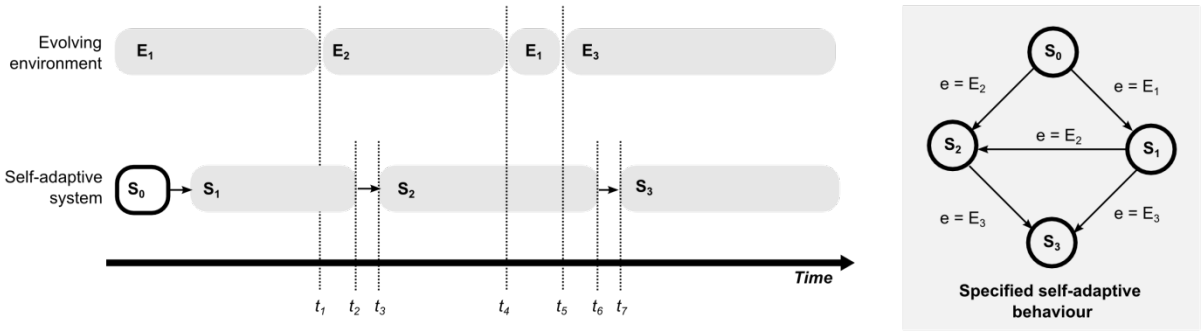

Figure 11-7 Principles of self-adaptation in software systems

Figure 11-7 illustrates self-adaptation on a simplified example. The system (at the bottom left) monitors its environment (the upper left part) and adjusts its internal state. While the environment transitions from E_1, E_2 and E_3, the system respond by transitioning from S_0, S_1, S_2, and S_3. A possible specification of the adaptive behaviour of the system is shown on the right-hand side, as an automaton capturing the transition function that decides which states fits which environment. In this setting, self-adaptive systems can be categorized in as:

- Static adaptation where the states, the environments, and the transition function are all pre calculated at design-time.
- Semi-dynamic adaptation where the states are preselected at runtime, but the transition function dynamically select the state that best fits the current environment.
- Dynamic adaptation where the neither the states nor the transition function is explicit at design- time. During its execution, the system dynamically generates new configurations that better fits its environments.

The more dynamic is the adaptation, the more difficult it becomes to guarantee that the system will behave properly. Dynamic software product lines (DSPL) are a means to define an envelope [PCD08, ACF09, CHZ09], in which the system is allowed to evolve (more or less freely). Using DSPL, self-adaptive systems are switching from products to products in order to deliver the expected QoS in the presence of a constraining environment. Ideally, under dynamic adaptation, the system should entirely synthesize the product that best fits its environment.

In most cases yet, the fully dynamic adaptation is not practical due to limitations in computing power, and one has to settle for either static adaptation or semi-dynamic adaptation. In these cases, it is of the utmost important to find the appropriate set of states (or products) which minimises the impact the environment has on the delivered QoS.

11.2.3.2 Interaction Coverage as a Heuristic to the State Selection Problem

In this setting, we investigate to which extent, the coverage of t-wise interactions provided by the ICPL tool can be a good heuristic to select a set of states relevant to build robust self-adaptive systems.

Our experimental setup is described as follows:

- A software product line ℓ, as the set P of software products p that can be legally derived from ℓ. In the DREAMS context, the product line reflects the DREAMS application running in on top of the DREAMS platform, or the possible configuration of the applications.
- A set E of environments called e that the system is expected to face. In the DREAMS context, environments capture external configurations of the DREAMS platform and or external factor which has an impact on the overall QoS delivered by the application.
- A set of sampling strategies σ that generate a subset of any given product lines (i.e., $\sigma: P \rightarrow 2^P$). ICPL provides several of such strategies that cover specific level of interactions

(1-wise, 2-wise and 3-wise). Alternatively, our experiment baseline is the random selection of a predefined number of products.

- A function called Δ, to measure the actual discrepancy between a given product p and its execution environment e , such as $\Delta: P \times E \rightarrow \mathbb{R}$. In DREAMS, such a function represents the evaluation of extra-functional properties such as memory consumption, response time, etc.

With the above definitions, the overall cost of a given sampling strategy σ is given by:

$$cost_{\ell}(\sigma) = \sum_{e \in E} \min_{p \in \sigma(\ell)} \Delta(p, e)$$

The use of the minimum operator reflects here our assumption of semi-dynamic adaptation, where the system is able to select the state that best fits its environment among a set of predefined ones (i.e., the state with lowest discrepancy).

The above experimental setup should help us evaluate various strategies to leverage SPL to design self-adaptive systems.

11.3 Challenges/Shortcomings w.r.t. MCS

In the following, the challenges of applying the approach summarized in Section 11 to MCS will be briefly discussed.

- The approach currently does not explicitly consider multiple criticality levels. In order to consider this information, an additional cost function could be integrated into the analysis that assigns a penalty in case tasks of different criticality levels are mapped to the same processing element (resulting in increased certification cost or demanding for the application of additional TSP mechanisms).
- The reliability analysis is based on a purely time-trigged execution model, which might not be adequate for the integration of low criticality tasks. However, the two-step encoding used in the exploration process provides the basis for the integration of additional offline scheduling approaches.
- While the slack sharing scheme provided by the fault-tolerance mechanisms configured by the exploration process provides a certain level of runtime adaptation, also the concept of mode changes could be adopted. Here, a set of configurations for different system states (e.g., due to degradation, or different system use cases) is pre-computed at design-time.

11.4 References

[BLT03] Bleuler, S., Laumanns, M., Thiele, L., and Zitzler, E. (2003). PISA – a platform and programming language independent interface for search algorithms. In Fonseca, C. M., Fleming, P. J., Zitzler, E., Thiele, L., and Deb, K., editors, *Evolutionary Multi-Criterion Optimization*, volume 2632 of *LNCS*, pages 494–508. Springer, Berlin Heidelberg.

[HBR14] Huang, J., Barner, S., Raabe, A., Buckl, C., and Knoll, A. (2014). A framework for reliability-aware embedded system design on multiprocessor platforms. *Microprocess. Microsyst.* In Press.

[HBR11] Huang, J., Blech, J. O., Raabe, A., Buckl, C., and Knoll, A. (2011). Analysis and optimization of fault-tolerant task scheduling on multiprocessor embedded systems. In *Proceedings of the Seventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '11)*, pages 247–256, New York, NY, USA. ACM.

[IPE05] Izosimov, V., Pop, P., Eles, P., and Peng, Z. (2005). Design optimization of time- and cost-constrained fault-tolerant distributed embedded systems. In *Proceedings of the Design, Automation and Test in Europe Conference (DATE '05)*, pages 864–869 Vol. 2.

[KHM03] Kandasamy, N., Hayes, J. P., and Murray, B. T. (2003). Transparent recovery from intermittent faults in time-triggered distributed systems. *IEEE Trans. Comput.*, 52(2):113–125.

[LGH07] Lukasiewycz, M., Glaß, M., Haubelt, C., and Teich, J. (2007). SAT-decoding in evolutionary algorithms for discrete constrained optimization problems. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 07)*, pages 935–942.

[LGR11] Lukasiewycz, M., Glaß, M., Reimann, F., and Teich, J. (2011). Opt4J: A modular framework for meta-heuristic optimization. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, pages 1723–1730, New York, NY, USA. ACM.

[VL00] Veldhuizen, D. A. V. and Lamont, G. B. (2000). Multiobjective evolutionary algorithms: Analyzing the state-of-the-art. *Evolutionary Computation*, 8(2):125–147.

[ZQL11] Zhou, A., Qu, B.-Y., Li, H., Zhao, S.-Z., Suganthan, P. N., and Zhang, Q. (2011). Multiobjective evolutionary algorithms: A survey of the state of the art. *Swarm and Evolutionary Computation*, 1(1):32–49.

[ZA09] Zhu, D. and Aydin, H. (2009). Reliability-aware energy management for periodic real-time tasks. *IEEE Trans. Comput.*, 58(10):1382–1397.

[ZLT02] Zitzler, E., Laumanns, M., and Thiele, L. (2002). SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. In Giannakoglou, K. et al., editors, *Evolutionary Methods for Design, Optimisation and Control with Application to Industrial Problems (EUROGEN '01)*, pages 95–100. International Center for Numerical Methods in Engineering (CIMNE).

[ZT99] Zitzler, E. and Thiele, L. (1999). Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE Trans. Evol. Comput.*, 3(4):257–271.

[Jo13] Martin Fagereng Johansen, *Testing Product Lines of Industrial Size: Advancements in Combinatorial Interaction Testing*, PhD thesis, University of Oslo, 2013.

[PCD08] Perrouin, G., Chauvel, F., DeAntoni, J., & Jézéquel, J. M. (2008). Modeling the variability space of self-adaptive applications. In *2nd Dynamic Software Product Lines Workshop (SPLC 2008, Volume 2)* (pp. 15-22).

[ACF09] Acher, M., Collet, P., Fleurey, F., Lahire, P., Moisan, S., & Rigault, J. P. (2009, October). Modeling context and dynamic adaptations with feature models. In *Proceedings of the 4th International Workshop Models@ run. time*.

[CHZ09] Carlos Cetina, Øystein Haugen, Xiaorui Zhang, Franck Fleurey, Vicente Pelechano. Strategies for Variability Transformation at Run-time. 13th International Software Product Lines Conference (SPLC). San Francisco, California. 2009.

[HW13] Haugen, O., A. Wasowski, et al. (2013). CVL: common variability language. Proceedings of the 17th International Software Product Line Conference. Tokyo, Japan, ACM: 277-277.

[KC90] Kang, K., S. Cohen, et al. (1990). Feature-Oriented Domain Analysis (FODA) Feasibility Study. Pittsburgh, PA, Software Engineering Institute, Carnegie Mellon University.

[MoSIS-D2.1.4] Ø. Haugen et al. D2.1.4: Consolidated CVL language and tool. Deliverable of the MoSiS project. ITEA IP06035. WP2 Deliverable 2.1.4.

[PB05] Pohl, K., G. Böckle, et al. (2005). Software Product Line Engineering. Foundations, Principles and Techniques, Springer.