Adaptability and reconfiguration of automotive embedded systems

Amel Belaggoun

THÈSE DE DOCTORAT DE
l'UNIVERSITÉ PIERRE ET MARIE CURIE

Spécialité

Informatique

École doctorale Informatique, Télécommunications et Électronique (Paris)

Présentée par

Amel BELAGGOUN

Pour obtenir le grade de

DOCTEUR de l'UNIVERSITÉ PIERRE ET MARIE CURIE

Sujet de la thèse :

Adaptability and Reconfiguration of Automotive Embedded Systems

soutenance prévue pour le 10 Octobre 2017

devant le jury composé de :

Marisol Garcia Valls (Universidad Carlos III de Madrid, SP)	Rapporteur
Christos Kloukinas(City University, UK)	Rapporteur
Nelly BENCOMO (Aston University, UK)	Examinateur
Robert Davis (York University, UK)	Examinateur
Laurent Pautet (Télécom ParisTech, FR)	Examinateur
Valerie ISSARNY(Inria,FR)	Directrice de thèse
Ansgar RADERMACHER(CEA LIST, FR)	Co-directeur de thèse

Acknowledgements

Valerie ISSARNY introduced me to the research world when I first visited MiMove team for an internship at INRIA-Paris. Her dedication, suggestions and guidance converted the lost student that arrived in 2012 into someone able to have his own ideas, express them and convert them into a PhD thesis. Most importantly, she supported my work with valuable suggestions and stimulating discussions that strongly influenced the content of the thesis. Thanks Valerie for your patience, for teaching me how an article should be written and presented.

I extend my gratitude to Ansgar RADERMACHER, my co-advisor, for his continuous feedback to my work. During my thesis years, he provided me with the nicest research environment, allowing me to participate in conferences and summer schools. This thesis is the result of all you have taught me. It has not only been an honor to work with you both, it was really a pleasure.

I sincerely thank the reviewers and the jury: Marisol Garcia-Valls and Christos KLOUKINAS for accepting reviewing this thesis and for all the comments they made; Nelly BENCOMO and Laurent PAUTET for accepting to be in my PhD thesis committee.

Especial thanks to. Davis Robert from Real-Time Systems Research Group at the University of York for his help and the interesting discussions, suggestions about my work and for giving me confidence to continue this work. My thanks also go to the CEO of Evidence srl Paolo Gai for his availability and his help on the implementation of my work in ERIKA Enterprise Real-time Operating System.

I would like also to thank my colleagues in CEA Tech-LISE research group for stimulating discussion and providing peer review of the work described in this thesis. In particular I would like to thank Sébastien Gérard and Frédérique Descreaux who contributed in various way to these efforts.

I want to thank these colleagues with whom I have shared not only the office and lunch breaks, but also amusing and unforgettable moments: Fatma Cheick, Sahar Guermazi, Asma Mehiaoui and Fadwa Rekik.

I owe unconditional thanks for the support of my family who had to encourage. Thank you for the trust you have given me, a big thank you for your supports during my studies. I would also like to thank my mother for her unfailing support, and thanks to whom

I could reach this stage. She always has been there for me, in all situations and I am infinitely grateful to her.

My heartfelt thanks goes to my dear Husband for his patience and daily unwavering support. You have not ceased to comfort me and you have always been able to cheer me up.

Abstract

Modern vehicles have become increasingly computerized to satisfy the more strict safety requirements and to provide better driving experiences. Therefore, the number of electronic control units (ECUs) in modern vehicles has continuously increased in the last few decades. In addition, advanced applications put higher computational demand on ECUs and have both hard and soft timing constraints, hence a unified approach handling both constraints is required. Moreover, economic pressures and multi-core architectures are driving the integration of several levels of safety-criticality onto the same platform. Such applications have been traditionally designed using static approaches; however, static approaches are no longer feasible in highly dynamic environments due to increasing complexity and tight cost constraints, and more flexible solutions are required. This means that, to cope with dynamic environments, an automotive system must be adaptive; that is, it must be able to adapt its structure and/or behaviour at runtime in response to frequent changes in its environment.

These new requirements cannot be faced by the current state-of-the-art approaches of automotive software systems. Instead, a new design of the overall Electric/Electronic (E/E) architecture of a vehicle needs to be developed. Recently, the automotive industry agreed upon changing the current AUTOSAR platform to the "AUTOSAR Adaptive Platform". This platform is being developed by the AUTOSAR consortium as an additional product to the current AUTOSAR classic platform. This is an ongoing feasibility study based on the POSIX operating system and uses service-oriented communication to integrate applications into the system at any desired time.

The main idea of this thesis is to develop novel architecture concepts based on adaptation to address the needs of a new E/E architecture for Fully Electric Vehicles (FEVs) regarding safety, reliability and cost-efficiency, and integrate these in AUTOSAR. We define the ASLA (Adaptive System Level in AUTOSAR) architecture, which is a framework that provides an adaptive solution for AUTOSAR. ASLA incorporates tasks-level reconfiguration features such as addition, deletion and migration of tasks in AUTOSAR. The main difference between ASLA and the Adaptive AUTOSAR platform is that ASLA enables the allocation of mixed critical functions on the same ECU as well as time-bound adaptations while adaptive AUTOSAR separates critical, hard real-time functions (running on the classic platform) from non-critical/soft-real-time functions

(running on the adaptive platform). To assess the validity of our proposed architecture, we provide an early prototype implementation of ASLA and evaluate its performance through experiments.

Keywords: AUTOSAR, E/E architecture, Runtime adaptation, Real-time systems

Table of contents

List of figures

List of tables

Nomenclature

Acronyms / Abbreviations

ABS Antiblockiersystem

ADAS Advanced Driver Assistance Systems

AET Average Execution Time

AUTOSAR AUTomotive Open System ARchitecture

BSW Basic SoftWare

CAN Controller Area Network

CBS Constant Bandwidth Server

CDD Complex Device Driver

COM Communication

CPU Central Processing Unit

DBF Demand Bound Function

DDN Dynamic Decision Network

DL Door Locks

E/E Electric/Electronic

ECU Electronic Control Unit

ECUAL ECU Abstraction Layer

EDF Earliest Deadline First

FEV Fully Electric Vehicle

HW Hardware

I/O Input/Output

IRV Inter Runnable Variable

ISR Interrupt Service Routine

LIN Local Interconnect Network

MCAL Micro Controller Abstraction Layer

MCU Micro Controller UNIT

MM Mapping Manager

NFR Non Functional Requirement

OEM Original Equipment Manufacturer

OIL Osek Implementation Language

OS Operating System

OSEK, OSEK/VDX Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug

PCP Priority Ceiling Protocol

PDF Probability Distribution Function

POSIX Portable Operating System Interface

QoS Quality of Service

RAM Random Access Memory

RM Rate Monotonic

RM Reconfiguration Manager

ROM Read Only Memory

RTE RunTime Environment

RTOS Real time Operating System

SAS Self Adaptive System

SBW Steer By Wire

SW Software

SWC Software Component

VFB Virtual Function Bus

WCET Worst Case Execution Time

WS Wiper Status

Chapter 1

Introduction

Modern vehicles increasingly offer Advanced Driver Assistance Systems (ADAS) such as collision avoidance and adaptive cruise control. These functions further require large and complex software systems that have to respect timing and safety requirements. Cost reduction requires the integration of mixed criticality functions on the same Electronic Control Unit (ECU).

> **Definition 1** *ECU is an embedded computer system that is used to control and regulate various functions in the vehicle. The ECU is a networked component that consists of both hardware and software elements [85].*

Traditionally, ADAS applications have been designed using static approaches, i.e., design-time mechanisms are used to validate and allocate the applications' functions. However, static approaches no longer meet the complexity of today's systems that in particular increases the number of potential failures. As a result, ADAS need to be adaptive.

> **Definition 2** *An adaptive system is a system that is, able to change its structure and/or behavior at runtime in response to environmental changes or failures [42].*

As we can see in Figure 1.1, initially, the automobile was completely mechanical. Gradually, embedded systems were introduced into various subsystems. Examples are electronic Engine Management System, Anti-lock brakes, power windows, etc. Then came the desire for dynamic. The first functionality to go dynamic was the automatic transmission, which requires little human intervention during operation, and is, therefore, adaptive according

to our definition. Next came the cruise control and traction control functionality, which are enabled by embedded systems, and incorporate a certain level of adaptivity within them. Notice that at this point, the automobile has three features with varying levels of adaptivity, but the automobile as a whole is still not fully adaptive and/or autonomous. Also, at this point conflicts start appearing between functionalities. For instance: What if the cruise control system wishes to accelerate the vehicle while simultaneously the traction control system wishes to apply the brakes? Such undesirable interaction between functionalities is termed *functionality and/or feature interaction* in the engineering lexicon [82][64].

As the complexity and the autonomy of individual functionalities grow, the automotive industry is finding increasingly difficult and costly to deal with such functionality interactions and to design safe and reliable vehicles.

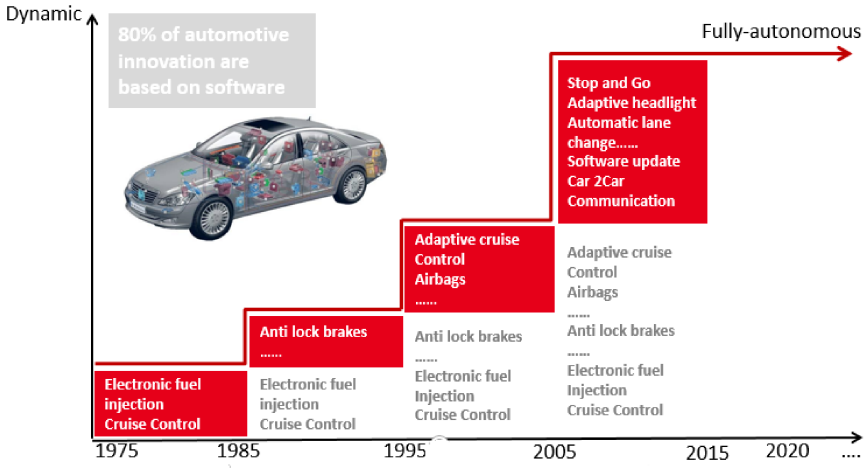

Fig. 1.1 Growth of automotive embedded systems [112].

Currently, AUTOSAR [11]– a widely used automotive operating system and framework– has been used by the automotive industry to cope with the increasing complexity of in-vehicle software. AUTOSAR decouples the Basic SoftWare (BSW) that needs to exist in every ECU, from the application software that is node-specific. Automotive systems are very cost-sensitive, and the ECU hardware is traditionally kept to a minimum. Therefore, AUTOSAR has been designed to execute with limited resources. Hence, the system's configuration is fixed at design-time with no support for runtime adaptation.

Making AUTOSAR adaptive requires specific support at different levels of the software architecture. The most important component affecting adaptivity is the Operating System (OS), but some flexibility can be introduced in the Run-Time Environment (RTE). Therefore, an architectural solution is needed that can handle the adaptation of applications with soft and hard (mixed) real-time as well as safety requirements. With the goal of creating a new architecture that can satisfy the above requirements, in this dissertation, we present a new E/E architecture that tackles various challenges over the

current E/E architecture. In particular, the challenges that this thesis addresses can be stated in terms of the following three Research Questions:

• RQ1: How to enhance the current AUTOSAR tasks model in order to deal with runtime adaptation?

• RQ2: How to enhance the RTE to ensure flexibility in AUTOSAR configuration?

• RQ3: How to manage migration, replacement, and insertion of a new software component on the platform?

Our work revolves around the following thesis statement: "*The focus of my thesis is to design and develop an* ***architectural solution*** *and strategies for software reconfiguration in automotive systems, and integrate those in* ***AUTOSAR*** *in order to contribute to making automotive systems more* ***evolvable****.*"

1.1 Scope and Requirements of the Thesis

The observations made hitherto lead us to the following scope and requirements which drive the work described in this thesis. We use Kiviat diagrams [104] to show visually the characteristics of our solution (Figure. 1.2) and the requirements we consider for adaptive automotive systems (Figure. 1.3). These diagrams provide developers of automotive software an easy way of viewing the characteristics of their applications. The dimensions represent axes of the Kiviat diagrams and characteristics of the dimensions represent the set of properties to be met by our solution (i.e., the red bullets).

1.1.1 The Scope of our Research

The adaptation Model. The adaptation may typically be *synchronous* and/or *asynchronous* with respect to the execution of applications. In the *synchronous* case, the applications synchronize their execution, and new tasks (actions) are introduced only after all applications have finished performing the actions specified in the initial configuration. The schedulability analysis of the adaptation is thus not required because there is no interference between tasks before and after the adaptation (i.e., in the new system configuration). On the other hand, in the case of *asynchronous* adaptation, all the applications start changing their configuration as soon as they receive an adaptation trigger without considering the behavior of the other applications. As a result, actions of the initial system configuration run concurrently with the new ones during the transition, which calls for schedulability analysis.

Promptness. Adaptation is well suited for systems that require reactive behavior, there is no need to wait until an idle period or slack before performing such a change as in Tindell's model [110] or at the end of cycles like in some approaches based on cyclic executive scheduling [97]. The adaptation may be classified in three different categories in

the time domain: (i) *time-based adaptation:* These are adaptations where we know the arrival time of the adaptation request in advance. (ii) *event-driven adaptation:* These are adaptations triggered by events rather than time. We don't know exactly when they happen. (iii) *irregular event-driven adaptation*: These are adaptations when no prediction can be made about the arrival time of the adaptation request. An example of this type is a system fault. The system may change its configuration and migrate to a degraded one where not all the functionalities are provided. In our work, we consider all three types of adaptation.

Scheduling policy. The use of dynamic priorities scheme suits better with systems running in highly dynamic environments [97][73].

Scheduling algorithm. Combining two scheduling mechanisms Constant Bandwidth Server (CBS) and Earliest Deadline First (EDF) has proven its efficiency to solve the problem of temporal isolation due to the integration of soft and hard real-time applications on the same platform [73][99].

Architecture. Distributed architecture. In our work, a real-time distributed system is defined to be a system with multiple autonomous processing units (ECUs) cooperating together to achieve a common goal. We use the term distributed architecture to refer to loosely coupled architectures where message passing is required (full connectivity is assumed between ECUs, i.e., each of the ECUs is connected to each other).

Timing requirement criticality. In our solution we are dealing with mixed-criticality systems, to the best of our knowledge, no one has applied runtime adaptation considering both soft and hard real-time constraints in AUTOSAR.

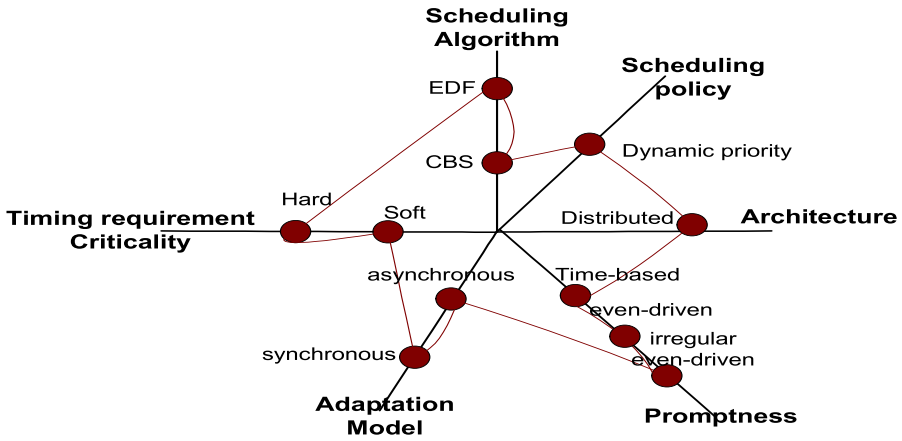

Fig. 1.2 The scope of our research.

1.1.2 Requirements for Runtime Adaptation in Real-Time Systems

(R1) Timeliness: The timeliness requirements of an adaptation characterize the time constraints under which the adaptation is executed. Hard real-time constraints require the execution of adaptation within a firm deadline. Adaptations executed under soft real-time constraints minimize the adaptation execution and blackout time (which is the time the application is unavailable due to state transfer and reconfiguration). Unbounded adaptations are executed without any time bound [52].

(R2) Consistency: Preserving the system consistency and leaving the system under change in a correct state after adaptation are two major requirements that must be ensured when performing adaptation of the running system. Many adaptation approaches freeze the entities to be reconfigured into an adaptation safe state called quiescent state [72].

(R3) Flexibility: The dynamic behavior of real-time systems requires executing applications with certain flexibility requirements in which the temporal properties and the number of applications vary during runtime. That means, the tasks of flexible application provide implementations that can adapt their execution to the available processing resources. These tasks have variable period and/or may demand variable WCET (i.e., stochastic [73]). So, executing flexible applications prevents the use of an efficient static temporal partitioning of the processing time. A static temporal partitioning that lasts over the entire lifetime of a system would result in an oversized system. Hence, in order to efficiently use the processing time, the flexibility of applications and possible demand changes during runtime have to be considered during runtime analysis.

(R4) Adaptation trigger: Adaptation can be triggered either *internally* due to the monitoring infrastructure or *externally* requested from an outside entity, for example the user. Furthermore, adaptation triggers may arrive synchronously (i.e., at a specific time) or asynchronously (i.e., with unknown arrival pattern).

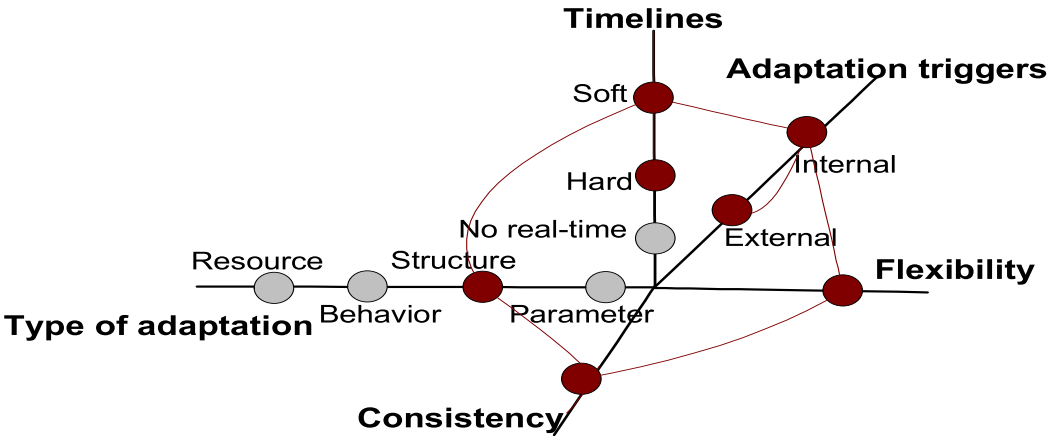

Fig. 1.3 Requirements for runtime adaptation.

(R5) Type of adaptation: The type of adaptation defines what is being reconfigured. We distinguish: *Resource adaptation* dynamically allocates resources based on current conditions. *Software adaptation* is a category that comprises *parameter adaptation, behavioral adaptation, and structural adaptation. Parameter adaptation* involves modifying variable values that determine program behaviour. *Behavioral adaptation* changes the behavior of the application, e.g., a change in distance to the vehicle ahead (e.g., driving closer to a lead vehicle with ABS). *Structural adaptation* changes the software architecture of the application, e.g., by removing a SWC, introducing a new one or replacing/updating an existing SWC with another newer version.

.

1.2 Research Approach and Contributions

To address the needs described above, we have been developing in this dissertation *the Adaptive System-Level in AUTOSAR (ASLA)* solution (Figure 1.4) which incorporates **runtime adaptation to AUTOSAR while maintaining mixed Soft and Hard real-time requirements**. ASLA provides the ability to dynamically reconfigure the system, such as adding a new application or moving an existing application to a different ECU. In AUTOSAR, the system configuration is, by design, static: the AUTOSAR Run-Time Environment (RTE) is configured at design-time for specific ECUs and partly generated based on the requirements of the software components (SWCs). A reconfiguration of the system, such as adding an application or moving an application from one ECU to another, cannot be done dynamically at runtime. ASLA extends AUTOSAR in two ways. It changes the scheduling policy from a fixed-priority (assigned to tasks at design time) to a dynamic pre-emptive policy based on EDF and CBS schedulers to guarantee mixed critical requirements. ASLA also contains RTE extension that supports the runtime application migration between ECUs in response to anticipated changes caused by the environment, such as network connectivity, as well as unexpected failures in both software and hardware. The deployment and reconfiguration of an application onto an ASLA system is handled by a SWC called the Adaptive Software Component that is responsible for reconfiguring applications running on the system and composed of: (i) a Monitor, that monitors events triggering adaptation, (ii) a Mapping Manager, offering a dynamic deployment of tasks on the ECUs and (iii) a Reconfiguration Manager, which automatically reconfigures tasks inside /or between the different ECUs. The plug-in offers a task execution container and enables any task launched by ASLA to be periodically executed.

In particular, this dissertation involves a combination of:

1. **Contribution 1: Issues of AUTOSAR to support runtime adaptation**

   We review the challenges and issues about current AUTOSAR ECU status.

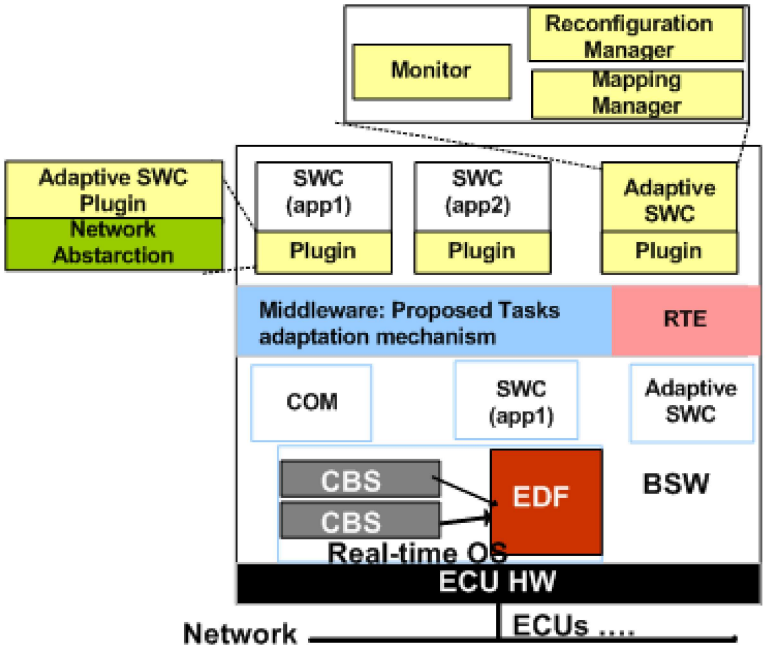

Fig. 1.4 Overview of ASLA architecture.

Mainly, we describe the major drawbacks with its current task model, runtime environment, mode manager, etc. Second, we show why it is necessary to change the AUTOSAR architecture to make it adequate for reconfiguration. Third, we present an approach towards making AUTOSAR dynamic, starting from the application down to the operating system level. Surprisingly, the identified issues of AUTOSAR and the problem tackled in this dissertation are recently becoming the focus of the newly established branch of the AUTOSAR initiative, the so-called Adaptive AUTOSAR platform [10].

2. **Contribution 2: Runtime adaptation support for AUTOSAR.**

   We propose a layer called ASLA (Adaptive System-Level in AUTOSAR) to incorporate tasks-level adaptation features in AUTOSAR. ASLA aims at extending the AUTOSAR architecture, starting from the application down to the operating system layer (task model and RTE)(i.e., extending AUTOSAR ECU Software architecture).

3. **Contribution 3: Resource allocation and scheduling for automotive systems.** We have developed a task-partitioning strategy for allocating SWCs to the ECUs based on the work of [99]. In [99], the authors propose TSMBA, "Tabu Search Mapping and Bandwidth Allocation" algorithm, which provides a comprehensive solution that allocates mixed hard and soft SWCs onto a distributed architecture and performs processor bandwidth reservation for guaranteeing timing requirements even in the presence of faults. TSMBA [99] cannot be used directly for our goal. Specifically, TSMBA cannot be used directly in systems with tasks dependencies (i.e., pipeline task model). Therefore, we propose a new allocation algorithm based

on TSMBA. To incorporate task dependencies, we have defined an abstraction called Operational Chains, which enables timing analysis. An ECU assignment methodology called O-TSMBA (Operational chains-Tabu Search Mapping and Bandwidth allocation) is introduced to consider task dependencies. TSMBA is also adapted to an AUTOSAR-compliant platform so that the proposed algorithm can be used in the automotive context.

4. **Contribution 4: Proposing an algorithm for task mapping, bandwidth allocation and reconfiguration for mixed hard/soft distributed systems.**

   Conventional real-time theories are in general applicable to automotive systems. However, they do not incorporate highly dynamic attributes and hence do not provide tight schedulability analyses. To the best of our knowledge, TSeRBA is the first algorithm that supports runtime adaptation while taking into account schedulability analysis and task allocation for mixed hard/soft applications in AUTOSAR.

1.3 Thesis Outline

In accordance with the contributions we specified above, we structure our thesis as follows:

- Chapter 2 gives a more detailed introduction to the field of embedded real-time systems. The reader is given a deeper insight into the specific challenges during the design and implementation of safety critical systems being subject to real-time constraints and distributed topology. The automotive systems targeted in the thesis are clearly within that field. In addition, the chapter surveys existing approaches related to this dissertation. Our work falls into the following categories: (1) runtime adaptation in real-time embedded systems and (2) mode change protocols.

- Subsequently, in Chapter 3, the architecture of automotive systems is discussed. Furthermore, a summary of current automotive software development concepts is given. In addition, a summary of existing approaches for runtime adaptation in automotive systems is discussed.

- Chapter 4 describes our distributed framework called ASLA (Adaptive-System-level in AUTOSAR) used to incorporate tasks-level adaptation techniques in AUTOSAR. In addition, ASLA's development methodology is given.

- Chapter 5 introduces ASLA's algorithms to support dynamic task re-allocation and task-level adaptation.

- Chapter 6 presents our prototype implementation of ASLA along with a set of extensive evaluations. In addition, the results of the resource allocation and schedulability analysis methods used in ASLA are reported.

- Chapter 7 emphasizes the contribution of our work and identifies challenges and research gaps that require further exploration.

  A portion of our work has been published in the following conference and workshop paper:

- **Conference paper:**

  - In [23], we presented the design and the implementation of ASLA's framework.

- **Workshop paper:**

  - In [24], we presented the open issues and the necessary requirements for making AUTOSAR adaptive.

- **Ongoing conference paper:**

  - Dynamic Software Adaptation and Reconfiguration in AUTOSAR. In this ongoing publication, we will present the ASLA's theoretical studies and algorithms.

Chapter 2

Foundations of Embedded Real-Time Systems

The software development concept presented in this thesis is targeted at automotive systems, which belong to the family of embedded real-time systems. Real-time systems are typically operating under stringent timing constraints. The moment in time the system sends a result is as important as the result itself for correct behavior. As these systems are not visible as computers in an automobile, but rather interact with their environment using sensors and actuators, they are called embedded systems. In addition, automotive systems are typically distributed, which refers to their structure as an interconnected network of computing nodes, exchanging their respective current states. We summarize the key aspects of real-time and embedded systems in Sections 2.1 and 2.2, respectively, to give a better understanding of the general conditions for the implementation of automotive systems. In Section 2.3, we give a short overview of the actual practice of embedded real-time systems development. Section 2.4 defines the concept of self-adaptive systems. A survey of existing approaches for adaptive real-time embedded systems is presented in Section 2.5. Finally, in Section 2.6 we conclude with a summary of Chapter 2.

2.1 Real-Time Systems

The Oxford dictionary defines **real-time** as “the actual time during which a process or event occurs". In computer science, a real-time system is one where its timing requirements are an integral part of its behavioral specification.

Definition 3 . *Real-Time System:* *A real-time computer system is a computer system, in which the correctness of the system behavior depends not only on the logical results of the computations, but also on the physical instant at which these results are produced–– Hermann Kopetz [68]*

Definition 3 below is given by Kopetz [68]. This definition means that in strict real-time systems a late result is not just late but wrong. The meaning of "late" of course has to be defined dependent on the specific application. In case of an air-bag controller, it is intuitively clear what real-time means and it is easy to understand that a late firing of the air-bag is not only late but definitely wrong.

As most real-time systems are control systems, several other definitions consider a real-time system as a reactive system having timing specifications—that changes its state as a function of physical time, e.g., a chemical reaction continues to change its state even after its controlling computer system has stopped [50]. Based on this, a real-time system can be decomposed into a set of subsystems, i.e., the controlled object, the real-time computer system and the physical world. A real-time computer system must react to stimuli from the controlled object (or the physical world) within time intervals dictated by its environment. This interaction with the environment is done by acquiring information from sensors and acting upon that environment through actuators, as depicted in Figure 2.1.

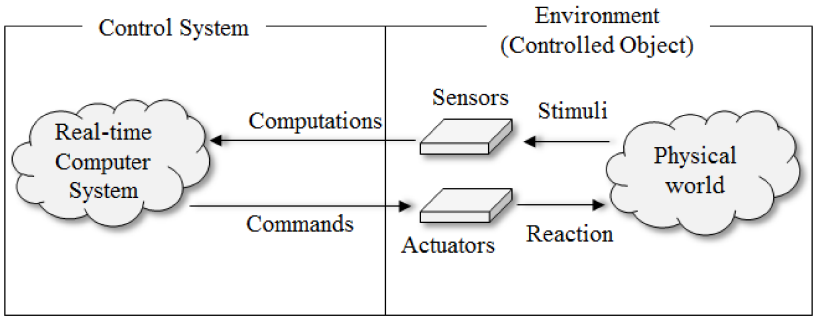

Fig. 2.1 Schematic representation of a real-time system.

2.1.1 Soft and Hard Real-Time Systems

Depending on time constraints, real-time systems can be organized into two main categories:

– **Hard Real-Time Systems:** *Hard real-time systems* have very strict time constraints, in which missing the specified deadline is unacceptable. The system must be designed to guarantee all time constraints. Many safety-critical systems are hard-real-time systems. Examples of hard-real-time systems include embedded tactical

systems for military applications, flight mission control, traffic control, production control, robotics, nuclear plant control, etc [68].

Figure 2.2 shows an example of a hard real-time safety-critical application that determines driver's intentions and driving conditions in real-time.

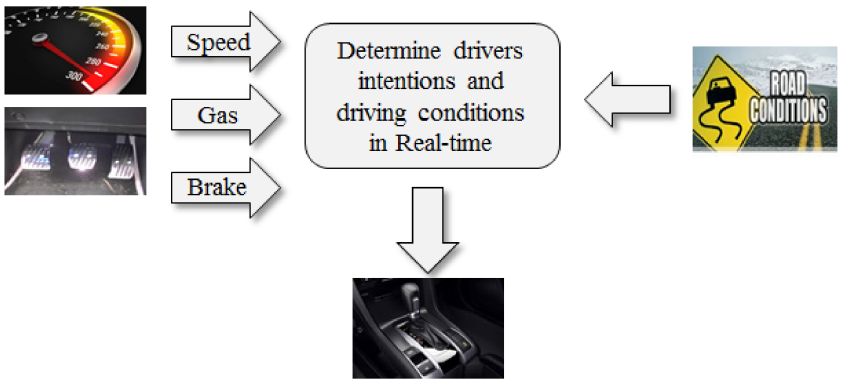

Fig. 2.2 An automobile shift control system is an example of a hard real-time safety-critical system.

- **Soft Real-time Systems:** *Soft real-time systems* also have time constraints; however, missing some deadlines may not lead to a catastrophic failure of the system. Thus, soft real-time systems are similar to hard real-time systems in their infrastructure requirements, but it is not necessary that every time constraint is always met. In other words, some time constraints are not strict, but they are nonetheless important. A soft real-time system is not equivalent to a non-real-time system, because the goal of the system is still to meet as many deadlines as possible. Typical examples of applications with soft constraints are multimedia applications.

- **Mixed Soft and Hard Real-time Systems:** *Mixed soft and hard real-time systems* are dual criticality systems with soft and hard tasks. In these systems, in contrast to soft tasks, the timely execution of hard tasks is guaranteed. Soft real-time tasks are scheduled in a best-effort strategy, for example, to reduce their response time [36].

2.1.2 Internal Structure of a Real-time System

The structure of a real-time system is mainly composed of two layers: (i) hardware architecture and (ii) software architecture. Figure 2.3 illustrates the general structure of a real-time system. Starting from the hardware layer up to the software layer. The hardware layer includes computation resources (i.e., processors), communication resources (i.e., networks), storage resources (i.e., memories) and I/O peripherals (i.e., sensors and actuators). The software layers provide most of the features and flexibility in the system. These layers include: low-level drivers, a Real-Time Operating System (RTOS) and

the application. The RTOS is composed of different modules providing communication services, synchronization, and execution. It is the responsibility of the RTOS to ensure that all the applications meet their respective time constraints.

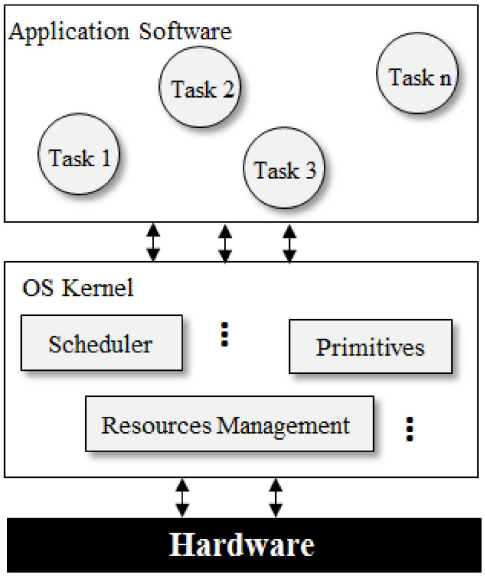

Fig. 2.3 Internal structure of a real-time system.

We examine in next sections how an RTOS can ensure that applications meet their respective timing constraints through appropriate scheduling strategies. There are open-source RTOS (e.g., FreeRTOS, MicroC/OS-II and RTEMS) or a closed source executive (from over 300 commercial/proprietary options).

The software layer contains a computer program that controls the environment of a real-time system. This program may comprise several execution entities treated by the operating system [39]. These entities are called **processes** or **tasks**.

Real-time tasks are also classified with respect to their real-time constraints. We distinguish: periodic tasks, aperiodic tasks and sporadic tasks [118] [101].

1. Periodic tasks make a resource request at regular periodic intervals. The processing time and the time elapsed between the request and the deadline are always the same for each request of a particular task; they may however be different for different tasks. Monitoring patient's vitals is an example of periodic task system. Figure 2.4 summarizes the characteristics of a periodic task, a task τ_i is characterized by the following timing parameters:

   – the **arrival time** a_i or **release time** r_i is the time at which the task is ready for execution.

- the **execution requirement** C_i, specifies an upper limit on the execution requirement of the generated task; it is also called the worst-case execution time (WCET).
- the **period** or **minimum inter-arrival time** T_i, denotes the temporal separation between the arrival times of the task.
- the **absolute deadline** d_i is the time at which the job should be completed.
- the **relative deadline** D_i, is the time length between the arrival time and the absolute deadline.
- the **start time** s_i is the time at which the task starts its execution.
- the **finishing time** f_i is the time at which the tasks finishes it execution.
- the **response time** R_i is the time length at which the task finishes its execution after its arrival, which is $f_i - a_i$.

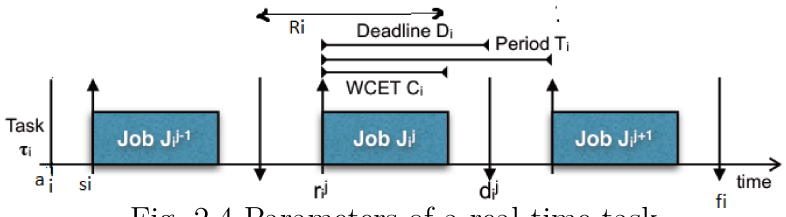

Fig. 2.4 Parameters of a real-time task.

A real-time task is characterized by the relation between its deadline and its period:

- **Implicit-deadline**: task τ_i has a relative deadline equal to its period ($D_i = T_i$).
- **Constrained-deadline**: task τ_i has a relative deadline less than or equal to its period ($D_i \leq T_i$)
- **Arbitrary-deadline**: task τ_i has a relative deadline which can be less, equal or greater than its period.

2. Aperiodic tasks are tasks that are executed on demand (or with unknown period). A task is executed in response to an event. It should be noted that the arrival time may not be specified in some systems, and the ready time is defined by the arrival of an event.

3. Sporadic tasks are similar to periodic tasks, except that the parameter T_i denotes the *minimum* rather than the exact, separation between successive tasks. These tasks are defined random arrival times.

A task set is referred to as *synchronous* or *asynchronous* based on the first activation scenario of its tasks. In the case of a synchronous task set, the task set is defined as a set whose first job of its tasks are activated at the same time. Whereas in the case of asynchronous task set, the first jobs are activated at different time.

2.1.3 Real-Time Scheduling

A real-time scheduling is defined as the process that defines the execution order of tasks on a platform of processors. The main problem that real-time theory is concerned with is the following: **The feasibility analysis**— *Given* the specification of a task system, and real-time constraints on the scheduling environment determine whether there exists a schedule for the task system that will meet all deadlines [16]. The Real-Time Operating System (RTOS) is an example of such systems that is responsible for choosing which tasks to execute on which processor and at what time. The main objective of a real-time scheduler is to guarantee the correctness of the results while respecting the timing constraints of the tasks (no deadline miss). It is important to clarify that real-time scheduling does not necessarily mean executing tasks as soon as possible, but taking scheduling decisions that guarantee their timing constraints. Furthermore, a real-time scheduling is divided into two categories based on the scheduling decisions and when they are taken:

1. **Static scheduling**: A real-time system is scheduled based on a scheduling table that contains all scheduling decisions of the system and the activation times of all tasks. Hence, the scheduling decisions are taken prior to the running of the system and they rely on a knowledge of the process behavior (i.e., at compile time). The main advantage of static scheduling is its simplicity (hence a small overhead), no runtime overhead is incurred by such a method since all the scheduling decisions are made at design time [39]. However, this scheduling approach is too rigid to cope with environmental changes occurring during the system execution, since it assumes that all the parameters, including the time to wake-up tasks, are fixed at design time [100].

2. **Dynamic scheduling**: Scheduling decisions are taken during the execution of the system based on certain priority assignment rules defined by the scheduling algorithm. Depending on whether the scheduling decisions are taken at runtime or at design time, the scheduler uses different algorithms (or policies) to schedule, define the priority assignment of tasks and select which tasks to execute on which processors. We distinguish three main categories:

   - **First Come First Served Scheduling (FCFS):** FCFS uses a simple "first, in first out" (FIFO) queue [89]. Tasks are dispatched according to their arrival time on the ready queue. Being a non-preemptive method, once a task has a CPU, it runs to completion.
   - **Fixed-Priority Scheduling (FPS):** This algorithm is well suited for static, hard real-time systems. In this algorithm the priorities are assigned to tasks at design time. Priorities can be assigned on the basis of *urgency*– that is,

tasks that have the shortest timing constraints will have the highest priority level. Liu and Layland [78] proposed two methods for assigning fixed priorities to tasks based on urgency, *Rate Monotonic* **(RM)** [76] and *Deadline Monotonic* **(DM)** [9]. In **RM**, tasks are assigned priorities according to periods such that tasks with shorter periods get higher priorities. **DM** assigns the highest priority to the task with the shortest deadline.

- **Dynamic-Priority Scheduling (DPS):** This algorithm works similarly to FPS with the difference that the task priorities are computed during the execution of the application rather than during the design time. Several methods fall in this category, the *Earliest Deadline First* (EDF) is one of them. EDF scheduling is one of the first DPS proposed [62][6]. As the name implies, tasks are selected for execution in the order of their deadline. The *Least Laxity First* **(LLF)** is another example: the highest priority task is the task with the smallest *Slack time* (i.e., the temporal difference between the deadline, the ready time and the run-time).

3. **The Constant Bandwidth Server (CBS):** The Constant Bandwidth Server (CBS) is a scheduling mechanism proposed by Abeni and Butazzo [3] to implement resource reservations in EDF-based systems. Resource reservations are an effective technique to support soft and hard real-time applications in open system environment [5]. The idea behind this mechanism is that each task (or set of tasks) is assigned a fraction of the CPU, and is scheduled in such a way that it will never demand more than its reserved bandwidth. With this abstraction, the processor capacity is viewed as a quantifiable resource that can be reserved, like physical memory. In particular, this approach provides (i) *temporal isolation* (see Definition 4) between tasks and (ii) *schedulability analysis for hard real-time tasks*. A fully detailed description of CBS can be found in [73]. Intuitively, CBS divides the processor into virtual chunks by allocating a certain processor time for the tasks, as depicted in Figure 2.5.

   Each task is allocated a Constant bandwidth server S_i with two parameters Q_s, where Q_s is the *server budget* and T_s is the *server period*. The *server bandwidth* $U_s = Q_s/T_s$ is the fraction of the CPU bandwidth assigned to S_i. The algorithm dynamically updates two variables (q_s, δ_s) for each server S_i: q_s is the server's*current budget* and keeps track of the consumed bandwidth; δ_s is the server's *current scheduling deadline*. Initially, q_s is set to the maximum budget Q_s and δ_s is set to 0. S_i is *active* if the corresponding task has a pending instance. The CBS reservations are implemented by means of an Earliest Deadline First (EDF) scheduler. At each instant, the

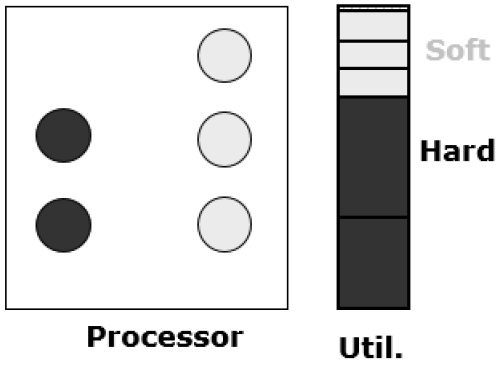

Fig. 2.5 CBS scheduling mechanism.

active server with the earliest deadline δ_s is selected and the corresponding task is dispatched to be executed. The CBS updates its parameters as follows:

• **Rule A:** When job $J_{i,j}$ of a task τ_i arrives at time $a_{i,j}$, the server checks the following condition:

$$q_s <= Q_s.\frac{\delta_s - a_{i,j}}{T_s}$$

If the condition holds, the current pair (q_s, δ_s) is computed as $q_s \leftarrow Q_s$ and $\delta_s \leftarrow a_{i,j} + T_s$.
• **Rule B:** If the server S_i executes for Δt units of time, the budget is decreased accordingly: $q_s <= q_s - \Delta t$.
• **Rule C:** Server S_i is allowed to execute while $q_s > 0$. When the budget is exhausted ($q_s = 0$) and the server job has not finished yet, a new pair (q_s, δ_s) is computed, the scheduling deadline is postponed to $q_s \leftarrow q_s + T_s$ and the budget is recharged to $q_s \leftarrow Q_s$.

As a consequence, each task is reserved an amount of computation time Q_s in each server period T_s regardless of the behavior of the other tasks (as explained in the CBS rules). As we said above this is the *temporal isolation* property and it holds as long as the system satisfies the following schedulability condition:

Theorem 1 *Given a system of n servers with* $\sum_{s=1}^{n} U(T_s) < 1$*, no server misses its scheduling deadline, regardless of the behavior of the other tasks [4].*

Definition 4 . *Temporal Isolation:* *A kernel mechanism able to enforce resource reservation on the processor, so that the tasks running within the reservation are guaranteed to receive the reserved amount of time and do not interfere with the others for more than the reserved amount [39].*

Real-time systems are also distinguished based on their implementation:

- **Preemptive systems**: The running task can be interrupted at any time by another task with higher priority, and be resumed to continue when all higher priority tasks have completed. In other systems, preemption may be disabled for certain intervals of time during the execution of critical operations (e.g., interrupt service routines, critical sections, etc). In some situations, preemption can be completely forbidden to avoid unpredictable interference among tasks and achieve a higher degree of predictability (although higher blocking times) [39].

- **Non-preemptive systems** : They do not permit preemption. It is easier to design preemptive scheduling algorithms for real-time systems. However, non-preemptive scheduling is more efficient, particularly for soft real-time applications, than the preemptive approach caused by the reduced overhead needed for switching among tasks [65] [63].

2.1.4 Validation and Verification of Real-Time Systems

In order to ensure that a real-time system is temporally correct, we need to analyze its schedulability (see Definition 5). Several approaches are used to determine whether a set of tasks meets its timing constraints [20] [77]. In general, these approaches are based on a well-defined schedulability conditions that are founded on temporal requirements of system's tasks. Therefore, schedulability conditions can be defined to determine the status of the task set and whether it is schedulable or not using a given scheduling algorithm $\mathcal{A}$ before its implementation. There are three types of schedulability conditions w.r.t a given algorithm $\mathcal{A}$: sufficient, necessary and exact conditions (see Definition 5). These conditions depend on the special features of the systems for which they are used.

Definition 5 . *Real-time Feasibility and Schedulability:*
*For a given scheduling algorithm $\mathcal{A}$, a system is referred to as $\mathcal{A}$–**schedulable** if all its tasks meet their deadlines when scheduled with the algorithm $\mathcal{A}$. Similarly, a task set is said to be $\mathcal{A}$–**schedulable** if all of its tasks are$\mathcal{A}$–**schedulable**.*
Sufficient condition: if the condition is true then the task set is deemed $\mathcal{A}$–schedulable, otherwise, it cannot be concluded whether the set of tasks is schedulable or not.
Necessary condition: if the condition is false, then the task set is unschedulable using algorithm $\mathcal{A}$; otherwise, it cannot be concluded whether the set of tasks is schedulable or not
Exact condition: it is the combination of sufficient and necessary conditions. If the condition is true, then the task set is $\mathcal{A}$–schedulable, otherwise it is unschedulable.

According to [16] [21], three approaches are used for the schedulability analysis : (i) the processor utilization,(ii) the demand bound function and (iii) and response time analysis. In the sequel, we detail(i) and (ii); details about (iii) can be found in [8].

- **The processor utilization**: Informally, task utilization represents the fraction of computational capacity that a task requires on a single processor. The amount of execution over any interval of length t on processing platform Π that a task τ_i requires is upper bounded by $u_i \times t$. Below is a more formal definition of processor utilization [16].

  Definition 6 *The utilization $U(\tau)$ of a periodic or sporadic task T_i is defined to be the ratio of its execution requirement to its period: $U(T_i) = C_i/P_i$. The utilization $U(\tau)$ of a periodic or sporadic task system τ is defined to be the sum of all the utilization of all tasks in τ:*

$$U(\tau) = \sum_{T_i \in \tau} U(T_i) \tag{2.1}$$

  The workload of real-time tasks can also be characterized by the *Demand Bound Function* which is defined as follows:

- **Demand Bound Function (DBF)** is useful for the purpose of schedulability analysis, to quantify the amount of computation required over an interval by a real-time instance. We call this quantity the demand over the interval. Informally, demand is an indication of how " temporally constrained" the system is over that interval. Below is a more formal definition of demand bound function [21].

Definition 7 *The Demand Bound Function of tasks τ_i in a time interval [0,t] for any $t > 0$ is defined as the sum of the execution time of all jobs of τ_i that have both their arrival time and deadline in τ_i:*

$$DBF(\tau_i, t) = max\left(0, \left\lfloor \frac{t - D_i}{P_i} \right\rfloor + 1\right).C_i \tag{2.2}$$

Consequently, the processor load [21] of task set τ is defined as :

$$load(\tau) = max_{\forall t>0}\left(\frac{\sum_{i=1}^{n} DBF(\tau_i, t)}{t}\right) \tag{2.3}$$

2.1.5 Real-Time Communication

Definition 8 . *Distributed System:* *A distributed system consists of a collection of autonomous computers, connected through a network and distribution middleware, which enables computers to coordinate their activities and to share the resources of the system, so that users perceive the system as a single, integrated computing facility [108].*

Real-time systems are often of *distributed* nature. A distributed real-time system architecture consists of a set of nodes and a communication network between these nodes. Thus, the system's functionality is not necessarily implemented on a single node, but may be achieved by an arbitrary number of cooperating nodes in the system (see Definition 8).

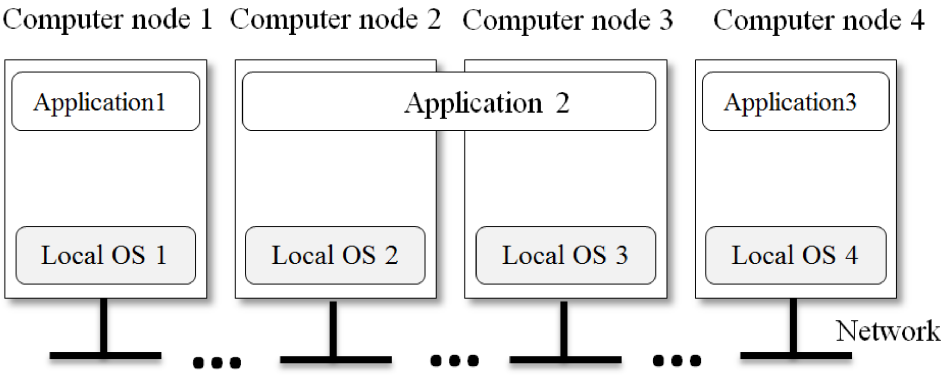

Fig. 2.6 Distributed system.

From a functional point of view there is no difference whether the computation is distributed over several nodes, or executed on a single node, as depicted in Figure 2.6. The distributed solution, however, yields several advantages, we cite some advantages from [68].

– **Architecture**: It may be advantageous to execute a function on dedicated hardware. This approach allows the employment of hardware components, for example, a digital signal processor, which are optimized for the respective function. The resulting node is able to calculate results faster or more efficiently. If several nodes with an identical hardware architecture can be used in the overall system, the opportunity of mass production might also reduce the cost per node. Other constraints for the system's architecture may originate from the historical evolution of the system. Considering an automotive system, the functionality of nodes has been largely improved by replacing simple circuitry with ECUs. The resulting system is a large set of small nodes. Often it is easier to situate the simpler, and hence smaller, nodes in the installation space available, than larger pieces of hardware. At the same time, the linking might be reduced if each node's physical location is close to the sensors and actuators it uses, compared to a star topology in a centralized system. Further, from a system architect's point of view, the abstraction provided by the distributed nodes eases the design of the overall system. As explained by Kopetz [68], each node is an autonomous real-time system with essential functional and temporal properties. The exact implementation of the node is hidden beneath its interface.

– **Extensibility**: With all nodes providing a common interface, easy extensibility of the real-time system is achieved. If the communication system provides the necessary resources, it is sufficient from an architect's point of view to add further nodes to the system by connecting them to the communication system. Of course this approach requires the subsystems to be composable, that is, their integration into the overall system via the communication mechanism must not invalidate the timeliness property achieved by the system.

– **Fault containment:** Hiding a node's internal state is also advantageous in case of failure of a node. Since each node is implemented as an autonomous subsystem, it can be prevented from influencing the rest of the system if it fails. The communication system is liable for providing a mechanism that avoids the propagation of erroneous values to other nodes of the system. This approach largely improves the reliability, and thus safety, of the overall system.

– **Parallelism**: The availability of a larger number of processors and memory, compared to a single node solution, enables the parallel execution of tasks. This is mainly of interest for large scale systems, where several independent tasks are ready for computation at the same time. Parallelization of the system not only improves processing speeds, but also lowers overall system cost.

2.2 Embedded Systems

An *embedded systems* consists of a physical device and an integrated computer system that controls the functions of the physical device acting in the physical environment [69]. Most of today's machines and appliances (e.g., automotive engines, washing machines, medical devices, mobile phones) are controlled by an embedded computer system (see Figure 2.7).

Fig. 2.7 Embedded system applications.

According to Kopetz [70], embedded systems have a number of distinctive characteristics that influence the system's development process:

Mass production. Many embedded systems are designed for a mass market and consequently for mass production in highly automated assembly plants. This implies that the production cost of a single unit must be as low as possible, i.e., efficient memory and processor utilization are of concern.

Static structure. Usually, the embedded system is used unaltered throughout its lifetime and has a dedicated use. The a priori known static environment can be analyzed at design time to simplify the software design and requirements. This avoids unnecessary complexity of the systems, since little flexibility or dynamic algorithms are needed.

Ability to communicate. In case of distributed embedded systems like cars, robust and deterministic behavior of the employed communication system is more important than transmission speeds. Therefore, the focus for the employed protocols results in predictable timing and absence of collisions.

Low power. Many mobiles embedded devices are powered by a battery and should consume low power.The lifetime of a battery load is a critical parameter for the utility of a system. Embedded systems consist of hardware and software components. The hardware components include: processor (1 or more), memory, I/O peripherals including network devices, sensors, actuators, timers, etc. The software components are mainly composed of two parts: (i) the system software layer that contains an Operating system and drivers of I/O devices, and (ii) the application software that runs on top of the OS and executes tasks that users wish to perform.

2.3 Embedded Systems Development

The design of embedded real-time systems is guided by a load of different prerequisites, arising from the system's environment, its intended functionality, and the safety requirements. The size and complexity of today's distributed systems (for example in automobiles, they are expected to reach up to 100 million lines of code in 2020 [103]), make the development of such systems difficult. In order to tackle this complexity, engineers/designers use an abstraction and models for the system description. These models enable the designers to solve different issues yield by the separation of the development steps [31] [32], [67]. Several researchers [47] [98] show that separating the system's behavior from its structure eases the design of complex real-time systems. Figure 2.8 depicts a design flow development of embedded systems. This design starts with the "system specification" step. Typically, it involves a consideration of both *software and hardware models*. Since both have to be taken into account during embedded system design. There are many ways to model a software (see [68] for a description of the common modeling formalism used in embedded system). In this thesis, we use task graphs [68], the exact model is presented in Chapter 5. For the hardware model, we consider heterogeneous distributed platforms, consisting of computation nodes connected to a bus. Details about the hardware model is presented in Chapter 5.

In the "design tasks" step, the system is partitioned into distributed components, the mapping of these components to their respective location (i.e., computation node) is achieved based on a model of the target platform. This step also comprises: design space exploration, compilation and the scheduling.

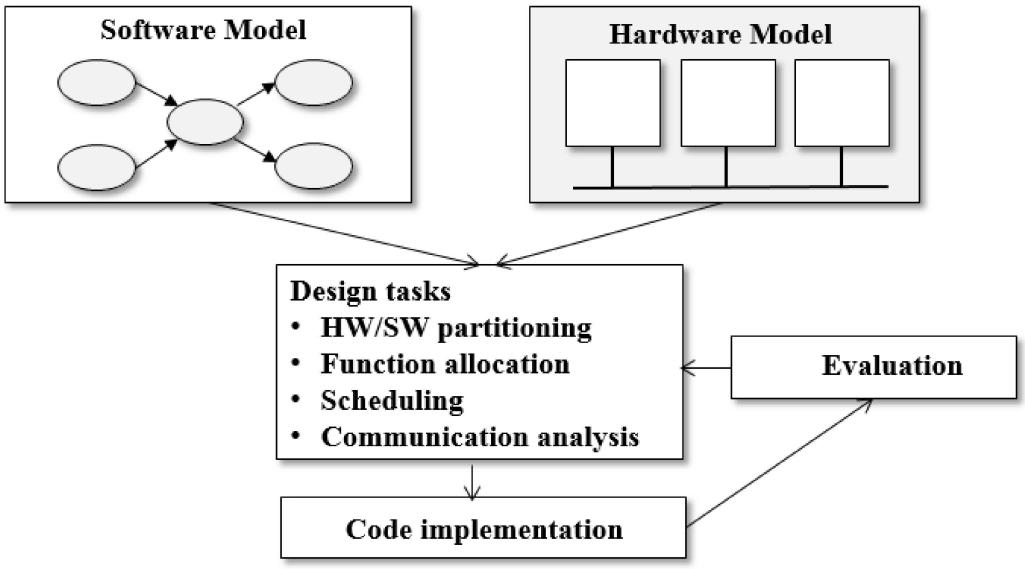

Fig. 2.8 Embedded system development.

2.4 Adaptive Systems

In [43], authors have defined self-adaptive systems as systems that are able to modify their behavior according to changes in their environment. Self-adaptation endows a software system with the capability to adapt itself to internal dynamics and dynamics in the environment in order to achieve certain goals. Examples are a system that heals itself when certain errors occur, or a system that optimizes its performance under changing conditions. Different loosely connected communities have studied self-adaptation. Prominent examples are the SEAMS community (International Symposium on Software Engineering for Adaptive and Self-Managing Systems, http://www.self-adaptive.org/). Self-adaptive systems automatically take the correct actions based on the knowledge of what is happening in the system, guided by objectives and needs of stakeholders. Self-adaptive systems are characterized by three core functionalities:

- Monitor (sensing) the environment to recognize "problems";
- Take decisions on which behavior to exhibit;
- Realize the behavior change by adaptation.

Therefore, to act adaptively, a self-adaptive system, which ideally bases on some specialized architecture to support previously mentioned functionalities, needs knowledge about:

- What to monitor and for which symptoms;
- Which alternative behaviors are available;
- Decision criteria for the selection of a specific behavior.

Several challenges were tackled in the research roadmap of self-Adaptive Systems [43]. We summarize the most important challenges, as follows:

Modeling dimensions, in which authors have defined models that can represent a wide range of system properties.

Requirements, in this challenge they have defined a language to capture the kind of uncertainty at the requirements phase and still there is need of means to manage it.

The third challenge is about *Engineering*, authors have set the role of feedback control loops in the life cycle of self-adaptive systems, in other word to show the importance of making the adaptation control loops explicit.

The last challenge is concerned by *assurance*, in which the authors have studied how to supplement traditional and V methods applied at requirements and design stages of development with run-time assurances. All these challenges result from the dynamic nature of self-adaptation, which brings uncertainty. This uncertainty restricts the applicability of traditional software engineering principles and practices but motivates the search for new approaches for developing, deploying, managing and evolving self-adaptive software

systems. In this thesis, we develop novel architecture concepts based on adaptation to address the needs of a new E/E architecture for FEVs regarding safety and cost-efficiency.

2.5 Runtime Adaptation in Real-time Embedded Systems

There is an increasing need for embedded software systems that are able to adapt dynamically to changes in their environment. However, there is still a lack of applicable techniques for handling adaptation [38, 37] in this setting. Several research initiatives have recently aimed to tackle runtime adaptation in real-time embedded systems. These approaches can be divided into two different groups targeting on two different scenarios: *(i) Runtime adaptation of embedded software:* which tries to add, replace, migrate or remove SWCs from the running system, and *(ii) Runtime adaptation and mode change concept:* which focuses on the switching between predefined configuration or modes of system at runtime.

2.5.1 Runtime Adaptation of Embedded Software.

Concerning this first group, we begin with the work of Marisol Garcia Valls *et al.* [57]. In this paper, two strategies for component replacements scheduling are presented. One of the replacements scheduling approaches allows the replacement of a component each time the component is executed, which leads to add the replacement time to the execution time of the component. This approach has a high processor time reservation; however, the replacement is accomplished in shorter completion time. The other approach allows the replacement of the component at a predefined safe time. The time reserved for replacements can be adjusted to the application needs.

The same authors (i.e., Marisol Garcia Valls *et al.*) introduced iLAND (mIddLewAre for deterministic dynamically reconfigurable NetworkeD embedded systems) [53, 56], a middleware that supports time-deterministic reconfiguration in distributed soft real-time service oriented systems. iLAND stands out the work carried out in real-time service composition, [54], and QoS models for supporting composition, [55], which deals with the problem of dynamic allocation of services with multiple available implementations and dynamic component-based reconfiguration [57]. The authors addressed the problem of finding a suitable system configuration with composition algorithms, which contain heuristics and figures of merit. The use of these techniques allows reducing the overhead when executing these algorithms by several orders in magnitude. Both references focus on service-based dynamic reconfiguration of soft real-time systems, and do not address mixed hard and soft real-time requirements of distributed real-time systems.

Patrick Lardieri *et al.* [75] have presented a Multi-Layer Resource Management (MLRM) architecture. MLRM was developed to demonstrate DRM (Dynamic Resource management) capabilities in a shipboard computing environment. This environment consists of a grid of computers that manage many aspects of a ship's power, navigation, command, control, and tactical operations, using standards-based DRM services that support multiple QoS requirements, such as survivability, predictability, security, and efficient resource utilization. MLRM was developed for the DARPA's Adaptive and Reflective Middleware Systems (ARMS) program, which is applying DRM technologies to coordinate a computing grid that manages and automates many aspects of shipboard computing, and to support DRM in enterprise distributed real-time embedded systems. The ARMS MLRM was designed to manage computing resources dynamically and ensures proper execution of missions in response to mission mode changes and/or resource load changes and failures, as well as capability upgrades. The work done in this paper is somehow closer to our focus however they considered hard real-time systems.

Rasche and Polze, in [95, 96], present an approach for dynamic reconfiguration of component-based software, in which components are blocked to apply management tasks such as reconfiguration of components. The reconfiguration is managed and applied in a way that the execution of all the current transactions between components is correct, but the application is blocked until the reconfiguration is finished without taking into account real-time deadlines. Other approaches are in fact concerned about real-time properties as the one proposed by Michael Whaler *et al.* in [116]. The model proposed is mainly focused on a model for component replacements. This model may require several iterations to complete the replacement and does not guarantee that the copy of the component state is completed in a fixed number of periods; given that the component is still working and its state is modified. This method also obviates the need to analyze the available slack time to make these component updates. Several techniques aiming at dynamically updating component-oriented embedded system were studied and compared in [111]. Yet, none of the proposed techniques is designed for automotive systems. Adaptation in fault-tolerant distributed embedded systems has been extensively studied in the literature [48] [15] [84] and there have also been efforts on building adaptive real-time fault-tolerant systems. Priya Narasimhan *et al.* proposed MEAD [84], Middleware for Embedded Adaptive Dependability for avionics, provides a proactive fail-over framework using a failure prediction method to overcome the unpredictable nature of failure occurrences and supports somewhat predictable timing behavior. In [15] a Fault-tolerant, Load-aware and Adaptive middlewaRe (FLARe) is designed and implemented to maintain service availability and soft real-time performance in dynamic environments. FLARe supports distributed systems where application servers provide multiple long running services on a cluster of computing nodes. The services in a system are invoked by clients periodically via

remote operation requests. Further, these types of systems experience dynamic workloads when clients start and stop services at runtime. Clients demand both soft real-time performance as well as system availability despite workload fluctuations and processor and process failures. FLARe is targeted at soft real-time applications and does not provide hard guarantees on meeting every deadline.

One last approach to be discussed here is the one presented by Fritsch *et al.* [51]. They propose the TimeAdapt model, which supports runtime adaptation for component-based real-time systems. The model uses an admittance test for reconfiguration, which determines a probability whether a reconfiguration can meet a given time bound. If the probability of meeting the constraints is high enough, the reconfiguration can then be executed. The proposed model is not suitable for distributed embedded systems with mixed hard and soft constraints.

2.5.2 Runtime Adaptation and Mode Change Concept.

Hard real-time systems realize dynamic reconfiguration via mode changes. A system mode change *mostly refers to a switch of defined operating modes of the system, which is generally controlled by a mode change protocol* [34], i.e., when the system's internal state or the environment change all the tasks that belong to the old mode are removed whereas the tasks that belong to the new one are released. During the switching phase (i.e., transition from old to a new mode) the old tasks are still active and the new tasks are scheduled to the system. The mode change protocol is used to guarantee that tasks meet their deadlines [97]. So, we can see that there is a close relationship between mode changes and dynamic reconfiguration. In order to guarantee timing requirements in the presence of mode changes in the system, several approaches have been proposed and focused on mode changes protocols. Sha *et al.* [102] firstly introduces an algorithm allowing timing guarantees of a set of periodic tasks during mode changes in uniprocessor platform [110, 91, 97] (see the survey of [97] for more details). Other efforts have been performed towards extending the above approaches to multiprocessor platforms [114]. The analysis approach in [102] is improved and extended to deadline-monotonic scheduling in [110, 109]. The model is augmented with transition offsets during the execution of new tasks in [90, 91], which makes the schedulability analysis simpler and permits avoiding overload situations. However, a way to calculate such offsets is not provided. Similarly to [90, 91], a slightly different mode change protocol is introduced in [97], in which authors have proposed an asynchronous mode change protocol that permits to tasks from both new and old modes to run concurrently, in order to study the impact of introducing an offset during the mode change on the system schedulability analysis. Abeni and Butazzo [40, 4] have proposed Constant bandwidth server (CBS) and elastic model which allow the system to adapt its resources at runtime however their results were used just for

soft real-time systems. If we confine our interest to mode change protocols with EDF scheduling, they have been only a few studies (three for e.g., [87], [105], [114]) that support dynamic priority preemptive scheduling.

We can see that all the above-referenced approaches are limited to strictly periodic activation and the main limitation of their mentioned analysis is the restriction to static scheduling policies.

Mixed criticality Systems (**MCS**) scheduling was first presented in [113]. A recent survey about this field can be found in [36]. Vestal *et al.* [113, 17] was the first introduces an algorithm allowing all the tasks with different criticality levels to remain schedulable regardless of the changes of the system mode. In MCS, *a system mode change refers to a change of the criticality level of the system for example from lower to higher criticality.* The major limitation of Vestal's model [17] is that the WCET of all tasks must always be known for all criticality level, which is not always possible in practice.

Based on this, in 2013 researchers started looking for solutions to improve the scheduling efficiency of MCS, therefore, the new topic about Adaptive mixed criticality systems has been explored. Building further on this analysis, a variety of algorithms such as AMC [35], and EDF-VD [18] were developed. Burah *et al.* [19] developed Adaptive Mixed Criticality algorithm (AMC). This new algorithm was shown to dominate all the fixed priority preemptive scheduling algorithms for MCS.

Recently, in 2014, Burns and Davis [35] showed how AMC could use final-non preemptive region [44], their evaluation shows that this scheme improves the AMC in terms of schedulability analysis. However, one of the major issues hindering the real-world application of AMC is that most of them supported two criticality levels: *LO-criticality level* and *HI-criticality level* (with *HI-criticality level* is higher than *LO-criticality level*) and always *LO-criticality* tasks may be terminated in order to ensure the execution of *HI-criticality* tasks when the criticality level of the system increase. For dynamic scheduling, Hang *et al.* [107] and Mathieu *et al.* [79] addressed the issue of abandoning *LO-criticality* by using an elastic critical task model.[107] introduced a mechanism called Early-Release EDF (ER-EDF) which allows *LO-criticality* tasks to be released early on the slack generated by *HI-criticality* tasks, whereas [79], they used elastic task model by *stretching* a *LO-criticality* task's period in order to decrease the load on the system in *HI-criticality* mode. Previous methods aimed at allowing *LO-criticality* tasks to execute after a criticality mode change have mostly been best effort.

Discussion: In Table 2.1, we present our survey concerning the recent efforts for the design and development of adaptive real-time embedded systems. This classification is based on the requirements introduced in Chapter 1 Section 1.1. Regarding Table 2.1, for each *Publication*, we provide *the type of adaptation* (e.g., structural), the system's *timing constraint* that has been satisfied (e.g., soft real-time constraints) and the *scheduling*

method that has been used to schedule the adaptation (e.g., fixed priority scheduling). As we can conclude from our survey, none of the above mention-systems provides a flexible adaptation techniques for application with both hard and soft timing constraints. In addition, most of them use a fixed preemptive scheduling policy. Consequently, the solution we devise throughout this thesis is especially directed for handling adaptation for mixed hard and soft real-time applications using dynamic preemptive scheduling policy.

Note that, the mode change handling in the above approaches from the real-time literature is rather complementary to our work.

2.6 Summary

In this chapter, we have given an introduction to the basic requirements of real-time systems and challenges during their development. A classification into soft and hard real-time systems has been given, as well as the concept of self-adaptive systems has been introduced. In addition, A survey about the existing adaptive real-time system approaches was given.

Before giving a detailed overview of our approach in Chapters 4 and 5, we give a more detailed description of the specific needs of automotive systems, and its current state. The next chapter presents these needs and outlines recent architectural changes for automotive systems, which our approach takes care of.

Runtime adaptation of embedded software			
Publication	**type of adaptation**	**real-time requirement**	**scheduling mechanism**
[Valls et al., 13, 14]	replacement	soft	fixed
[Lardieri et al.,07]	upgrade	hard	fixed
[Rasche et al., 05, 08]	replacement	soft	fixed
[Wahler et al., 11]	replacement	hard	fixed
[Narasimhan et al., 05]	replication	soft	fixed
[Balasubramanian et al., 09, 13]	replication	soft	fixed
[Kim et al., 12, 13, 14]	replication	mixed	fixed
[Fritsch et al., 08]	upgrade, replacement, deletion	soft	fixed
Runtime adaptation and mode change			
Mode change approaches [from 89..16]	User-defined parameters	soft,hard or mixed	(80%) fixed or (20%)dynamic

Table 2.1 Comparison of different approaches to add runtime adaptation in real-time embedded systems.

Chapter 3

Automotive Embedded Systems: Focus on AUTOSAR

In Chapter 2, we have introduced the general concepts and foundations of embedded real-time systems. Automotive systems clearly fall in that domain, but compared to airplanes or other plant-automation systems cars have evolved differently over the years. The automotive domain faces more stringent constraints regarding cost and weight, and is affected by a different division of labor between OEMs and suppliers. A brief description of the architecture of automotive embedded systems is given in Section 3.1. The AUTOSAR standard is explained in Section 3.2. In Section 3.3, we introduce the AUTOSAR methodology for developing software for ECUs. Section 3.4 addresses some of the AUTOSAR issues to support runtime adaptation. Section 3.5 positions our contributions with respect to the related state of the art. Towards the objective of introducing an adaptive automotive software architecture. Afterwards, in Section 3.6, we give a brief overview of the current AUTOSAR status. Finally, a summary of the chapter is given in Section 3.7.

3.1 Architecture of Embedded Automotive Systems

3.1.1 Concepts of E/E Architecture:

The E/E architecture in vehicles includes sensors, actuators, and control units as well as other hardware components. The architecture does not specify the details about each sensor, but more that there should be a sensor measuring the distance to objects in front of the vehicle, i.e., if it is a radar, high-speed camera or a laser range finder is not part of the architecture.

Definition 9 ***Architecture:*** *This term means the fundamental architecture of the system embodied in its components, their relationships to each other and the environment and the principles guiding its design and evolution. The term architecture in automotive systems denotes the description of automotive electronic systems at different abstraction levels.*
Topology: *The term topology in automotive systems means the interconnection of the ECUs (nodes) in a vehicle with various communication networks and the E/E architecture of the ECUs, including the hardware and the software architectures.*

Further, the physical network, software, and wiring is part of the E/E architecture. A reason for this is the tight coupling between hardware and software. Before going into detail about the automotive electronic architectures, a few technical terms will be defined (see Definition 9): An automotive electronic system architecture can be described in many ways using different views as stipulated in [85]. Generally, there are two views: (1) the hardware architecture and (2) the software architecture.

1. **Hardware architecture:** The hardware architecture addresses the following elements:

   - **ECU hardware architecture:** This includes the physical components such as processor, sensor, actuator, RAM/ROM, internal communication, and bus connection. Figure 3.1 shows the hardware components of an ECU.

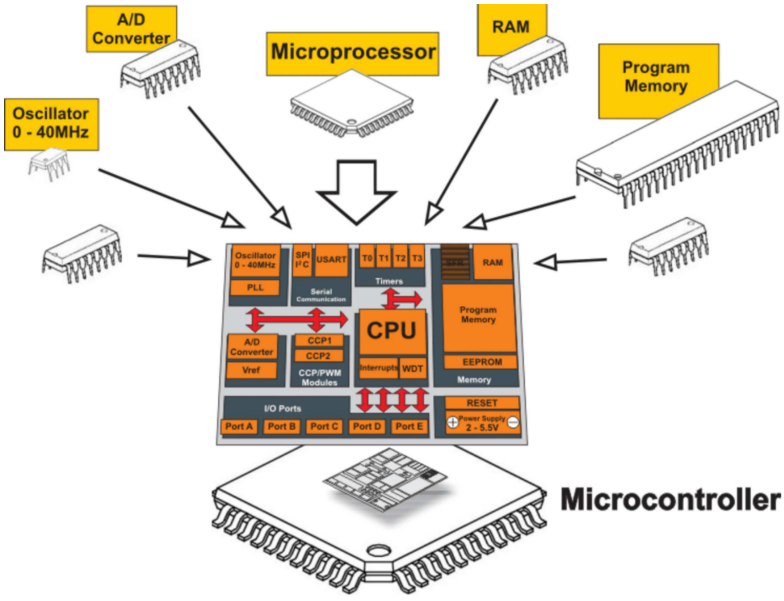

Fig. 3.1 Hardware component of an ECU.

   - **In-vehicle network topology:** This shows where the different ECUs are physically placed and also shows how they are connected in the vehicle. The topology

includes: (i) the number and types of ECUs, (ii) the number and types of communications networks such as Control Area Network(CAN) [45], FLEXRAY [80], etc, and (iii) how the ECUs are connected to the communications networks. In luxury cars such as Mercedes E-Class, the system topology of the communication is very complex with more than 60 ECUs as shown in Figure 3.2. The ECUs are divided in this case into fours vehicle domains: Chassis, cabin/body, power train and telematics, where the power train and the chassis are usually combined together. These ECUs are connected by various communication networks.

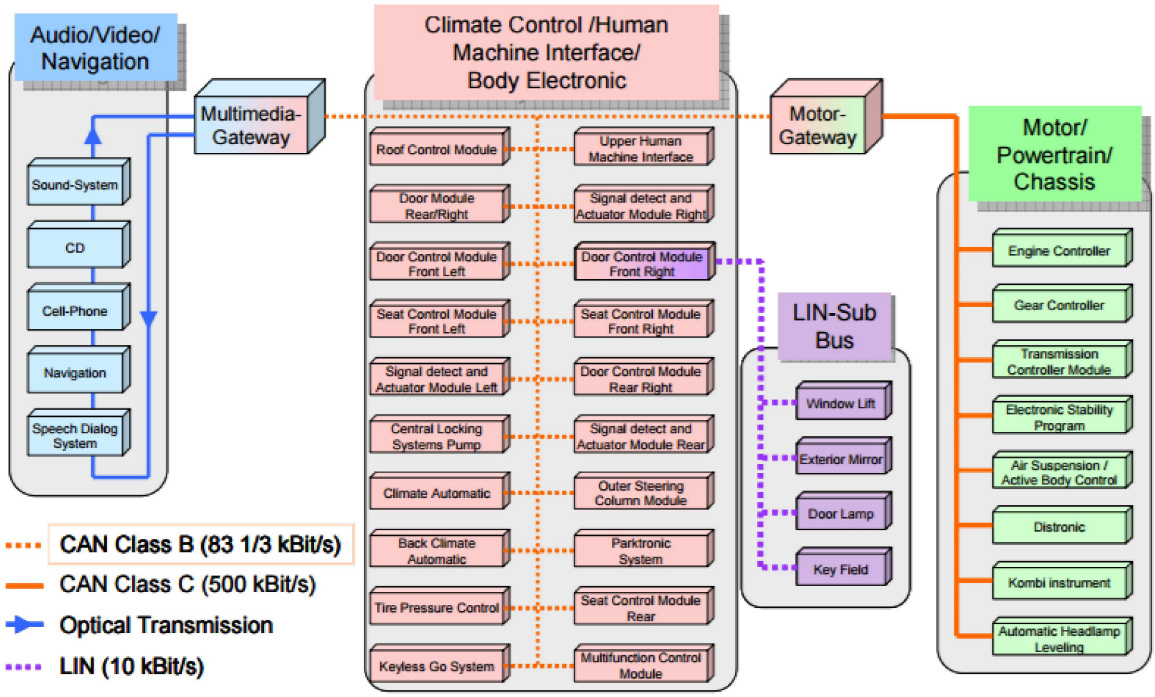

Fig. 3.2 ECU topology of Mercedes E-Class BR211 [85].

- **Communication system and protocol between the ECUs:** A widely used automotive communication network is the Controller Area Network (or CAN referred to CAN bus) [45]. The CAN is an event-triggered communication network with a transfer rate from approximately 20 Kbit/s to a maximum 1 Kbit/s. Bus access for CAN follows the arbitration concept based on the priority of each CAN message. This means that when one or more than one device wants to access the bus simultaneously, only the message with the highest priority can be sent. The other messages with low priority will be discarded and sent at a later time. Thus, the time period to access the bus is not deterministic because the messages with lower priority are discarded and other CAN messages can dominate the bus.

2. **Software architecture**: This is a description that includes the specification of a standard software components. It defines the components of a software system and how they use each other's functionality and data.

3.1.2 Characteristics of Automotive Systems

3.1.2.1 Complexity

The complexity of automotive systems is caused by two distribution aspects: first, the computing platform is composed of a huge number of ECUs or computing nodes. Second, the application software executed on an automotive platform is distributed over its hardware platform. This distribution requires reliable communication and compatible interfaces of the involved tasks for providing the desired functionality. At the same time, hardware and software distribution must not affect the timing requirements of the application in a negative way, causing the application to miss its deadlines. Traditionally, one of the reasons for the highly distributed nature of automotive architecture is *the supply chain*. In other words the relationship between automotive OEMs and their suppliers; in which, the system supplier provides the OEM with the system components ready for direct installation [70, 69].

3.1.2.2 Safety

With the large amount of interaction between the different functionalities in today's car, ensuring that the entire vehicular system operates safely is very demanding. Compared to other embedded systems that are facing a high demand of safety such as avionics; automotive softwares were limited to rather non-critical applications such as infotainment systems. As a result, neither safety considerations were applied nor systematic development processes are required by law [93]. Even today, standards like ISO26262 which are intended to enhance safety of an automotive system are optional. The use of such a standard still does not necessarily result in a safe system. Instead, the explicit use of methods for safety analysis is required [106, 115]. In the next section, we cite some general hardware/ software safety requirements according to the safety norm ISO26262 [46]:

• Reliability and restlessness in such a way that fault tolerance mechanisms can handle detected faults locally without propagation to the other software components in the system.

• Monitoring mechanisms dedicated to indicate the internal and the external failures to the driver.

• Certification and test of automotive software according to a safety norm is an important issue, especially, as the development time for of automotive software is becoming ever shorter.

3.1.2.3 Cost Constraints

The automotive domain is very cost sensitive compared with other domains. A car manufacturer produces 50.000 to 1 million entities of a model [61]. Even adding a small amount of memory or using a faster CPUs in one of the ECUs may add up to a large amount of money [71]. Therefore, automotive systems face more narrow resource constraints than others embedded software domains. Even if automotive engineers try to reduce this cost by reducing the amount of required memory and processing speed is not without problem [31]. As a result, the code needs to be squeezed for the given ECU. However adding a new functionality is made more difficult or even impossible if memory size of the ECUs was optimized too much during the development process [94]. In addition, it becomes harder to identify the error and fix bugs in case of faults [31]. The automotive industry needs new techniques that take into account rising development costs, maintenance cost, project risk, and time-to-market.

3.2 The AUTomotive Open System ARchitecture

AUTOSAR (AUTomotive Open System ARchitecture) defines an architecture for the development of automotive systems [11]. AUTOSAR was launched in 2003 and jointly developed by BMW, Bosch, Continental, Daimler Chrysler, Volkswagen and Siemens VDO. The goal was to create a standard for automotive E/E (Electronics/Electrics) architectures and to decouple application software from lower level Basic SoftWare (BSW) by means of a standardized middleware called RunTime Environment (RTE). This allows running the same application software seamlessly on different hardware platforms, as long as the underlying hardware is linked with the RTE through appropriate BSW. This standard enables the use of a component-based software design model for the design of a vehicular system. It defines as well a methodology that can be used to create the E/E system architecture starting from the design-model (see Figure 3.7). Basically, AUTOSAR-compliant automotive software can be divided into three layers: (i) the application software, (ii) the runtime environment (RTE) and (iii) the basic software layer (see Figure 3.3). The software developers focus on the implementation of application software components (SWCs), whereas the RTE and necessary basic software components such as the operating system, etc. are generated and configured according to the services required by the applications. Details about each layer are given in the next section.

3.2.1 AUTOSAR SoftWare Component (SWC) and Runnables

The AUTOSAR Software is realized by several Software Components (SWCs), which provide the functional behavior of the system. The communication between SWCs and

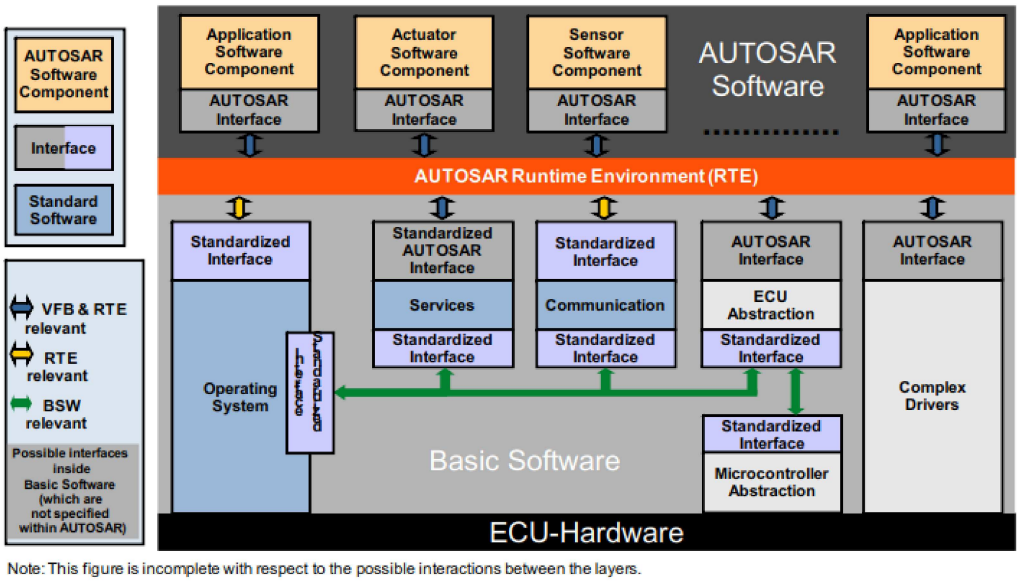

Fig. 3.3 The layered software architecture of AUTOSAR [11].

other components or with parts of the BSW is done through the RTE. AUTOSAR distinguishes two kinds of software components:

- **Sensor/Actuator Software Component.** This component depends on sensors and actuators which are available at the ECU. Due to performance reasons, such a component runs on the ECU to which the sensors and actuators are physically connected. Besides, it is as independent as an AUTOSAR Software Component.

- **AUTOSAR Software Component.** A SWC corresponds to application functions. It consists of entities called *Runnables*. Each application in AUTOSAR consists of one or more SWCs. These SWCs transmit information (i.e., Data Types) to other SWCs using defined interfaces (i.e., Ports) and communicate with each other using the RTE. Note that the application function can either be fitted in a single SWC or spread out in multiple components. Apart from the component implementation, AUTOSAR SWC came with its formal software description, which includes among other: General characteristics: (Name, manufacturers), Communication properties: (Pport, Rport, Interfaces), Composition: (sub-components, connections) and the required HW resources: (scheduling, memory, processing time, etc).

3.2.1.1 Runnable

Since an AUTOSAR SWC is not allowed to access directly the underlying HW or OS, its implementation cannot reflect artifacts like threads or process [1]. Furthermore, each

functionality that needs to be executed within the SWC is covered into Runnable entity or for short *Runnable* (see Definition 10). Since the unit of execution in AUTOSAR is an OS task, all runnables need to be mapped to such an OS task to be executed within the underlying OS task (i.e., an OS task acts as a container of one, or sequence of, runnable(s)).

Definition 10 . *Runnables:* *Runnables are a schedulable units of an SWC, which are basically a sequence of instructions (C-functions) that can be started by the RTE as a result of an event initiated by the RTE and are executed in the scope of an OS task. An SWC consists of at least one runnable [86].*

Depending on whether runnables have an internal wait-points; two main runnable categories exist. That are then allocated on different kind of operating system tasks (see Section 3.2.3.4):

- Cat1. Runnables are without wait-points (i.e., non-blocking), in other words, the runnables terminate in a finite time. Within this category, there are two subcategories: 1A which is only allowed to use implicitly defined APIs and 1B which is an extension of 1A making it also to use of explicit APIs. Runnables that fulfill these constraints are usually assigned to *Basic tasks* of the operating system (i.e which are not allowed to invoke any blocking system calls).

- Cat2. Runnables have at least one waiting point. Runnables that fulfill these constraints are usually assigned to *Extended tasks* (i.e., which are allowed to use blocking system calls).

Runnables activation. All runnables are activated by the RTE as a result of an RTEEvent. The RTEEvent specifies how and when the runnables should be invoked /or activated at runtime. AUTOSAR defined sevens RTEEvents that trigger the execution of the runnables, which is done either by activating or waking up them [1]. The following Table 3.1 names the most important RTEEvents that are defined by AUTOSAR.

3.2.1.2 Ports

Are the mechanism of SWCs to communicate between each other. There are two types of ports in AUTOSAR SWCs: a *PPort* or an *RPort*. A *PPort* provides an AUTOSAR Interface while an *RPort* requires one. If the interface that is encapsulated by the port is provided a module or service of the AUTOSAR Service layer then it has to be a *Standardized AUTOSAR interface*. Whereas, if it is provided by an AUTOSAR SWC or module of the I/O HW layer then this interface has to be an *AUTOSAR Interface*. This means it is used by the layer above the RTE as well as by ECUAL and CDD which

Table 3.1 Overview of RTEEvents.

RTEEvent Type	**Communication category**	**Description**
TimingEvent		Triggers a runnable periodically.
DataReceiveEvent	S/R	triggers a runnable upon reception of new data.
OperationInvoke Event	C/S	Triggers a runnable when a client wants to use one of its services provided on a PPort
Asynchronous- SeverCall	C/S	Triggers a runnable upon the return of an asynchronous call .

are below the RTE. A port always belongs to exactly one component and represents a point of interaction between a component and other components. This distinction describes the direction of the communication. An *AUTOSAR standardized Interfaces* or an *AUTOSAR interfaces* can either be a *Client-Server Interface* (C/S) or a *Sender-Receiver Interface*(S/R). Figure 3.4 shows C/S and S/R interfaces with their symbols.

- **C/S:** defining a set of operations that can be invoked, either synchronously or asynchronously.
- **S/R:** which allows only the usage asynchronous communication (i.e. all calls are non-blocking). The direction of the communication is given through the arrow.

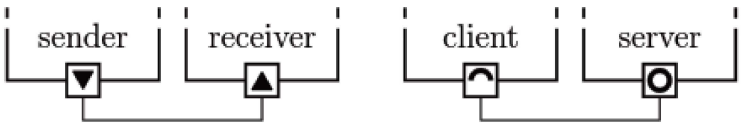

Fig. 3.4 Symbols of ports.

3.2.1.3 Inter Runnable Variable (IRV)

Ports define the interaction between SWCs, or more precise the interaction between the runnables of the SWCs. For runnables of one SWC another mechanism for internal communication exists. These are the Inter Runnable Variables (IRVs), which can be accessed by the runnables. Runnables could also use global variables for this internal communication, but with IRVs protection mechanism are provided by the RTE for concurrency.

3.2.2 The RunTime Environment (RTE) and Virtual Function Bus (VFB)

The RTE provides communication services for the application software (AUTOSAR SWCs and Sensor/Actuator). It implements the data exchange and controls the interaction between the application software components and the Basic Software Layer. Precisely the RTE is an implementation of the Virtual Function Bus (VFB), which is an abstract communication environment for connecting SWCs. Having a VFB to exchange data between the SWCs enables them to be independent from the underlying hardware platform.

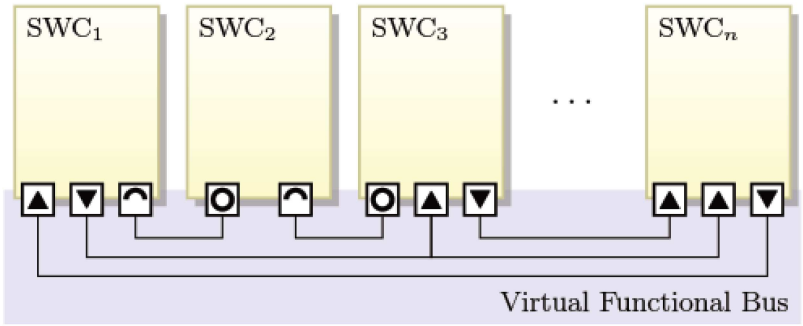

Fig. 3.5 Virtual function bus.

The interface defines the ports that can be used from the functions in the component. At the VFB level, these ports can be used to connect components by connecting compatible ports as it is shown in Figure 3.5. This represents the functional part of the system, independent from the architecture or infrastructure which is needed to execute this system.

3.2.3 The Basic SoftWare (BSW)

The Basic SoftWare (BSW for short) consists of four sub-layers presented from down to up as follows:

3.2.3.1 The ECU Abstraction Layer (ECUAL)

This layer offers access to all of an ECU functionalities such as I/O(Input/Output), communication and memory–regardless whether these functionalities are part of the microcontroller or are implemented by peripheral components.

3.2.3.2 The Complex Device Driver (CDD)

The CDD contains the drivers for the specific properties of a microcontroller or ECU, which are not standardized in AUTOSAR, so this layer provides non-AUTOSAR components (e.g., device drivers).

3.2.3.3 The Microcontroller Abstraction Layer (MCAL)

This is the lowest layer of the BSW. It contains hardware-specific drivers for access to memory, communication and I/O of the microcontroller. The purpose of this layer is to isolate higher software layers from the specifics of the microcontroller.

3.2.3.4 The Service Layer

This is the highest layer of the BSW, that provides different types of background services such as memory management, vehicle network communication services (bus communication), diagnostic services and the ECU mode management. The Operating System is also contained in this layer. In the following, we clarify the most important concepts for our work.

3.2.3.4.1 AUTOSAR Current Task Model.

The AUTOSAR Operating System (AUTOSAR-OS) is a real-time OS associated with the AUTOSAR standard. AUTOSAR scheduling tables are inherited and based on the same mechanism in the OSEK/VDX operating system [88]. These tables are executed periodically or once and they are an encapsulation of a statically defined set of expiry points, the later are offsets at which the OS activates tasks and/or set of events [2]. In other words, it encapsulates a set of points at which certain statically configured actions will be executed. Most of the general properties and attributes of OSEK-OS apply to AUTOSAR-OS as well. In particular, the main characteristics provided by OSEK-OS to AUTOSAR: *statically configured (i.e., the overall system configuration is known at compile-time)*, *fixed priority scheduling*, *interrupts handling* and some *protection mechanisms* against unintended use of the OS services.

AUTOSAR-OS differentiates between two kinds of tasks: Basic tasks and Extended tasks.

- Basic task: Basic tasks release the processor, only if they terminate, the scheduler switches to a task with a higher priority or an interrupt arises and an interrupt service routine is called.

- Extended task: The extended tasks provide in addition to the basic tasks the possibility to release the processor without termination and wait for an event. After the event occurs, the extended task can be waked up by the scheduler again. This is called a *wait-point* and can be used to synchronize tasks. Basic tasks can just be synchronized on the start or the end. Figure. 3.6 depicts the state transition model for extended tasks as defined in OSEK-OS [88] and AUTOSAR-OS [2], where each task is in one of the four states: *Ready*, *Suspend*, *Run* or *Wait*. The different transitions between these states are described in Table 3.2.

Table 3.2 States and status transitions for extended tasks defined by OSEK & AUTOSAR-OS.

Transition	**Initial state**	**target state**	**Details**
Activate	suspended	ready	A new tasks set transits into ready state by a system service which is function called from alarms, tasks or ISRs (i.e., it's used to set or clear the events).
start	ready	running	A ready task is selected by the scheduler to be executed.
wait	running	waiting	The transition into the waiting state is caused by a system service e.g., WaitEvent. The waiting task requires an event to continue its operation.
release	waiting	ready	At least one event has occurred for which a task has been waiting.
preempted	running	ready	The scheduler decides to start another task,the running task is put into the ready state.
terminate	running	suspended	The running task causes its transition into the suspended state by a system service.Task termination is only possible if the task terminates itself.

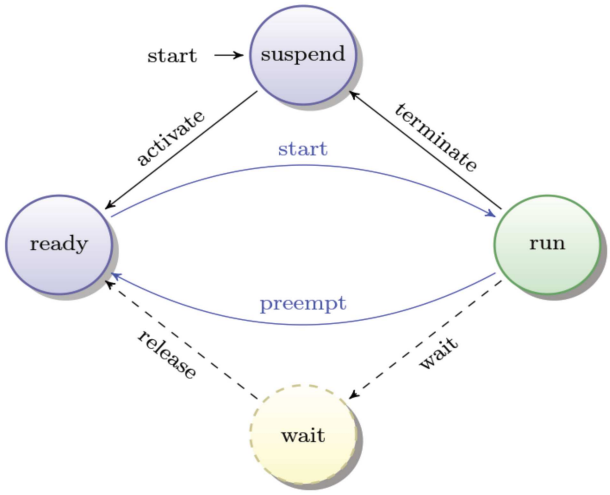

Fig. 3.6 AUTOSAR extended tasks state mode.

When an AUTOSAR application is defined, it must be loaded into an embedded target. For this, it is necessary to take into account the execution medium. This execution medium has two aspects: on the one hand, the tasks that execute the application software, and On the other hand, the intrinsic characteristics related to the target, namely the memory space used as well as the total CPU utilization. Table 3.3 depicts the characteristics of AUTOSAR tasks.

Table 3.3 Characteristics of AUTOSAR-OS tasks.

Concept	**Characteristics**	**Details**
Type	Basic / Extended	A basic task cannot wait for an event, only extended tasks can.
Activation Mode	Periodic or Event-triggered	Periodic tasks are triggered by alarms and sporadic tasks by events.
Preemptive	Full-preemptive Non-preemptive	Suspended if a higher-priority tasks is ready or by an interrupt. Only suspended by an interrupt.
Priority		A higher number means higher priority.
Trigger	Alarm Events/ISRs	Periodic Tasks Sporadic Tasks
Activation mode	Periodic / Sporadic	

3.2.3.4.2 AUTOSAR Mode Management

AUTOSAR has a basic concept for performing adaptation via the notion of mode [12] (i.e., its only consider static adaptation with a fixed set of modes). The adaptation can be for example achieved by changing the BSW components, the communication network or the RTE configuration at runtime. Specifically, AUTOSAR defines a *mode manager*, that is responsible for the state of the vehicle's system, from the lowest level (i.e., individual ECUs) to the highest level (i.e., the whole vehicle). The mode manager consists of two parts: *Mode Arbitration* and *Mode Control*, and it can be either SWC or a BSW module. The idea behind, is that upon receiving an initial request to change between the predefined mode for e.g., $M1$ to $M2$, this request will be distributed by the RTE to the locally defined *Basic SoftWare Mode Manager* (BSWM) in each ECU. Then, the BSWM propagates the received requests, evaluates its corresponding rules and runs the corresponding actions (this is the role of the mode arbitration and mode control in BSWM). Finally, the BSWM sends an RTE call to the mode switch API to transmit the resulting mode i.e., $M2$ to the *Mode User* in order to read the currently running mode.

3.2.3.4.3 AUTOSAR Fault Management

Faults and errors management in AUTOSAR are enabled through the *Diagnostic Event Manager*(DEM) service [11]. The DEM is a basic software module of the diagnostic Services. Relevant errors are reported either from application layer (resp. SWC) or basic Software modules. The DEM handles and stores events detected by *diagnostic monitors* in both SWC and BSW levels (see Definition 11). The stored event information is available via an interface to other BSW modules or SWCs. Each *monitoring path* is associated with exactly one *diagnostic event*. A diagnostic event is the status (pass/fail) of a system under test for a set of test conditions. If an event gets qualified as failed, it becomes active. If the event gets qualified as passed, it becomes passive. The DEM is also responsible for event qualification, confirmation and memory overflow indication.

Definition 11 . ***Diagnostic Event Manager:*** *A component that processes and stores Diagnostic Events (errors) and associated data. A Diagnostic Event defines the atomic unit that can be handled by the DEM module [12].*
Diagnostic monitor: *A diagnostic monitor is is a routine entity determining the proper functionality of a component by identifying a specific fault type (e.g., short to ground, open load, etc.) for a monitoring path.*
Monitoring path: *A monitoring path represents the physical system or a circuit that is being monitored.*

3.3 AUTOSAR Development Methodology

After the architecture is clarified, this section takes a look at the approach of AUTOSAR for developing software for ECUs. In AUTOSAR, this is done in multiple steps. The methodology [13] shown in Figure 3.7 handles the whole view of the complete system with multiple ECUs and is used to create the E/E system architecture starting from the design model to the generation to an ECU executable. This methodology consists of four steps:

The first step, "The system configuration", concerns the whole system. During this step the entire set of applications is specified in terms of software architecture : SWCs, ports, interfaces and connectors and their distribution to the ECUs. The system topology is also specified in term of: ECUs interconnection, the used protocol and the available data buses, the communication matrix and attributes (e.g., timing/latency, etc.). The hardware such as sensors, actuators and processors needs to be defined in this step with the ECUs resources. The output of this step is a System Description– an AUTOSAR XML file, which serves as input for the following phase.

The second step, "Extract ECU-Specific Information", concerns the SWCs implementation (i.e., the definition of the internal behavior the Runnables and the RTEEvents). During this phase the allocation of SWCs to the hardware architecture is specified and the application signals are mapped to bus frames. As a result of this step, we obtain an ECU extract of system Configuration or Extract of System Configuration Description– an AUTOSAR XML file, which serves as input for the following phase.

The third step called "ECU Configuration" concerns the configuration of the BSW modules and RTE of each ECU. The most important step adds all the implementation concepts, including the scheduling of tasks, mapping of runnables to OS tasks, the required BSW modules, etc. As a result of this step, an ECU Configuration Description is generated, which is aligned with System description and the ECU Extract.

The fourth and last step is "Generation of Software Executable". It concerns the Build executable based on the ECU description in "ECU Configuration Description", it involves also the code generation of the RTE, BSW, the compilation of SWC available as source code, the generated code and linking everything together.

In Appendix B.0.1 , we give a short example to clarify the concepts described in Chapter 3. The example presents the complexity of the XML configuration which is already reached by such a simple case. But it should help to understand, how the mechanisms described in this chapter are implemented in the source code.

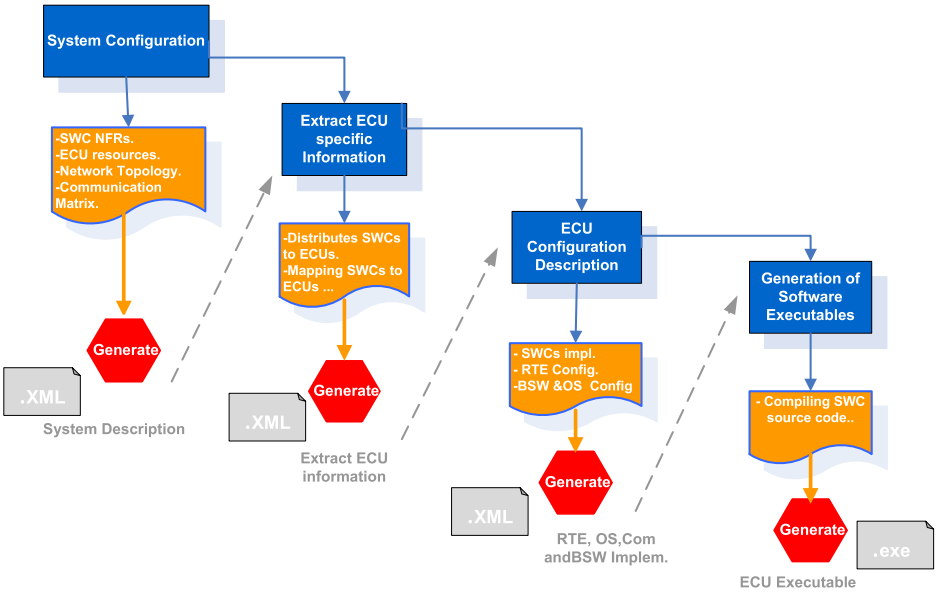

Fig. 3.7 AUTOSAR Development Methodology.

3.4 AUTOSAR Open Issues to Support Runtime Adaptation

A major drawback of AUTOSAR lies in its lack of flexibility. Software-wise, ECUs are tested, validated and uploaded; these three steps are performed in a monolithic process. Adding *adaptability* features into the standard is a major challenge in this context, and may result in consequent savings of time and money. In the next, we describe the major drawbacks with AUTOSAR current task model, fault and resource management, runtime environment and the mode manager.

I_1) ***Task Model:*** The AUTOSAR OS specification states that it must be run on a real-time operating system that can be configured statically and that is amenable to reasoning of real-time performance. The specification expects the underlying OS to control three factors (the execution time of tasks/ISRs, blocking time that tasks suffer from lower priority tasks locking shared resources and the inter-arrival rate of tasks/ISRs) at runtime to ensure timing protection. However, AUTOSAR describes no methodology for analyzing whether a system design will meet its real-time requirements. The timing extensions to AUTOSAR are relatively recent additions, and thus manufacturers previously had to use proprietary timing specifications and internal tools to perform automated reasoning about the timing of their specifications. Even with the standardized timing extensions, manufacturers are on their own to devise methods and tooling for checking whether an integrated system is schedulable, which is a highly non-trivial task. In addition, the concept of scheduling table definitely needs to be enhanced. Because if we change the

statically defined table to run and plan a new allocation at runtime, it may lead to a big overhead and there is a need for a complex schedulability analysis [121], [2] to guarantee correct system behavior in the new configuration. Also, AUTOSAR scheduling algorithm doesn't assign tasks dynamically to CPUs which limits the adaptation capabilities.

For these reasons, we believe that the current AUTOSAR-OS is not in a position to support adaptation at runtime. Hence, changing its current status that assigns static priority to tasks at design time to an adaptive/dynamic preemptive(a-)periodic using EDF (Earliest Deadline First) and CBS (Constant Bandwidth server) as scheduling mechanisms to guarantee real-time requirements for mixed hard/soft real-time systems work efficiently for adaptive real-time applications [18] [33] [38].

I_2) ***Fault Model:*** AUTOSAR does not specify a fault model, nor does it describe a specific mechanism or methodology for dealing with faults that may occur in the system. Such fault handling is high-level functionality must be built on-top of the basic functionalities that are specified by AUTOSAR. Kim *et al.* [66] describes one attempt at extending the AUTOSAR specification with fault tolerance provisions. This work presents the standard fault-tolerance concept of replication to provide duplicate copies of software components that are used when the primary copy of a software component fails. As we showed in the previous section, AUTOSAR uses the concept of *diagnostic monitors* to monitor specific physical systems. If a fault is detected, the diagnostic monitor logs a diagnostic event with the diagnostic event manager. As we can see AUTOSAR lacks complex fault detection mechanisms as well as **temporal isolation** strategies.

I_3) ***QoS support and resource management:*** AUTOSAR does not offer extra-functional properties support, i.e., **QoS model**. Ensuring application's meeting their expected level of performance requires a priori knowledge at design time about the system requirements. AUTOSAR application requirements are always statically satisfied. Thus, no way to describe resource levels and requirements that change over time in AUTOSAR. Instead, the system integrator is in charge of knowing all the requirements at design time and provision the system accordingly.

I_4) ***RunTime Environment (RTE):*** The Run Time Environment (RTE) within AUTOSAR is partly configured and partly generated based on the resource needs of the applications running on top of it. This approach allows the RTE to be optimized for a specific set of applications and ensures that the runtime system is neither under nor over-provisioned. The allocation of resources to software components in AUTOSAR is static, with many resources such as mutexes and memory being allocated once and never changing.

I_5) ***Mode Manager:*** With the actual mode manger, all possible configurations at runtime must be anticipated at design time and thus non-anticipated configurations are not supported. Another point is that neither the insertion of new tasks at runtime nor

changing the deployment of a SWC have so far been considered in AUTOSAR mode management. What we can conclude, that AUTOSAR mode manager lacks flexibility. Hence, the arbitration and control must then be extended to include other functionalities while running in dynamic environments. It seems natural to extend its responsibilities in order to deal for example with the case of dynamic linking or introducing a new software components in the existing system. Another remark that there is no information in [2] about the schedulability analysis behind the mode protocol used in AUTOSAR. All these drawbacks limit significantly the adaptation in AUTOSAR to anticipated adaptation.

Discussion. The ability to dynamically adapt the system, such as adding a new application or moving an existing application to a different ECU, is not possible with AUTOSAR. In AUTOSAR, the system configuration is, by design static: the AUTOSAR OS specification states that the underlying operating system must be able to be configured statically, such as the number of tasks and the number of resources. The AUTOSAR Run-Time Environment (RTE), which sits on top of the BSW but below the application level Software Components (SWCs), is not only configured at design time for specific ECUs but is also partly generated based on the requirements of the SWCs that will be running on it. A reconfiguration of the system, such as adding an application or moving an application from one ECU to another, cannot be done dynamically at runtime. On the other hand, the solution provided in this thesis is designed with such reconfiguration capabilities in mind. In fact, it is expected that applications may need to migrate, at runtime between ECUs in response to both anticipated changes caused by the environment, such as network connectivity, as well as unexpected failures in both software and hardware.

3.5 Runtime Adaptation in Automotive Systems

In recent years there have been a number of publications discussing the issue of creating runtime adaptive ECUs complying to the AUTOSAR standard. These approaches can be divided into two different groups targeting on two different scenarios. One group focuses on the switching between different configurations of SWCs at runtime. The second one actually tries to add or remove SWCs from the AUTOSAR ECUs while the vehicle is running.

For the first group which uses configuration switching as a basis in order to change the system, four different approaches have been identified.

The first one is the DySCAS (Dynamically Self-Configuring Automotive Systems) [7] project. The work introduces a new architecture, with focuses on context-aware adaptation mechanism,specified by execution and architecture aware contexts. The former context uses distributed policies to detect deviations and react, while the latter em-

beds meta-information of configuration reasoning (resource dependencies, QoS contracts, compatibility, composability, and dependability) within dynamically reconfigurable components. DySCAS deals with task migrations to cope with hardware failures and network balancing. A global node dynamically maintains the intentions of every node within the network and decides the possible configurations based on their requirements. Each node locally performs admission control deciding if a task is schedulable considering resource limitations (memory, CPU, bandwidth) and optimization of resources. The DySCAS middleware is one of the efforts towards context awareness and self-configuration in telematics domain in automotive systems. The approach is different from ours in some relevant aspects. They do not extend AUTOSAR architecture to support the adaptation, also they target applications with best effort requirements. In our case, we are concerned about runtime adaptation of mixed hard/soft applications within AUTSOAR. Other works such as [120, 29, 22] have studied runtime adaptation in automotive systems.

The authors of [120] (i.e., Zeller *et al.*) introduces requirements that must be considered for supporting the adaptation within AUTOSAR. The reconfiguration in this work aims at switching between different configurations of SWCs. They add two additional SWCs to the system: An Adaptation Service and an Adaptation Manager. These components are in charge of activating and de-activating SWCs installed on the device. The capability to add new SWCs and migrate SWCs between different ECUs at runtime is not considered in their work. Additionally, neither the details about the algorithm used for making adaptation nor the evaluation of their approach was given.

An approach quite similar to the one of Zeller *et al.* is presented by Berger and Tichy [29]. The authors integrated the Operator-Controller-Module (OCM) approach for self-adaptive mechatronic systems to the AUTOSAR architecture. OCM is an architectural model used for example in factory automation to introduce self-x properties. It specifies three different control loops namely a motor loop that actually controls the mechatronic device underneath, a reflective loop that allows to monitor and change the configuration of the entities within the motor loop and finally a cognitive loop that gathers information on the system itself as well as its surroundings to improve the reconfiguration mechanisms using a behavioral approach (see [58]). Berger and Tichy combine this architectural style with AUTOSAR in order to allow runtime adaptation. However, this solution is quite limited since it causes the same problems as the approach of Zeller *et al.* by not adding adaptability to the BSW but integrate all possible configurations at design time instead. Hereby it suffers from the same drawbacks such as overheads in the development process as well as in the final implementation.

The last approach to be named within this group has been presented by Becker *et al.* in [22]. Just like the other publications before it is a technique to switch between pre-defined configurations without adding real flexibility to the BSW. However, this approach goes one

step further by not only defining a configuration for the communication handles within the BSW but for the overall software architecture of the ECU. An additional component called State Manager organizes the switches between these different configurations at runtime and deploys the predefined one for each upcoming situation. Although being integrated into a well defined and AUTOSAR compatible development procedure this approach is even worse in terms of development and implementation efficiency when comparing it to the proposals of Zeller *et al.*, Berger and Tichy. This is because developing a single configuration equals to the development of a complete ECU in the conventional AUTOSAR procedures. Also, none of the mentioned-approaches studied runtime adaptation with taking into account schedulability analysis of mixed hard/soft applications.

The second group of approaches has a different perspective. In order to allow upgrades or extensions to AUTOSAR-based ECUs that are already running within a vehicle, these proposals target on dynamically adding and removing SWCs at runtime. The first one, presented by Zeeb in [119], intends to enable installing software updates at runtime. This is done by reserving memory capacity for future software variants at design time and re-flashing parts of the memory in the event of an update while maintaining the subroutine addresses of the SWCs. This focus on updates sets up two main restrictions. First of all, it is limited to the update of already installed SWCs rather than allowing adding additional ones. Second, since those updated components do use the same interfaces to the RTE and the BSW no changes within these layers are necessary. Hence, no flexibility is added to these modules within this approach of Zeeb. This results in the conclusion that, although this proposal adds some kind of runtime adaptivity to AUTOSAR it does not add any dynamics to the modules underneath the Application Layer.

The main goal of another approach presented by Axelson *et al.* in [14] to plug in new components at runtime. Therefore, the authors suggest adding two SWCs to each AUTOSAR ECU: the first one holds a virtual machine to run Java applications. The second one which is called "external communication manager" establishes a direct connection to an external software source. The approach follows the idea that software components developed in Java are downloaded via the external communication manager and installed into the virtual machine at runtime. However, the paper leaves unclear how the Java components downloaded are interacting with the static RTE and BSW underneath since no solution is presented on how to make the communication stack more flexible. This leads to the assumption that those Java-based software modules can either only access a predefined set of channels to the hardware and communication devices or are not interacting with the remaining parts of the ECU and the overall system accessible through the networks. This does not only reduce the fields of application but also does not provide any answer to the requirements on a dynamic BSW as needed for adaptive automotive system design.

The last approach to be discussed in this group is the one by Martorell *et al.* [81], which provides a way to design and develop a partial dynamic update in AUTOSAR application. Just as Axelson *et al.* the authors suggest adding several empty SWCs to AUTOSAR architecture. These can be filled with runnables at runtime. However, the approach uses a static configuration of the OS level of AUTOSAR, specifically the BSW layer, instead of introducing some kind of flexibility. The update is applied at the application level. This means that all potential hardware handles and communication routes have to be foreseen at design time and that no real flexibility is added to the AUTOSAR low-level layers (i.e., BSW layer). No details are given about the schedulability analysis for this partial update.

Discussion. The discussion of the existing approaches to integrate runtime adaptation into AUTOSAR is summarized in Table 3.4. As shown in this table, the suggestions vary in their type of adaptation supported and the scheduling technique they use. However, to the best of our knowledge, no one has provided a framework to support runtime adaptation while taking into account schedulability analysis and task allocation for mixed hard and soft distributed real-time systems in AUTOSAR.

3.6 Current State of AUTOSAR

Recently in March 2017, the automotive industry agreed upon changing the current AUTOSAR platform to the "AUTOSAR Adaptive Platform", this platform is being developed by the AUTOSAR consortium as an additional product to the current AUTOSAR classic platform. This is an ongoing feasibility study based on the POSIX operating system and uses service-oriented communication to integrate applications into the system at any desired time. The AUTOSAR Adaptive Platform is designed to help engineers create more flexible architectures. AUTOSAR Adaptive will provide a software framework for more complex systems and help engineers increase bandwidth by implementing Ethernet. Most suppliers currently seem to be interested in moving fairly quickly to get the standard completed and implemented. It is expected to become a piece of the in-vehicle infrastructure used on the road to autonomous vehicles. The first parts of AUTOSAR adaptive may appear on highways as early as 2019/2020 [10]. Vector company expects a more general adoption of AUTOSAR adaptive for production by 2022 and late.

3.7 Summary

In this chapter, the specific challenges of automotive systems and their implementation have been introduced. As explained, the number of computing nodes included in an automotive system has grown continuously over the past years. Implementing such a system is a huge challenge regarding the complexity of current automotive software. Even

more, so when taking into account the high-quality demands arising from the safety criticality of the implemented functions. In order to tackle this complexity automotive industry have developed the standard AUTOSAR. In this chapter, we have introduced the basic concepts of this standard more precisely, we have presented functional parts (i.e., SWCs) of AUTOSAR software and infrastructural parts (i.e., BSW), which provides the necessary basis to run the SWCs. In addition, we discussed some of the major drawbacks of AUTOSAR for handling runtime adaptation. Also, the existing approaches for runtime adaptation in AUTOSAR were presented. Finally, we concluded the chapter with the current status of the AUTOSAR standard.

Runtime Adaptation in AUTOSAR			
Publication	**type of adaptation**	**real-time requirement**	**scheduling mechanism**
[Richard Anthony *et al.*, 09]	migration	best effort	fixed
[Zeller *et al.*, 12, 13]	active/deactivate	hard	fixed
[Berger *et al.*, 12]	active/deactivate, deletion	soft	fixed
[Becker *et al.*, 09]	active/deactivate, deletion	hard	fixed
[Axelson *et al.*, 13]	plug an play	soft	fixed
[Zeeb *et al.*, 12]	update,upgrade	soft	fixed
[Martorell *et al.*, 14, 15]	update,upgrade	soft	fixed

Table 3.4 Comparison of different approaches to add runtime adaptation to AUTOSAR.

Chapter 4

Runtime Adaptation Support in Automotive Software Architecture

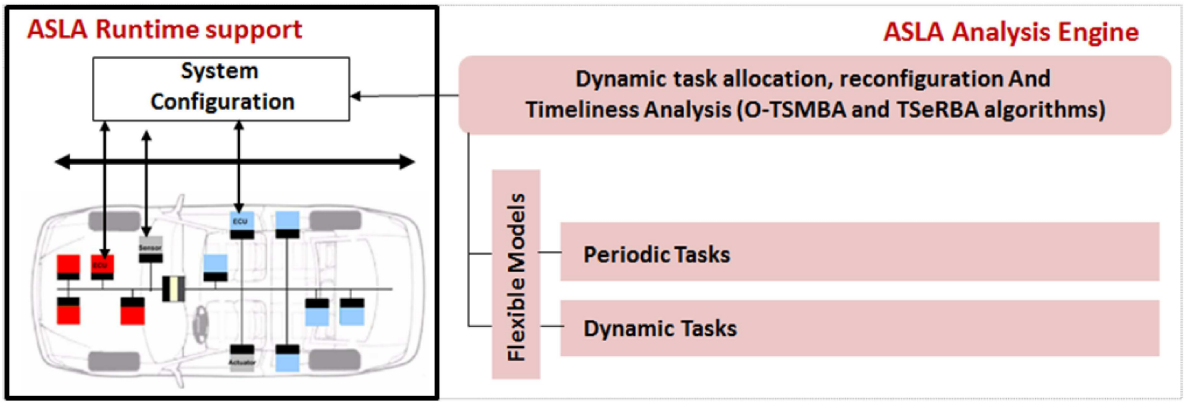

Fig. 4.1 Runtime support for adaptation features in the dissertation overview.

More than ten years have passed since the establishment of the AUTOSAR (Automotive Open System Architecture) standard. AUTOSAR was introduced to simplify automotive system design while offering interoperability, scalability, and extensibility. As we said in Chapter 3, recently in March 2017, the automotive industry agreed upon changing AUTOSAR to making it able to support runtime adaptation; i.e.: changing the system's structure and/or behavior at runtime in response to environmental changes or failures. In the actual version of the standard, the system configuration is still static by design from the application down to the Operating System (OS) layer. A reconfiguration of the system, such as adding an application or moving an application from one Electronic Control Unit (ECU) to another cannot be done dynamically at runtime. Making AUTOSAR adaptive requires specific support at different layers of the software architecture. Therefore, our objective is to elaborate an **architectural solution** that can handle the **adaptation of mixed hard and soft** applications while respecting timing and safety requirements

and offering a high degree of **flexibility**. To address this challenge we describe in this chapter our distributed layer called ASLA (Adaptive System-Level in AUTOSAR) used to incorporate task-level adaptation techniques in AUTOSAR as depicted in Figure 4.1. In this chapter we outline how the AUTOSAR standard can be adaptive by changing its architecture. Section 4.1 gives an overview of the proposed approach (i.e., ASLA). Subsequently, in Section 4.2 we introduce ASLA development methodology. We conclude this chapter in Section 4.3.

4.1 The Architecture of ASLA

We now describe our solution ASLA, which is designed to overcome the limitations of AUTOSAR cited in Chapter 3 Section 3.4. We first detail the responsibilities of the major components of ASLA. Then we describe how these components work together with different runtime algorithms to provide tasks adaptation capabilities in AUTOSAR. Figure 4.2 provides an overview of the ASLA architecture. Every ECU that supports adaptation through the ASLA layer consists of a real-time OS with EDF and CBS scheduling policies, an Adaptive SWC that is responsible for reconfiguring applications running on the system, RTE, and application layers. The real-time OS is responsible for HW abstraction, communication, scheduling and executing tasks in real-time. We assume that the underlying HW is a fail-silent system and the communication network is fault-tolerant. Our application layer consists of a set of SWCs (similar to AUTOSAR's) and a new Adaptive SWC which can be distributed over several ECUs. The RTE provides a communication abstraction to SWCs. Unlike AUTOSAR, our RTE extension contains functions to support adaptation. These functions are managed by the Adaptive SWC (more precisely by the Reconfiguration Manager (RM)) which also communicates with the others Adaptive SWCs running on the different ECUs to make one of the adaptation actions such as: adding, deleting or updating application. When a new application is being added, the mapping between the application's SWCs and the ECUs is given to the RM, then each RM analyzes the mapping and renews the RTE's function.

The ASLA layer is composed of an Adaptive SWC (one on each ECU) and plugin offering a task execution container. This plugin enables any task launched on the ASLA layer to be periodically executed. The adaptive component has a coordination-based architecture i.e., One Adaptive SWC acts as a coordinator of the other Adaptive SWCs which are responsible for handling tasks on each ECU and monitoring a health vector. The latter contains all Non-Functional Requirements (NFRs) needed for the adaptation such as the ECU's processor utilization, resources, QoS, HW NFRs, etc. All operational ECUs compute their resources and processor utilization in form of a health vector at a fixed time period and share their health vector with each other. This provides each ECU

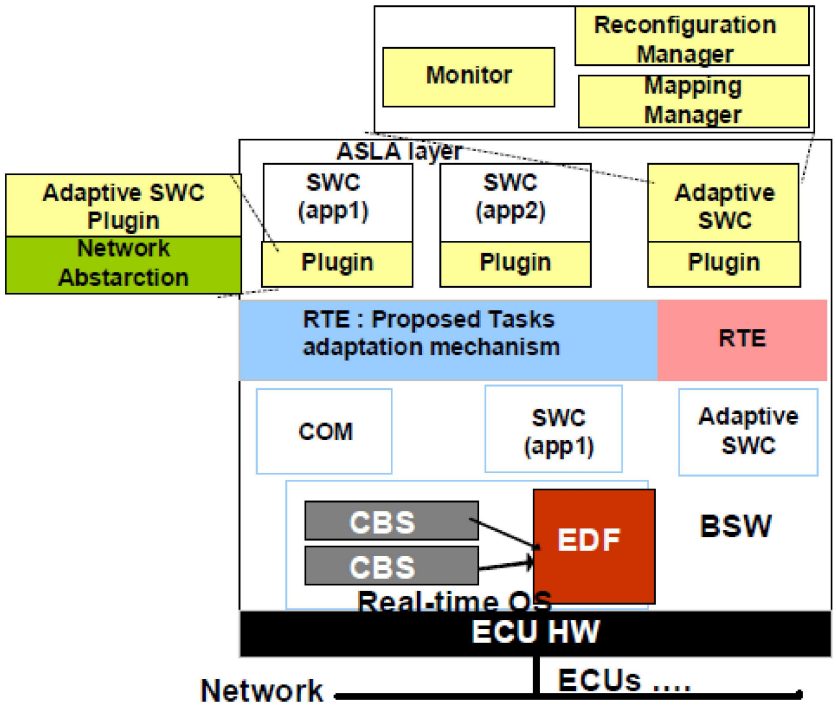

Fig. 4.2 The ASLA architecture.

a consistent view of the available resources and utilization on the other nodes. Figure. 4.3 illustrates a brief overview of our health vector.

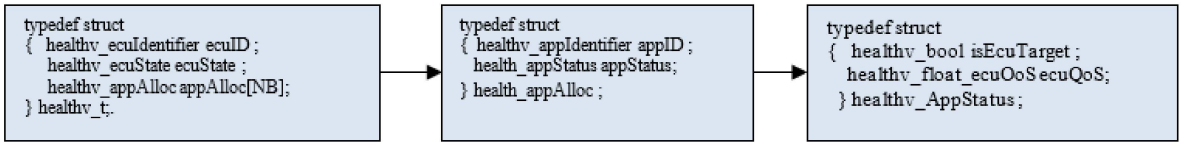

Fig. 4.3 An example of health vector structure.

Since our Adaptive SWC has a coordination based-architecture, we define an adaptation-management protocol between the different Adaptive SWCs inspired by [83]. However, we differ from them in the sense that our protocol is used for managing the process of an adaptation in distributed real-time systems. In our protocol, all the Adaptive SWCs, including the coordinator, broadcast messages to each other. The coordinator can detect the failure of the other Adaptive SWCs by the absence of heartbeat messages (our adaptation-management protocol is described in Section 4.1.2) .

The major components of ASLA are described below.

4.1.1 The ASLA Components

4.1.1.1 The Adaptive SWC

As illustrated in Figure 4.2, an Adaptive SWC is composed of a Monitor, a Mapping Manager (MM) and a Reconfiguration Manager (RM).

• **The Monitor.** The monitor is responsible for monitoring events that trigger the adaptation. Hence, it periodically sends messages to other ECUs in the system via the network. The monitor allows ASLA to agree on the availability of each ECU. Any adaptation trigger received by the application during its execution may invoke the monitor, which sends a message to the RM in order to adapt the application. The loss of a message for two consecutive cycles means that the ECU is no longer alive and the adaptation needs to be triggered to accommodate the desired changes.

• **The Mapping Manager (MM).** The MM offers a dynamic deployment of tasks on ECUs. We use O-TSMBA (OPerational chains-TSMBA) algorithm (described in Chapter 5), a variant of TSMBA[99] that supports task dependencies. The MM takes as input the application description (an initial system configuration file) and changes the current mapping when it is necessary to do so. Changes of the allocation can occur due to the adaptation or in case of one or several ECU failures.

• **The Reconfiguration Manager (RM).** The RM automatically reconfigure tasks inside/or between the different ECUs. The RM is a sporadic task that gets triggered upon the reception of an adaptation trigger (requests for adding new tasks, requests for migrating failed tasks/and or failed ECUs, replacement of tasks with an improved version and removing tasks). Figure 4.4 shows the flowchart of our algorithm TSeRBA (Tabu Search Reconfiguration and Bandwidth Allocation).The ASLA reconfiguration manager uses TSeRBA for adding and/or migrating applications at runtime. TSeRBA uses the following inputs: (1) a schedulable solution obtained from the mapping manager and (2) the adaptation triggers from the monitor (for e.g., the request for adding a new application or a request for migrating a failed applications). The output of TSeRBA is a new system configuration, which needs to be schedulable. To deal with the current adaptation, the algorithm starts by finding the target ECU to host the new and/or the failed applications. TSeRBA maintains an ordered list of ECU candidates, and the one which gives the best QoS is selected as a target ECU. This QoS is the probability of meeting the deadline for applications with soft real-time requirements and it depends on the allocated bandwidth Q_i [73]. Then if the mapping is still valid, the new application is simply added on the target ECU (or the failed applications are simply migrated on the

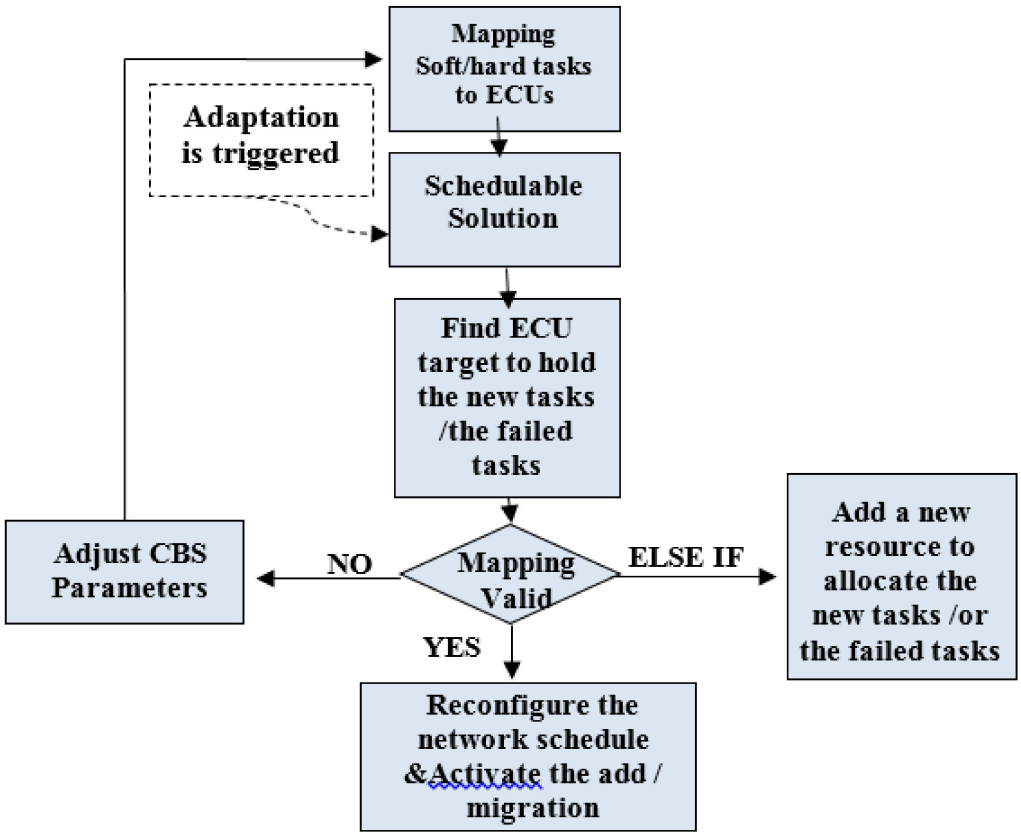

Fig. 4.4 Flowchart of tasks adaptation algorithm (i.e., TSeRBA).

target ECU). Thus, in case of adding a new application, the RM loads, instantiates and connects the new application, changes the network schedule and dynamically finds and binds it to the correct interface. However, if the RM cannot analyze this new application i.e., the mapping is not valid; the bandwidth Q_i associated with applications with soft real-time requirements on that ECU is decreased, and then the new application can be mapped on the ECU. A completely new resource is added to host the new application if the mapping is not valid even when we adjust the bandwidth.

4.1.1.2 ASLA Plugins

All applications will run on top of the ASLA plugins. ASLA plugins support the mechanisms for task reconfiguration and bandwidth allocation and also enable tasks to have guaranteed and protected access to required processing resources during reconfiguration in a timely manner.

4.1.2 A Simple Adaptation-Management Protocol

There are many ways and protocols for managing the process of adaptation in real-time systems. The protocol presented in this section is a simple one that illustrates one of

many possibilities. The process adaptation-management is done based on the protocol of [83]. Our adaptation-management protocol is implemented as a separate thread in ASLA plugin which will be used by the application SWCs and the adaptive SWC as well. Our protocol is designed to provide deterministic adaptation times. In this protocol, each ECU participates in rounds of information exchange, sends a heartbeat message (health vector) to the Adaptive SWC coordinator. The dedicated ECU for the coordinator broadcasts its own health vector including the list of the health vectors to all the other ECUs. Based on this information, only the coordinator will decide which ECU is operational or not. The coordinator can detect if one of the ECUs is no more alive since no heartbeat messages are received from that ECU. Similarly, the other ECUs can detect the failure of the coordinator and one of the ECUs will be promoted to become a coordinator as illustrated in Figure 4.5. The semantic behind this protocol consists of three steps:

1. *Initialization*: At a given time period, each ECU constructs its health vector (i.e., all the NFRs, the utilization processor available on that ECU).

2. *Election* : Each ECU sends its health vector to the other ECUs, upon the reception of the health vector from these ECUs, it compares the available information and the one which has the best health properties, tags its Adaptive SWC as coordinator.

3. *Execution*: This step deals with the runtime management of the adaptation.

We will now discuss the above steps in more detail:

First, The adaptation request is triggered from processing a set of events; these events can be either alarms or triggering conditions monitored from sensors. Each request is assigned an identifier that characterizes it on the distributed system. The adaptation request can be seen as a message containing the ID of the target system configuration, which is sent directly to the Adaptive SWC. These adaptation requests start when event triggering the adaptation is detected.

Second, the adaptation request is identified from its source ECU by the Reconfiguration Manager (RM). So the network is responsible of this step. As we need to send the reconfiguration request to the RM in order to be treated. As we may have regular & irregular event-triggered adaptations we need to take advantage of the service provided by the communication bus i.e., the fact that it allows transmission of sporadic messages.

Third, the RM coordinator receives the reconfiguration from the ECUs or from the RMs running on the others ECUs. The coordinator tracks the evolution of the distributed process and it has a global vision and knowledge about the whole system or application. The coordinator becomes a supervisor of all incoming adaptation requests. Its role it can either to process the request, to reject or accept it, or to send it to the other nodes. For

simplicity we suppose there are no conflicting requests. Once the node is localized (i.e., the adaptation is supervised) the Adaptive SWC coordinator must specify which ECUs need to adapt and which local adaptation each ECU must progress to. Furthermore, a distributed adaptation management protocol is required to ensure the correct coordination of applications under adaptation.

Fourth, The RM broadcasts the reconfiguration message to all the ECUs connected to the network but only the ECUs concerned by the adaptation request open and process the adaptation message. By broadcasting the message the ECU obtains the confirmation of its adaptation request and at the same time all ECUs participating in adaptation will be notified that a change is started in the system by broadcasting adaptation message all the ECUs are aware of the new system configuration.

In the ***Fifth*** step the change is executed. Once receiving the adaptation message, each ECU starts changing the source (current configuration) set of tasks to a target (new configuration) one.

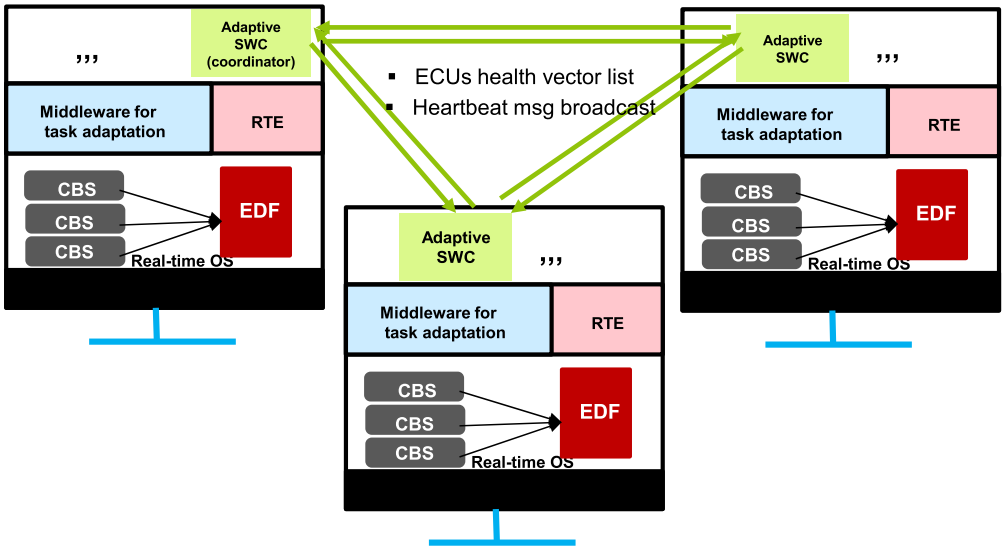

Fig. 4.5 ASLA adaptation-management protocol.

4.2 ASLA Development Methodology

The standard development process is slightly modified for introducing the necessary mechanisms for runtime adaptation. Yet, it remains fully compliant with the AUTOSAR development methodology described in Chapter 3 Section 3.3. The ASLA development process is depicted in Figure. 4.6 and it consists of four steps:

The first step, "The System Configuration" concerns the whole system. During this step the entire set of the applications is specified in terms of software architecture: SWCs, the Adaptive SWC, ports, real-time constraints, HW resource requirements and other

information needed in the vehicle. In our approach we are interested in mixed critical applications with soft and hard real-time requirements. The output of this step is the System Description-an AUTOSAR XML file, which serves as input for the following phase.

The second step, "Extract ASLA ECU-Specific Information" concerns the SWCs and the Adaptive SWC implementation (i.e., the definition of the internal behavior of the Runnables and the RTEEvents). We consider that each SWC contains one runnable and is represented by one AUTOSAR task. During this phase the initial allocation of SWCs with both soft/hard real-time requirements to ECUs is specified and the application signals are mapped to bus frames. As a result of this step we obtain an initial solution (AUTOSAR XML) which is not necessary schedulable and it serves as input for the mapping manager and the following phase.

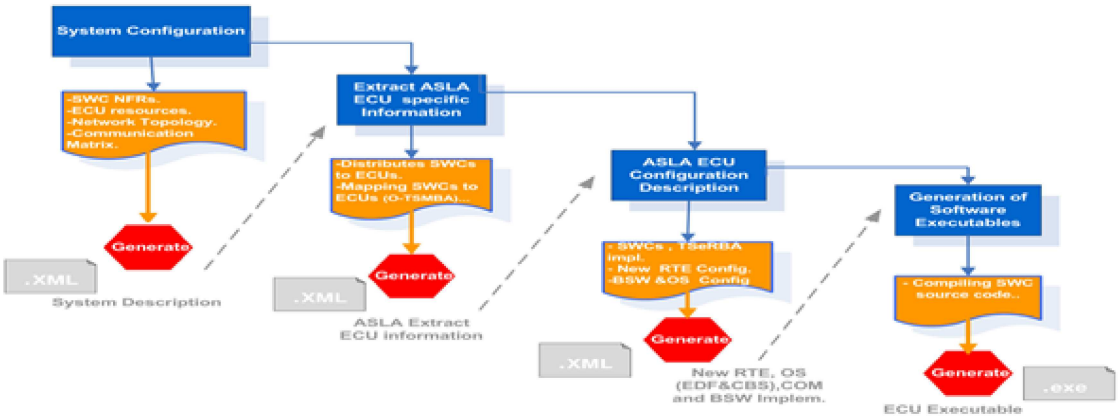

Fig. 4.6 ASLA development process.

The third step, "ASLA ECU Configuration" concerns the configuration of the BSW modules and the RTE of each ECU. Our RTE extension contains functions to support adaptation. The Adaptive SWC and its three components, i.e., the RM, MM and the Monitor, are configured as tasks at the OS level with their corresponding runnables as application components. The callback functions configured at COM and the corresponding APIs defined at the RTE level are used for signal handling. This step includes the scheduling and the tasks mapping concepts. In our approach, we assume that the initial mapping of runnables to AUTOSAR tasks is given at design time similarly to AUTOSAR and all runnables are executed periodically within the context of an AUTOSAR task. However, the mapping of tasks to ECUs is performed using our task mapping algorithm O-TSMBA which we designed for the operational chain model. The output of O-TSBMA algorithm is used as input for the runtime adaptation algorithm (i.e. TSeRBA) described in the previous section. A result of this step is the ASLA ECU Configuration Description is generated aligned with the above steps. Finally, the software executables are generated.

4.3 Summary

We have proposed ASLA (Adaptive system-Level in AUTOSAR) architecture, a novel framework that supports task-level reconfiguration features in AUTOSAR. In this Chapter, we described the main functionality of ASLA and it development methodology. We proceed in the following chapter (i.e., Chapter 5) to describe, in detail, the theoretical and technical aspects of the algorithms behind ASLA architecture.

Chapter 5

The ASLA Approach: Theoretical Study

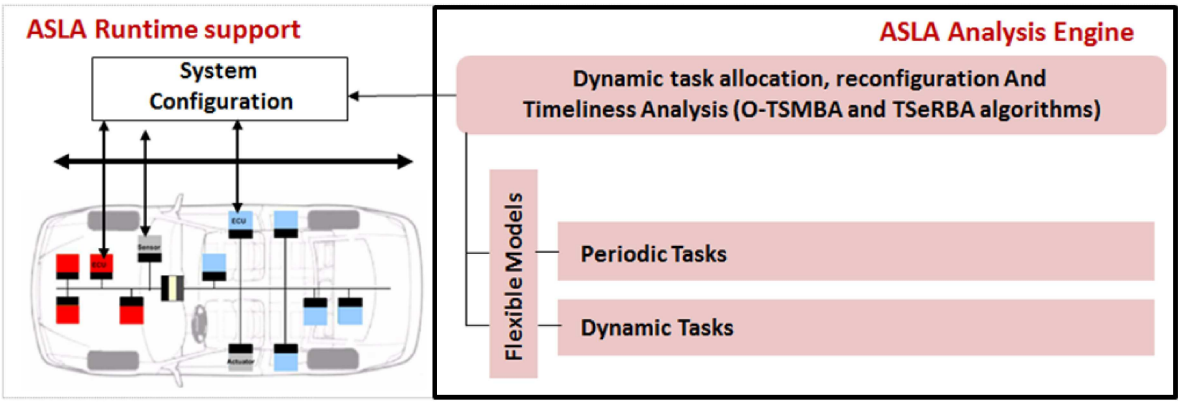

Fig. 5.1 Dynamic task model in the dissertation overview.

Traditionally, automotive systems have been designed to be unchanged at runtime in order to maintain the system predictability. However with the new trends in automotive industries and the emerging applications such as smart cars, etc, has proved that some **flexibility** can be provided to real-time systems mainly soft real-time systems. Dealing with runtime adaptation in real-time systems is a hard problem to solve due to the complex deployment and configuration issues involved in satisfying multiples requirements such as **timeliness** and **fault-tolerance**. Effective deployment requires developing and evaluating a range of task allocation and reconfiguration algorithms that satisfy automotive system requirements while reducing resources usage. Therefore, we tackle in this chapter the problem of task mapping and reconfiguration (see Figure 5.1). This chapter makes two contributions. First, it describes a novel task allocation algorithm for mapping runnables to ECUs. Second, it presents TSeRBA (Tabu Search Mapping and Bandwidth allocation) algorithm for task reconfiguration and bandwidth allocation. These two contributions

are realized with ASLA, the framework we designed for task adaptation in AUTOSAR (Chapter 4).

5.1 ASLA System Models and Assumptions

In this section, we state our assumptions, the problem definition for our work on ASLA. We describe ASLA's system, fault and reconfiguration models.

5.1.1 ASLA's System Model

Applications in the automotive system use sensor information to obtain current system's information. For instance, in Steer-by-Wire (SBW) applications [60], sensors are typically used to measure and obtain information about steering wheel movement. This information is then fed into the system to be processed. The controllers compute signals for steering movements with the information from sensors. The computed signals are then sent to the steering wheel actuator for the motors, which are handled periodically for timely handling user operations and reactions to the environment. In order to reflect this nature, we define an "OPerational chain" OP, which is composed of periodically executing runnables generating data and events regularly that propagate through multiple runnables. An "OPerational chain" also has an end–to–end delay from the input to the output[1]. Within an OP, runnables are classified into sensor/actuator runnables and other computational runnables. For instance, an actuator runnable controlling the steering wheel motors must run on the ECU connected to the motor in an SBW application. Every runnable generates data to be fed to other runnables, except actuator runnables which terminate the OP. Figure 5.2 shows an example of operational chains (with hard and soft real-time requirements) applied to E/E vehicle. Handling adaptations on demand with bounded time is a requirement for real-time systems. By handling adaptations within a specified timing boundary, time to adapt can be bounded, and the system can operate continuously. To limit time to adapt, authors of [117] classified software tasks into three classes: Hard Tasks, Soft Tasks, and Best-Effort Tasks. In our work, we apply the same classification to SWCs, where a runnable is a part of an SWC [2]. In this work, we consider a system with two criticality levels: Hard and Soft Tasks. Considering multiple criticality levels is left as future work. An SWC with strict timing constraints, for which deadline misses lead to a catastrophic degradation of the system is classified as a *Hard software Component (Hard SWC)*. An SWC which has flexible timing constraints,i.e., the deadline misses are tolerable is classified as a *Soft Software Component(Soft SWC)*. The TSMBA (Tabu Search Mapping and Bandwidth Allocation) algorithm [99] provides a comprehensive solution

[1] Dealing with End-to-End delay of an OP is beyond the scope of this work, we calculate it similarly to [74]

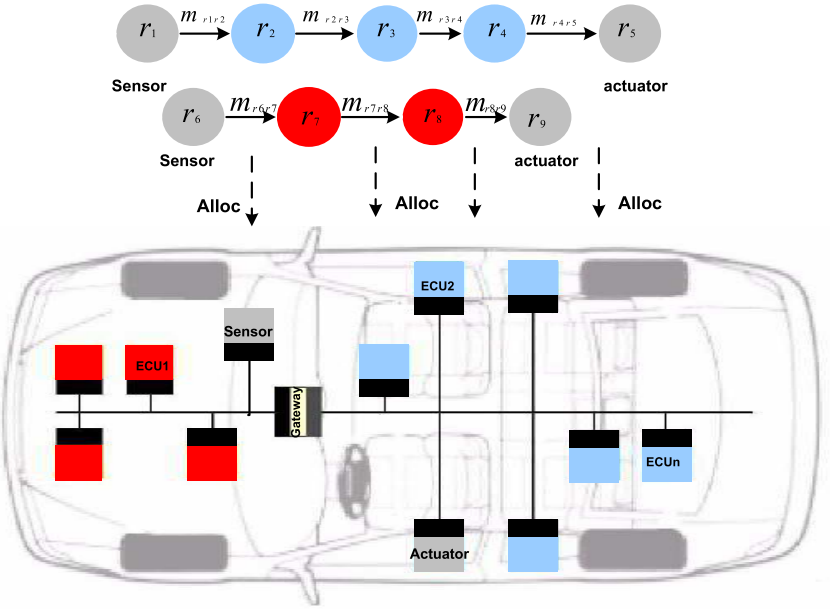

Fig. 5.2 An example of operational chains can be applicable to an E/E vehicle, where r represents a runnable, and m denotes a message between two runnables.

that allocates *Hard SWC*, and *Soft SWC* to distributed heterogeneous architectures and performs processor bandwidth reservation for guaranteeing timing requirements even in the presence of faults. From dependability perspective, a single failure of a runnable within an OP will affect all its successors such that the overall OP requirement is violated. TSMBA cannot be used directly in systems with data dependencies. Therefore, we propose a new allocation algorithm based on TSMBA, O-TSMBA (OPerational chain-TSMBA) which is a variant of TSMBA that takes dependencies among SWCs.

Then we add support for reconfiguration to O-TSMBA. We call this algorithm TSeRBA(Tabu Search Reconfiguration and bandwidth Allocation). TSeRBA has two properties that distinguish it from TSMBA: (i) It supports dynamic SWCs allocation and reconfiguration within AUTOSAR, As we said in the previous chapters, currently, there is no support for doing this within AUTOSAR. In our work, we propose enhancements to the different layers of AUTOSAR (i.e., from the application down to the OS layer) to enable runtime adaptation and, therefore, provide support for TSeRBA, (ii) it considers dependencies among SWCs.

5.1.1.1 Software Architecture

We model an application as a set R of interacting runnables $\mathsf{r_i} \in$ R. R is composed of n runnables,$\mathsf{r_1}, \ldots, \mathsf{r_n}$. Each runnable is part of an atomic SWC, $\mathsf{sw_j}$, which may have several runnables. In order to represent a relationship between a runnable, $\mathsf{r_i}$ and the SWC, $\mathsf{sw_j}$, we define a function ω such that $\omega(\mathsf{r_i}) = \mathsf{sw_j}$ when $\mathsf{sw_j}$ contains $\mathsf{r_i}$. The function

inverse of ω,ω^{-1} returns all the runnables inside an SWC. Runnables receive data via input ports $\mathsf{P}^{\mathsf{in}}_{\mathsf{r_i}}$ and send out the data via output ports $\mathsf{P}^{\mathsf{out}}_{\mathsf{r_i}}$. $\mathsf{P}^{\mathsf{in}}_{\mathsf{r_i}}$ and $\mathsf{P}^{\mathsf{out}}_{\mathsf{r_i}}$ are defined as a set of input, output ports of a runnable $\mathsf{r_i}$, respectively. Runnables interact between each other via $\mathcal{L}$, which is a set of links. An interaction between runnables is represented by a link $\mathsf{l_i}$. Runnables exchange data through signals $\mathsf{s_i}$ on the links. We define $\mathsf{Snd}(\mathsf{s_i})$,$\mathsf{Rcv}(\mathsf{s_i})$ as a function returning the runnables that send a data signal $\mathsf{s_i}$ and function returning the runnables receiving a data signal $\mathsf{s_i}$, respectively. In order to express the dependency relationship between the SWCs, we define a function Dep: R→R, such that $\mathsf{Dep}(\mathsf{r_i})$ returns all the runnables that depend on $\mathsf{r_i}$, in other words, a runnable $\mathsf{r_i}$ depends on a runnable $\mathsf{r_j}$, if there is a link $\mathsf{l_i}$ between both runnables $\mathsf{r_i}$ and $\mathsf{r_j}$ such that $\mathsf{l_i}$ connects the input port of $\mathsf{r_j}$ (i.e., $\mathsf{P}^{\mathsf{in}}_{\mathsf{r_j}}$) to the output port of $\mathsf{r_i}$ (i.e., $\mathsf{P}^{\mathsf{out}}_{\mathsf{r_i}}$). The set of SWCs are also given, Γ, for guaranteeing different timing requirements, Γ is classified by a function RQ into two sets.

The function RQ: $\Gamma \rightarrow \{Hard, Soft\}$, determines if a SWC is in hard or soft real-time subsets, respectively. The SWCs are allocated on a distributed heterogeneous architecture denoted by a set of ECUs. The allocation is denoted by the function[2] χ: R→ECUs. This allocation is not yet known but would be decided by the *Cost function* described in Section 5.1.4. We assume that each runnable is represented by a periodic task [99] $\mathsf{r_i}$, which releases a job every $\mathsf{T_i}$ units of time, where each job consumes at most $\mathsf{C_i}$ units of computation time and should be completed within a relative deadline $\mathsf{D_i} = \mathsf{T_i}$. Both runnables Soft and Hard are periodic and have a period $\mathsf{T_i}$. A subset $\mathsf{OP_i} \subset$R contains runnables for the $\mathsf{i^{th}}$ operational chain out of total m operational chains. In certain cases, a runnable $\mathsf{r_i}$ can be an element of $\mathsf{OP_i}$ and also an element of $\mathsf{OP_l}$, where $\mathsf{i} \neq \mathsf{l}$. The relationship among the tasks in the $\mathsf{i^{th}}$ operational chain $\mathsf{OP_i}$ is represented by a directed graph $\mathsf{G_i}$. In this graph, a node u denotes a runnable, $\mathsf{r_i} \in$ R, and an edge (u, v) of $\mathsf{G_i}$ indicates data flow from u to v; the edge (u, v) has its own message, $\mathsf{m_{uv}}$, which is generated by the node u and consumed by the node v in $\mathsf{G_i}$. We assume that the allocation of each runnable, $\mathsf{r_i}$, to the graph is given at design time. Certain runnables can only be mapped on certain ECUs. Let denote $\mathrm{u}(\mathsf{G_i}, \mathsf{r_i})$ to be the node of $\mathsf{G_i}$ allocated to the runnable, $\mathsf{r_i}$. The $\mathrm{u}(\mathsf{G_i}, \mathsf{r_i})$ value is $\varnothing$ if the runnable $\mathsf{r_i}$ is not an element of $\mathsf{OP_i}$. This function is also applicable to a SWC. An $\mathsf{OP_i}$ represented by a pair $(\mathsf{T^{OP}_i}, \Delta)$ where $\mathsf{T^{OP}_i}$ is the period of the application $\mathsf{OP_i}$ and Δ is an end–to–end delay, which is defined as the worst-case delay between the release time of the first executed node in the graph $\mathsf{G_i}$ and the completion time of the last executed node in $\mathsf{G_i}$. All runnables in $\mathsf{OP_i}$ have the same period $\mathsf{T^{OP}_i}$. So a runnable with hard real-time requirement $\mathsf{r_i}$ is characterized by its Worst Case Execution Time (WCET) $\mathsf{C_i}$ and a deadline $\mathsf{D_i}$. For a hard runnable $\mathsf{r_i}$, the WCET $\mathsf{C^{ECU_j}_i}$ is known for each processing element $\mathsf{ECU_j}$ where $\mathsf{r_i}$ is considered for the allocation. Soft runnables are characterized by the Probability Distribution Function

[2] Allocating a runnable in the AUTOSAR implies that the corresponding SWC is also allocated. In other words, all runnables of a SWC should be assigned to one ECU

(PDF) $U_i^{ECU_j}$ of their execution times and a soft deadline δ_i. $U_i^{ECU_j}(h)$ is the probability that the job $J_{i,k}$ of r_i has an execution time of h on the processing elements ECU_j. Each soft runnable (or task) must have An Expected Utility (EU) function $EU_{ci}(r_i, ECU_j)$ for each ECU_j whose values is defined as the probability-weighted sum of possible values of the soft tasks' execution time (i.e., c_i). The QoS of a soft runnable r_i is defined as the probability of meeting the deadline δ_i, i.e, QoS $(\delta_i) = P\{f_{i,k} \leq ar_{i,k} + \delta_i\}$,where $f_{i,k}$ and $ar_{i,k}$ are the finishing and arrival time of the k^{th} job of runnable r_i, respectively. An SWC sw_j is also characterized by a worst-case execution time C_j and a deadline D_j, however, it differs from runnables in the sense that it: $C_j = \Sigma_{\forall r_i \in \omega^{-1}} C_i$, $T_j = min_{\forall r_i \in \omega^{-1}} T_i$ and the deadline is $D_j = min_{\forall r_i \in \omega^{-1}} D_i$. Let U_i denote the utilization of an SWC sw_j with hard real-time requirement and it is defined as C_j/T_j. whereas an SWC sw_j with Soft real-time requirement its utilization is defined as Q_j/T_j. Throughout this dissertation, we restrict ourselves to implicit-deadline task systems.

5.1.1.2 Hardware Architecture

We model the hardware architecture as an undirected graph $G_h = (V_h, F_h)$. Nodes V_h represent a set of hardware resources and the edges F_h represent the communications links (or buses) between them. The hardware resources consist of a set of heterogeneous processing units (or ECUs), to which a runnable set R can be potentially mapped, run and delivered as part of SWCs. The $ECUs$ are interconnected by a communication channel bus B on which messages m_{ij} are exchanged. We assume that we use a fault-tolerant network such as CAN [45] as the underlying in-vehicle network. Communicating SWCs allocated on different ECUs exchange messages on the communication channel. We assume that the network has an upper-bound on message delivery and is completely connected. This assumption is reasonable for automotive systems. Relaxing this assumption through the integration of our framework with the network level fault-tolerance techniques is an area of future work. Each $ECU_i \in ECUs$ is composed of an EDF scheduler and a middleware implementing online tasks reconfiguration mechanism. The soft tasks are scheduled by the CBS server. The hard tasks and the CBS servers are scheduled using EDF. The temporal isolation between the hard and the soft real-time tasks is enforced by CBS, thus guaranteeing the schedulability of the hard tasks. The soft tasks τ_i are assigned to a CBS, characterized by the couple (Q_i, T_i) where Q_i (bandwidth server) represents a time that the soft task is allowed to use ECU every period T_i. The $ECUs$ have access to a shared memory where the code of the tasks is stored. The system model is illustrated in Figure. 5.3.

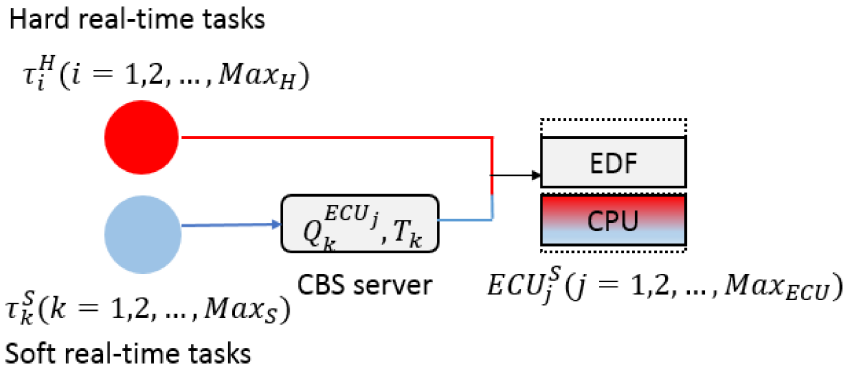

Fig. 5.3 Soft real-time tasks are served by a CBS server, while hard real-time tasks are directly scheduled.

5.1.2 Problem Statement

Based on the software and the hardware architecture presented above, we state the problem to solve as follows: *given a set of ECUs and a system Configuration (consisting of a set of OPs with hard and soft real-time requirements) determine whether all tasks of every OP* [3] *will always meet all deadlines (both hard and soft constraints) under any possible adaptation sequences.* However, there are many sub-problems that we need to consider in conjunction with the adaptation.

P1. ***Allocation problem.*** Can we find an optimal allocation schemes for each OP to maintain the schedulability analysis? By applying our global scheduling algorithm; we determine the mapping and the utilization such that the deadlines for all hard tasks are satisfied and the probability of meeting the deadline (i.e.,QoS [4]) for the soft tasks is maximized.

P2. ***Adaptation problem.*** Knowing a best allocation, can we guarantee all deadlines both (hard and soft) in the presence of the adaptation such as adding, deleting and migrating tasks? The objective of scheduling the runtime adaptation is to guarantee that a system is schedulable not only after the adaptation but also during the adaptation path. In the next we will give a simple example to show the difficultly of runtime adaptation.

Task τ	1	2	3	4	5	6	7	7′	8	9	10
U_i	0.19	0.186	0.16	0.1	0.165	0.15	0.4	0.4	0.42	0.32	0.32
Q_i	29	29	31	31	33	35	-	-	-	-	-

Table 5.1 Example task set.

[3] The number of OPs depends on the number of applications in the system
[4] The QoS derivation is described in Section 5.1.4

Example 1.2 Consider a system configuration in Figure. 5.4 with a 3 ECUs, 4 hard tasks and 6 soft tasks. The task set parameters are reported in Table 5.1. Task parameters were chosen to keep the example simple and easily understandable; they should not be considered as real task cases. We assume that the allocation tasks to ECUs depicted in Figure. 5.4 has been already defined and it is so far the best solution (i.e., within which the deadline for hard tasks is satisfied and the QoS for soft tasks is maximized). The system configuration is schedulable under EDF in the absence of adaptation.

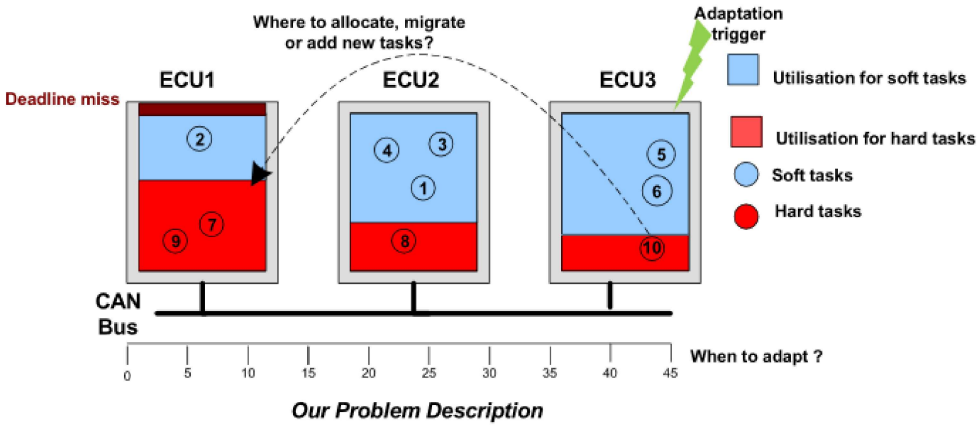

Fig. 5.4 The missed deadline during the adaptation.

Further, let us assume that during runtime, specifically at time t=9 an adaptation is triggered that updates the task τ_7 to $\tau_7^{'}$; this adaptation will lead other tasks on ECU_1 to miss their deadlines at time t=12. This example illustrates that reconfiguring a task may cause deadline miss of other tasks on the system.

5.1.3 Fault Model

This work focuses on a fail-stop model of failures, where tasks or ECUs can fail and the remainder of the system can continue executing. We assume that ASLA employs a failure recovery policy[99], which consists of restoring the last non-faulty state of the failing task, i.e., to recover from faults. This state has to be saved in advance in the shared memory and will be restored if the task fails.

5.1.4 Cost Function Calculation (i.e., QoS)

The schedulability analysis of a soft task τ_i (i.e., the probability of meeting its deadline δ_i; QoS(δ_i)) is calculated using the stochastic analysis method described in [73] (see Chapter 2 Section 2.1.3–CBS). This QoS guarantee depends on both the PDFs and the bandwidth server Q_i. As we said previously in Section 5.1.1, for a given soft real-time task τ_i, The QoS r_i is defined as the probability of meeting the deadline δ_i, i.e., QoS

$(\delta_i) = P\{f_{i,k} \leq ar_{i,k} + \delta_i\}$,where $f_{i,k}$ and $ar_{i,k}$ are the finishing and arrival time of the k^{th} job of runnable r_i, respectively. This QoS guarantee is calculated by by modeling the CBS server(serving the Soft tasks) as a queue [73]. The arriving job $j_{i,k}$ are seen as tokens to be served by the server having the capacity Q_i (where is the capacity of the server Q_i). As we had previously described, the PDF of the computation times of jobs of the tasks allocated on the ECU_j is defined to be $U_i^{ECU_j}$. In this model, $U_i^{ECU_j}$ represents the PDF of the incoming requests for the queue: $U_i^{ECU_j}(h) = P\{c_{i,k} = h\}$, Probability that the arriving job requires h units of computation time. The system is modeled with a queue where: a request of $c_{i,k}$ units arrives every T_i. Since the server capacity is Q_i, at most Q_i can be served every T_i units of time. The system is described with a random process defined as follows:

$$\begin{cases} v_1 = & c_{i,1} \\ v_k = & max\{0, v_{k-1} - Q_i\} + c_{i,k} \end{cases}$$

The state variable v_k indicates the length of the queue (in time units) immediately after the job $j_{i,k}$ having a computation time $c_{i,k}$ arrives. It can be shown that the job $j_{i,k}$ will finish before this time:

$$f_{i,k} = ar_{i,k} + \lceil \frac{v_k}{Q_i} \rceil T_i$$

Thus, the probability that the queue length is v_k immediately after a job arrives is a lower bound to the probability that the job would finish before the deadline $\delta_i = \lceil \frac{v_k}{Q_i} \rceil T_i$.

Let $\pi_m^{[k]} = P\{c_{i,k} = h\}$ be the state probability of the process v_k. We already have the PDF of the request times of the arriving jobs, i.e., $U_i^{ECU_j} = P\{c_{i,k} = h\}$. Since we know that $c_{i,k}$ is time invariant, as $U_i^{ECU_j}(h)$does not depend on i, k, the value of $\pi_m^{[k]}$ can be calculated as follows:

$$\pi_m^{(k)} = P\{v_k = m\} = P\{max\{v_{k-1} - Q_i, 0\} + c_k = m\}$$

Finally the solution comes out to be:

$$\begin{aligned} \pi_m^{(k)} &= \sum_{h=0}^{Q_i} (U_i(m) \times \pi_h^{(k-1)}) \\ &+ \sum_{h=Q_i+1}^{\infty} (U_i(m - h + Q_i) \times \pi_h^{(k-1)}) \end{aligned}$$

Using a matrix notation, one can solve for $\pi_m^{(k)}$ using the following equation:

$$\Pi^k = M\Pi^{k-1}$$

where:

$$M = \begin{pmatrix} U_i^{ECU_j}(0) & . & .. & U_i^{ECU_j}(0) & 0 & 0 & .. \\ U_i^{ECU_j}(1) & . & .. & U_i^{ECU_j}(1) & U_i^{ECU_j}(0) & 0 & .. \end{pmatrix}$$

and

$$\Pi^k = \begin{pmatrix} \pi_0^k \\ \pi_1^k \\ \pi_2^k \\ . \\ . \end{pmatrix}$$

From the queuing model theory the condition for the queue to be stable (i.e., the number of elements in the queue do not diverge to infinity) is presented as :

$$\mathcal{P} = \frac{mean \;\; interarrival \;\; time}{mean \;\; service \;\; rate} < 1$$

In our case the stability condition is achieved when the mathematical expectation of the PDF of incoming requests is less than the server capacity, i.e., $\overline{\mathsf{c_{i,k}}} < \mathsf{Q_i}$. If this condition is not satisfied, the difference between deadline $\mathsf{f_{i,k}}$ assigned by the server for the job $\mathsf{J_{i,k}}$ and the job release time $\mathsf{ar_{i,k}}$ would increase indefinitely, thus the queue would overflow and the schedulability of the other tasks would slow down in unpredictable manner. For stable queue, a stationary solution of the Markov chain describing the queue can be found, i.e. there exists a solution Π such that $\Pi = \lim_{i,k \to +\infty} \Pi^{i,k}$. Since the size of the matrices M and Π are infinite, the calculation of an exact solution is computationally expensive. Thus, the matrices can be truncated and an approximate solution can be found. However, by doing this we get inaccurate (approximated) results, but the computation complexity is highly improved. The Matrix M is truncated to an $N \times N$ matrix $M^{'}$ and the problem of finding the stationary solution becomes an eigenvector problem, i.e., one has to find $\Pi^{'}$ such that $\Pi^{'} = M^{'}\Pi^{'}$.

The resulting eigenvector $\Pi^{'}$ has to be normalized since the steady state probabilities must sum to one. This is done by dividing all the elements of the eigenvector by their combined sum. More formally:

$$\Pi^{''} \frac{\Pi^{'}}{\sum_N \Pi^{'}}$$

The resulting stationary solution is in fact — a PDF of the stochastic variable v. In our case, we would like to compute the CDF as it describes the system completely. We can recall that the CDF of a stochastic variable expresses the probability that the variable is less than or equal to a given value. The CDF $D(v)$ of the given stochastic variable v can be easily computed as

$$D(v) = \sum_{m=0}^{v} \pi_m.$$

Since we know that the deadline δ is related to the length of the queue as $\delta_i = \lceil \frac{v_k}{\mathsf{Q_i}} \rceil \mathsf{T_i}$, the probabilistic guarantee $Qos(\delta_i)$ can be easily computed as :

$$Qos(\delta_i) = D(\lceil \frac{\delta_i}{\mathsf{T_i}} \rceil \mathsf{Q_i})$$

The quality of solution depends a lot on the truncation point N. Essentially, by setting a truncation point on a state probability vector, the state space is reduced. Thus, it would be a good idea to set the truncation point such that, only the states which are impossible to reach are removed. In this case, the elements in the state probability vector Π represents the probability of the queue having a certain length. By setting the truncation point at the point representing the maximum possible length of the queue, we can have a safe assumption. The maximum possible length of the queue can be calculated by considering the PDF of the incoming requests and having a certain upper limit on the total number of jobs.

Let's assume that the total number of jobs are: JOBS. The probability distribution function of incoming jobs is given as $U_i^{ECU_j}$. The PDF is finite and let's say it has a length of L_i. The server capacity is given as $\mathsf{Q_i}$.

The worse case scenario in this case would be observed when all the jobs having request times greater than the server capacity would arrive one after another. Thus, the maximum possible queue size can be calculated as:

$$N = \sum_{h=Q_i+1}^{L_i} U_i^{ECU_j}(h) \times h \times JOBS$$

This value N can be a safe truncation point. We have used this in our implementation to truncate and then solve the Markov chain for a stationary solution, our proposed algorithm will decide on the appropriate Q values that minimize the cost function, thus maximizing the total QoS.

5.1.4.1 Cost Function Calculation – A Numerical Example

We considered a periodic task having a period of 8 and distributed computation times as shown in Figure. 5.5. The Q value for this example is 4 and the deadline δ is 6.

The worst case length of the queue (for 100 jobs) came out to be 67. Thus, this was set as the truncation point for calculation of the $M^{'}$ matrix. The dimensions of $M^{'}$ are fixed at 67x67. The $M^{'}$ matrix results as follows:

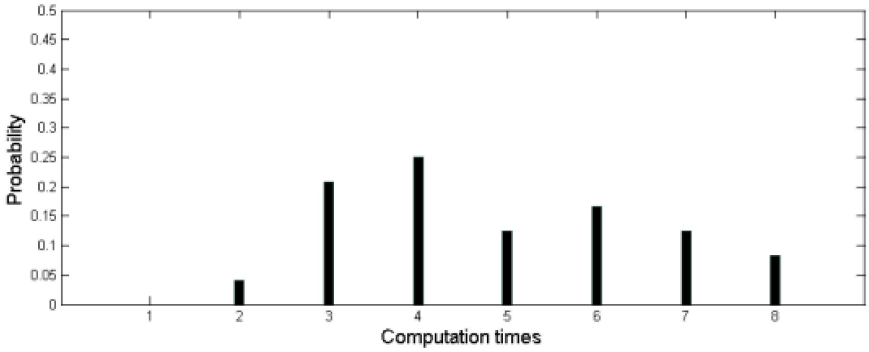

Fig. 5.5 PDF of the computation times.

$$M = \begin{pmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & . & . & . \\ 0.0417 & 0.0417 & 0.0417 & 0.0417 & 0.0417 & 0.0 & 0.0 & . & . & . \\ 0.2083 & 0.2083 & 0.2083 & 0.2083 & 0.2083 & 0.0417 & 0.0 & . & . & . \\ 0.25 & 0.25 & 0.25 & 0.25 & 0.25 & 0.2083 & 0.0417 & . & . & . \\ 0.125 & 0.125 & 0.125 & 0.125 & 0.125 & 0.25 & 0.2083 & . & . & . \\ 0.1667 & 0.1667 & 0.1667 & 0.1667 & 0.1667 & 0.125 & 0.25 & . & . & . \\ 0.125 & 0.125 & 0.125 & 0.125 & 0.125 & 0.1667 & 0.125 & . & . & . \\ 0.0833 & 0.0833 & 0.0833 & 0.0833 & 0.0833 & 0.125 & 0.1667 & . & . & . \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0833 & 0.125 & . & . & . \\ . & . & . & . & . & & . & . & . & . \\ . & . & . & . & . & & . & . & . & . \end{pmatrix}$$

This matrix $M^{'}$ is then raised to higher powers until the desired accuracy level is reached. The accuracy is measured by calculating the quadratic distance between the elements of the first and the last column. We have set this accuracy level as 10^{-40}. Figure. 5.6 shows the measured error values for different values of power. For this particular case, $M^{'}$ is raised to 19, since the measured error for higher powers is less than the fixed accuracy level. The error measured at iteration 19 is 1.3693056903764797E-41.

One of the columns is then extracted and the CDF is generated. Then $Qos(6)$ is calculated. It comes out to be 0.53. This is multiplied with its weight to give the final value of the cost function. The plot for $QoS(\delta_i)$ can be seen in Figure. 5.7

5.1.5 Definitions

We first introduce some definitions:

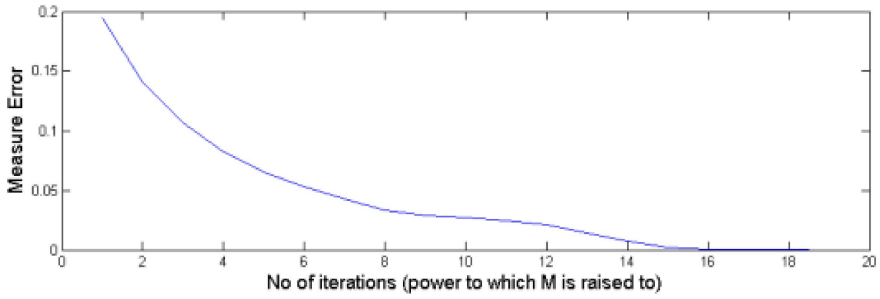

Fig. 5.6 Error in calculation of stationary solution vs number of iterations (power to which $M^{'}$is raised).

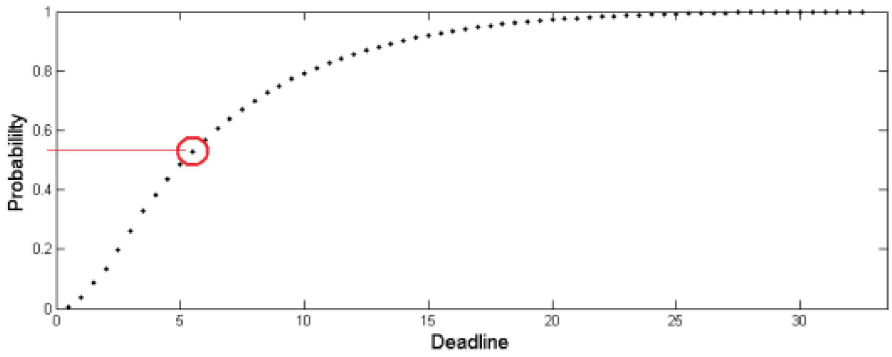

Fig. 5.7 Probability to satisfy a certain deadline, $Qos(\delta_i)$.

Definition 12 (Demand-Bound Function) *A demand-bound function* $\mathsf{dbf}(\tau_r, L)$ *gives an upper bound on the maximum possible execution demand of task* τ_r *in any time interval of length* L *where,* dbf *is calculated as the total amount of required execution requirements of all jobs of* τ_r *with their whole scheduling windows within the time interval,* dbf *can be calculated as follows.*

$$\mathsf{dbf}(\tau_r, L) \stackrel{\text{def}}{=} \max\left(0, \left\lfloor \frac{L - D_r}{P_r} \right\rfloor + 1\right).C_r \tag{5.1}$$

Definition 13 (Time2adapt) time2adapt *denoted* δ *is defined as the time instant relative to the release time of* τ_r^* *within which job of* τ_r^* *should be reconfigured. During this time the task's period is prolonged at runtime in order to accommodate the new request for reconfiguration. This operation is called task compression in which the period of task is prolonged from* T_i *to* T_i' *so the utilization of the task is reduced to* $U_i = C_i/T_i'$ *which gives some room to insert a new task i.e., the freed utilization* $U_i - U_i'$*. If a task* τ_i *is compressed at* t_r*, then:*

$$\delta = \begin{cases} d_i - \dfrac{C_i(t_r)}{U_i - U_i'} = (t_0 + T_i) - \dfrac{C_i(t_r)}{U_i - U_i'} & if\ d_i - \dfrac{C_i(t_r)}{U_i - U_i'} > t_r \\ t_r & if\ d_i - \dfrac{C_i(t_r)}{U_i - U_i'} \leq t_r \end{cases}$$

where d_i *is the deadline of the current instance of the task* τ_i. $C_i(t_r)$ *is the new computation time of* τ_r *which is executed with the new period* T'_i *and* t_r *is the time to accommodate the new requests during which the deadline of the instance is delayed from* $t_0 + T_i$ *to* $t_0 + T'_i$ *where* t_0 *is the release time of the current instance. At* t_r, $t_0 \leq t_r \leq t_0 + T_i \leq t_0 + T'_i$.

Definition 14 ***(Safe adaptation path)*** *A safe adaptation path* sp *comprises an ordered series of reconfiguration actions that modify the structure and the behavior of the system configuration (i.e., OP);* sp *has the following parameters* $sp = (lgth, \tau_r, D, nbr, Dep)$— *The length of the safe adaptation path* $lgth(sp_i)$ *which varies depending on the reconfiguration type i.e., the number of reconfiguration actions within the path* nbr, *with* $nbr \geq 1$ .Dep *function (see section 5.1.1.1) of the affected* SWCs *is used to determine which reconfiguration actions need to be grouped in* sp *to reach a target system configuration. The global deadline of* sp *is equal to the sum of all the reconfiguration actions deadlines,*D *where*

$$D_{max}(sp_i) \stackrel{def}{=} \Sigma D^{n_i}_{j^*=1}(\tau^*_r), \text{ with } lgth(sp_i) > D_{max}(sp_i)$$

Example 1.1 (Safe adaptation path) *Consider a system with a set of mixed tasks hard and soft, both tasks are schedulable under EDF without reconfiguration. However,*

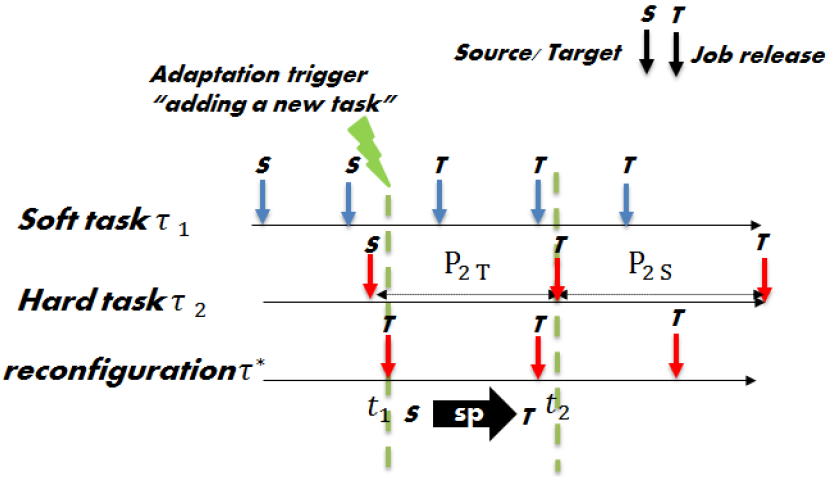

Fig. 5.8 Safe adaptation path overview.

at time t_1 *a reconfiguration task is triggered and safe adaptation path* sp *is released at* t_1 *from a source configuration* ***(S)*** *to a target configuration* ***(T)****: while this adaptation does not change any task parameter of* (τ_1), *it extends the period of* (τ_2) *and introduces a new task as illustrated in Figure. 5.8. With our approach, we want to determine whether a mixed task set* (τ) *is schedulable in the presence of a runtime adaptation. The actual execution behavior during a* sp *depends on online information, such as the time instant when the current adaptation trigger is released, and the release and execution patterns. Since our system is adaptive and evolve at runtime it is able to monitor and keep track of the information which are required for the schedulability analysis.*

In Chapter 2, we have shown how Baruah *et al.* [21] computes the total resource demand of sporadic tasks under EDF scheduling over an interval [L] i.e., DBF (see equation 5.1). By using the concept of DBF introduced in Definition 12, it is possible to formalize, bound and ensure the schedulability analysis of our safe adaptation path sp as follows:

Definition 15 ***(Demand-Bound Function for Safe Adaptation Path)*** *DBF for sp_i can be calculated as follows:*

$$\mathsf{DBF}_{\mathsf{sp_i}}(\tau^*_r, \mathsf{lgth}(\mathsf{sp_i})) = \max\left(0, \left\lfloor \frac{\mathsf{lgth}(\mathsf{sp_i}) - \mathsf{D}^*_r}{\mathsf{P}^*_r} \right\rfloor + 1\right).\mathsf{C}^*_r \text{ with :}$$

$$\begin{cases} \mathsf{C}^*_r = \Sigma \mathsf{c}^{\mathsf{n}}_{\mathsf{j}^*=1}(\tau^*_r) \\ \mathsf{D}^*_r = \Sigma \mathsf{d}^{\mathsf{n}}_{\mathsf{j}^*=1}(\tau^*_r) \\ \mathsf{lgth}(\mathsf{sp_i}) > \mathsf{D}^*_r \\ \text{and} \quad \mathsf{D}^*_r = \mathsf{P}^*_r. \end{cases}$$

Definition 16 ***(Feasible Safe Adaptation Path)*** *Let* sp *be a safe adaptation path.* sp *is* ***feasible*** *(i.e., schedulable) if and only if $\Sigma_{\tau^*_r \in \mathsf{T}} \mathsf{dbf}(\mathsf{T}, \mathsf{L}) \leq \mathsf{L}$ for some positive interval* L*, where* $\mathsf{L} = \mathsf{lgth}(\mathsf{sp})$.

Definition 17 ***(Feasible Mapping)*** *Let (Γ, ECU) denote a system with heterogeneous* ECU*, with $|\Gamma| = n$ and $|\mathsf{ECU}| = m$. Let $\chi : \Gamma \rightarrow \mathsf{ECU}$ denote a mapping from the tasks of Γ to the ECUs in* ECU*. Let $\Lambda_{\chi(j)}$ denote the set of all tasks hard and soft mapped onto $\mathsf{ECU_j}$ by the mapping $\chi : \Lambda_{\chi(j)} \stackrel{\text{def}}{=} \{i | \chi(i) = j\}$. χ is a* ***feasible mapping*** *if and only if it satisfies the following condition:*

$$\forall j : 1 \leq j \leq m : \Sigma_j(U_j(\Lambda_\chi(j)) \leq 1) \tag{5.2}$$

Definition 18 ***(Feasible Adaptation)*** *The adaptation is said to be feasible if (i) if the mapping is feasible and (ii) the safe adaptation path is feasible too.*

5.1.6 ASLA's Reconfiguration Model

The reconfiguration in our case is a sporadic task τ^*_r has the following four timing properties: $(\mathsf{C}^*_r, \mathsf{D}^*_r, \mathsf{P}^*_r, \mathsf{time2adapt})$—a worst case execution time C^*_r, a relative deadline D^*_r, a minimum inter-arrival time P^*_r and the time instant in which jobs of τ^*_r are executed time2adapt, which is used as the necessary bounded time in which we execute the reconfiguration and the system need not to be stopped. This time is mainly based on the work of [59], in which authors studied what is the best time to introduce new tasks into an EDF-scheduled system. τ^*_r comprising n reconfiguration actions [5], or so-called jobs i.e.,

[5] The terms reconfiguration action and job are used interchangeably

$\{\forall \tau_r^* \in \tau_i, \tau_r^* \stackrel{def}{=} \{j_1^*, j_2^*, .., j_n^*\}\}$. The jobs of τ_r^* are separated by at least P_r^* time units and each one has the maximum execution time C_r^* time units and must complete within D_r^* time units after its arrival. So, $C_r^* \stackrel{def}{=} \Sigma c_{j^*=1}^n(\tau_r^*)$ where $\{c_{j_1^*}, c_{j_2^*}, .., c_{j_n^*}\}$ denotes the processing times of n reconfiguration actions of the reconfiguration task τ_r^*. and $D_r^* = \Sigma d_{j^*=1}^n(\tau_r^*)$ where $\{d_{j_1^*}, d_{j_2^*}, .., d_{j_n^*}\}$ denotes the execution durations or the deadlines of the reconfiguration actions in τ_r^*. The utilization of the task τ_r^* is defined as $U_{RQ}^{ECU} \stackrel{def}{=} C_r^*/P_r^*$ which is the additional utilization while a task is being reconfigured i.e., the required resources for reconfiguring a task on the ECUs within the remaining time after triggering reconfiguration. Whereas the task system utilization is equal to $\Sigma U_{others} + U_{RQ}^{ECU}$.

The reconfiguration task has a global view on the tasks running on the different ECUs. An adaptation transforms a current system configuration to a new system configuration by applying reconfiguration actions on the respective configurations. ASLA uses the following six reconfiguration actions: *add*, *migrate*, *replace*, *link*, *unlink* and *remove*.

1. *Add.* Adds a new component SWC_{new} to the current architecture by taking its used and provided ports, constraints and implementation. As the SWC_{new} is a stateful this action initializes the execution state with default values.(See Algorithm 2).

2. *Replace.* The replacement can be classified into two categories given the required need. *2.1 Upgrade*–changes the behavior of an existing component with an improved version, the SWC will have the same interface, WCET, communication pattern temporal behavior, but has an improved internal behavior (runnables). *2.2 Update*– which implies adding a new functionality. The new functionality needs to be stored first in memory and as well as its communication pattern and the new version of runnables that will communicate with the new added functionality. The update is more difficult than the upgrade (see Algorithm 4).

3. *Link.* Creates a binding from a provided port of a SWC to a used port of another SWC.i.e.,
   $link(SWC_{ri}, SWC_{rj}, P_{ri}^{out}, P_{rj}^{in})$ a connection is created between the output port of the SWC_{ri} and the input port of the SWC_{rj}.

4. *Unlink.* Disconnects a binding from the current architecture by using the same parameters as *link*.

5. Migrate. The SWC is moved from its current location (ECUs) to a new location, we consider a migration as a special case of SWC replacement in which the version of the component doesn't change (See Algorithm 3)

6. *Remove.* Removes an existing component from the current architecture by taking the name of the component. This action first removes the SWC execution state and then the respective SWC is deleted (see Algorithm 5).

5.1.7 Notations

The notation introduced here will be used throughout this chapter as well as the rest of the manuscript.

Symbol	Description
r_i	i^{th}runnable
O_i	i^{th}operational chain
$U_i^{ECU_j}(h)$	Probability of job j has an execuion timeof h on ECU_j
$C_i^{ECU_j}$	The WCET of hard task on ECU_j
D_i	The deadline of hard task
δ_i	soft deadline
$QoS(\delta_i)$	The probability of meeting the deadline δ_i
T_j	The period of soft and hard tasks
Q_j	The bandwidth server
Q_j/T_j	The utilisation of soft tasks
C_j/T_j	The utilisation of hard tasks
τ_i	i^{th}periodic task
τ_r^*	reconfiguration (sporadic)task
C_r^*	WCET of τ_r^*
D_r^*	relative deadline of τ_r^*
P_r^*	Period of τ_r^*
$U_{RQ}^{ECU_j}$	The utilization required for the reconfiguration

Table 5.2 Main notation used throughout this chapter.

5.2 ASLA's Algorithms

5.2.1 ASLA's Task Allocation Algorithm

ASLA's mapping manager uses Algorithm 1 to deploy tasks on ECUs. O-TSMBA uses the application description (initial system configuration file-AUTOSAR XML file) as an input. This algorithm is executed whenever there is a need to compute a new deployment of tasks on ECUs. O-TSMBA produces the following outputs: (1) *The mapping*, that is where each of the tasks should be mapped to, and (2) *The utilization*, that is how much processor utilization we should allocate to these soft tasks, processor utilization for hard tasks is fixed. So that the deadlines for all the hard tasks are satisfied, even when there are faults and the probability of meeting the deadlines for the soft tasks is maximized. In order to handle tasks dependencies, our algorithm tries to allocate all corresponding SWCs of an OP as a single SWC (or task) when possible, Else, it splits these consolidated SWCs when necessary. O-TSMBA starts by combining SWCs as part of the same OPs

(whether it is an application with soft or hard real-time requirements) (Lines 1- 12). The OPs with hard real-time requirements are considered before the OPs with soft real-time requirements and the combined SWCs are ordered on decreasing order of their utilization (total utilization of OP) $\mathsf{C_i}/\mathsf{T_i}$ for hard and $\mathsf{EU_{c_i}}/\mathsf{T_i}$ for soft, where $\mathsf{EU_{c_i}}$ represents the Expected Utility of each ECU (Line 13).

ALGORITHM 1: O-TSMBA (Γ^c, ECUs)

1: $\Gamma^c = \phi$
2: **for** $i = 1....n$ **do**
3: **if** $\mathsf{SWC_i} \notin \Gamma^c$ **then**
4: Find a composite $\mathsf{SWC_j}; \mathsf{SWC_j} \in \mathsf{Dep(SWC_i)}$
5: $\Gamma^c_j = \Gamma^c_j \bigcup \{\mathsf{SWC_i}\}$
6: **else**
7: $\Gamma^c_{n(\Gamma^c)} = \{\mathsf{SWC_i}\}$
8: $\Gamma^c = \Gamma^c \bigcup \Gamma^c_{n+1(\Gamma^c)}$
9: **end if**
10: **end for**
11: Sort Γ^c in descending order; start with $\Gamma^c \in \{OP_{hard}\}$and after $\Gamma^c \in \{OP_{soft}\}$
12: $\mathcal{S}^\circ$ = Initial Solution($\mathsf{\Gamma^c}$, ECUs)
13: $\mathcal{S}^{current} = \mathcal{S}^{best} = \mathcal{S}^\circ$
14: $Cost^{best}$ = CostFunction($\mathcal{S}^\circ$)
15: $TabuList = \phi$
16: **for** max_iter iterations **do**
17: NS = Generate Neighborhood($\mathcal{S}^{current}$)
18: $\mathcal{S}^{current}$ = Select Solution(NS)
19: **if** CostFunction($\mathcal{S}^{current}$) < $Cost^{best}$ **then**
20: $Cost^{best}$ = CostFunction($\mathcal{S}^{current}$)
21: $\mathcal{S}^{best} = \mathcal{S}^{current}$
22: **end if**
23: $TabuList = TabuList \bigcup \mathcal{S}^{current}$
24: **end for**
25: return $\mathcal{S}^{best}$

A Tabu search algorithm takes the application (i.e., the set of OPs) and the ECUs as input and produces a solution "S" consisting of the allocation χ for all the tasks and a set of bandwidth values Q for all the tasks with soft real-time requirements. (Take note that we assume that a runnable is represented by a task and runnables to tasks mapping exists before O-TSMBA is used similarly to AUTOSAR). Once having the OPs with their timing requirements and the architecture, we will have an initial solution in which the tasks are mapped such that their utilization is evenly distributed among the ECUs (for the soft tasks we consider the Average utilization $\mathsf{AET}/\mathsf{T_i}$.). The bandwidth for the soft tasks is allocated to a value equal to their AET (Line 14).

This initial solution can either be schedulable or not. A schedulable solution is the one that satisfies Liu & Layland utilization test for EDF [78], to determine if the task

set combined of hard tasks and CBS servers is schedulable (see Definition 6 in Chapter 2 Section 2.1.4) i.e.,

$$\sum_{\forall \tau_i \in \mathsf{OP_{Hard}} \cap \chi(\tau_i) = \mathsf{ECU_j}} \frac{\mathsf{C_j}}{\mathsf{T_j}} + \sum_{\forall \tau_i \in \mathsf{OP_{Soft}} \cap \chi(\tau_i) = \mathsf{ECU_j}} \frac{\mathsf{Q_j}}{\mathsf{T_j}} + {\mathsf{U_{RQ}}}^{\mathsf{ECU_j}} \leq 1.$$

The total utilization which needs to be bounded by 1 is the sum of utilization of: the hard tasks, soft tasks and the reconfiguration task. For hard tasks it is simply, $(\mathsf{C_i}/\mathsf{T_i})$ where $\mathsf{C_i}$ is the execution time including the reconfiguration overhead, for soft tasks it is $\mathsf{Q_j}/\mathsf{T_j}$, and for the reconfiguration task, the utilization is the worst-case utilization needed to reconfigure tasks ${\mathsf{U_{RQ}}}^{\mathsf{ECU_j}}$. The neighborhood of the current solution is generated using design transformation (moves) that changes the current system implementation (Line 19). The neighborhood can be very large, thus we consider a limited number of neighboring solutions, called candidate set. Similarly to the original TSMBA. Two moves are considered in the mapping algorithm (i) *Mapping move* (Mm) and (ii) *Bandwidth Moves* (BM). In (i) the mapping for the tasks is changed by selecting randomly a set of consolidated SWCs from a randomly selected ECU, tries to allocate them as a single SWC and allocate them to another ECU also selected randomly. The cost for doing this operation is given to O-TSMBA to evaluate the solution. In (ii) BM, the bandwidth of tasks is changed also by selecting randomly consolidated SWCs with soft real-time requirements from randomly selected ECUs. In O-TSMBA algorithm all the visited solutions are maintained by the tabu search in a *TabuList* to avoid revisiting them again (Line 17). All the solutions that have been already visited are marked as Tabu and are put in *TabuList*, this list is updated in (Line 25). The *non-tabu* solution with the minimum cost is chosen and exploration continues. The selected solutions (Line 20) are the ones which minimize the following *cost function:*

$$\begin{aligned} &\sum_{\forall \mathsf{ECU_i} \in \mathsf{ECUs}} max(0, \mathsf{U_{ECU_i}} - 1) \times \mathsf{w_{penalty}} \\ &+ \sum_{\forall \mathsf{r_i}:\mathsf{R}(\mathsf{r_i}) = Soft} (1 - \mathsf{QoS}(\mathsf{r_i})) \times w_i(\mathsf{r}_i, \mathsf{ECU_j}) \end{aligned} \tag{5.3}$$

where $w_{penalty}$ corresponds to a very large penalty added to the cost of a solution in case of hard real-time tasks are not scheduled (in other words the utilization of the ECU's processor is greater than one). In case the hard tasks are schedulable the first is 0 and the second term of the cost function is the maximization of the QoS of the soft tasks. Different weights $w_i(\mathsf{r}_i, \mathsf{ECU_j})$ can be assigned to the soft task. These weights represent how important is to satisfy the QoS of a given task when being mapped on $\mathsf{ECU_j}$. Similarly to TSMBA, our algorithm iterates the procedure until a bound number of iteration max_{iter} is reached which is given by the designer (Line 18).

5.2.2 ASLA's Task Adaptation Algorithm–The Case for Adding a New Task

As we said in chapter 4 section 4.1, the RM is a sporadic task that gets triggered upon the reception of an adaptation triggers (requests for adding new tasks, requests for migrating failed tasks/and or failed ECUs, replacement of tasks with an improved version and removing tasks). ASLA's reconfiguration manager uses Algorithm 2 for adding a new application at runtime (i.e., TSeRBA). TSeRBA uses the following inputs: (1) a solution schedulable and tagged optimized obtained from mapping manager using O-TSMBA algorithm and (2) the adaptation triggers from the monitor (in this case: *request for adding a new application*). The output of TSeRBA is a new system configuration which needs to be schedulable. To deal with the current reconfiguration, the algorithm starts by finding the target ECU to host the new application (Line 3- 8). So it maintains an ordered list of ECU candidates, and the one which gives the best QoS is selected as the target ECU. This QoS is the probability of meeting the deadline for soft tasks and it depends on the allocated bandwidth $\mathsf{Q_i}$ [3], then if the mapping is still feasible (Line 10) and the time2adapt is reached, the new task is simply added in the target ECU (Lines 11, 12, 13 and 20) (see Definition 13). Thus, the RM loads, instantiates and connects a new component safely without affecting other components, changes the network schedule, and dynamically finds and binds to the correct interface as no SWC knows how to use this new component and the network schedule has to be reorganized to accommodate the communication pattern of this component (Lines 12, 13 and 14). If the RM cannot analyze and schedule this new component i.e., the mapping is not feasible (see Definition. 17) (Line 15), in that case the bandwidth Q associated with soft tasks on that ECU is decreased in proportion do their expectations, and then the new task can be mapped on the ECU. A completely new resource is added to host the new task if the mapping is not feasible even when we adjust the bandwidth (Line 16). For our example task system described in section 5.1.2, upon the receipt of an adaptation trigger to add new task to the system; TSeRBA tries to find the best target ECU to host this new task (as shown in Figure 5.9). ECU2 is selected as a candidate for adding this new task since it gives the best QoS. Furthermore, in Figure 5.9, the compressed task $\tau_i(C_i, T_i)$ is $\tau_8(15, 35)$ since it is the only hard task on ECU2. The only new task in Fig 5.9 is $\tau_{new}(1, 4)$, while the other unchanged tasks are: $\tau_1(9, 150), \tau_3(12, 190)$ *and* $\tau_4(19, 300)$. Suppose the release time of the current instance $t_0 = 0$ and the time to accommodate the new request (i.e., inserting a new task) $t_r = 2$, $\tau_8(15, 35)$ is compressed into $\tau_8(15, 70)$, thus using Definition 13, we have $C_i(t_r) = 2$ and $U_8 + U_1 + U3 + U_4 = (15/35) + (9/150) + (19/300) + (12/190) = 0.61$. $U_{new} = U_8 - U_8^{'} = (15/35) - (15/70) = 1/5$. From [59] and according to Definition 13, we get time2adapt $= 35 - 2/(1/5) = 25 > t_r$.It is shown in the Figure 5.9 that no deadline is missed.

ALGORITHM 2: TSeRBA for adding a new task

1: As soon as the add request is triggered
2: **for** all $\mathsf{SWC_i}$ such that $\chi(\mathsf{SWC_i}) \in \mathsf{ECUs}$ **do**
3: $QoS_{best} = 0$
4: **for** all $\mathsf{ECU_j} \in \mathsf{ECUs}$ **do**
5: $\mathsf{QoS_i} = GetQoS(\mathsf{SWC_i})$; $\mathsf{QoS_{currECU_j}} = \sum_{\forall \mathsf{SWC_k}:\chi(\mathsf{SWC_k})\in\mathsf{ECU_j}} GetQoS(\mathsf{SWC_k})$
6: **if** $QoS_{curr} > QoS_{best}$ **then**
7: $\mathsf{ECU_{target}} = \mathsf{ECU_j}$; $QoS_{best} = QoS_{curr}$
8: **end if**
9: **if** Feasible Mapping **then**
10: When $(\mathsf{time2adapt} = \delta)$
11: load the $\mathsf{SWC_i}()$;
12: link $\mathsf{SWC_i}$ to the correct interface();
13: Change the network schedule();
14: **else if** $Adjust\ Qs\ Proportionally(\mathsf{SWC_i}, \mathsf{ECU_j})$ **then**
15: $recompute\chi$
else
$\mathsf{ECUs} \leftarrow \mathsf{ECUs} + \mathsf{ECU_{new}}$
end
16: **end if**
17: $Undo\ Adjust\ Qs(\mathsf{ECU_j})$
18: **end for**
19: $Add(\mathsf{SWC_i}, \mathsf{ECU_{target}})$
20: **end for**

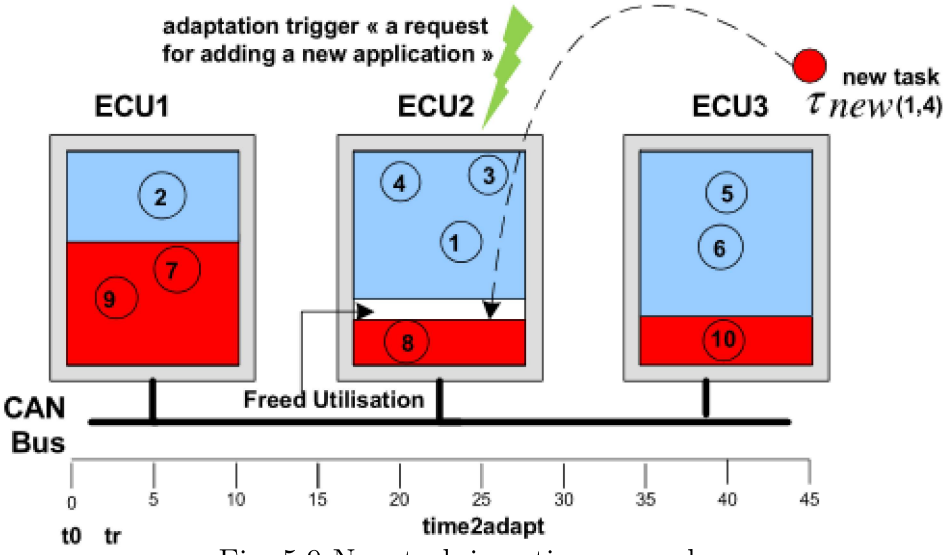

Fig. 5.9 New task insertion example.

5.2.3 ASLA's Task Adaptation Algorithm–The Case for Migrating Tasks

Similarly to Algorithm 2. ASLA's reconfiguration manager uses Algorithm 3 to migrate tasks between ECUs. In this case, TSeRBA tries to find first the target ECU (Line 3- 11) to host

the failed tasks on ECU3 (i.e.,$\tau_5, \tau_6, \tau_{10}$) (as shown in Figure 5.10). ECU1 is determined an infeasible solution since the combined utilization on ECU1 would exceed 100% if τ_{10} were allocated on ECU1 already hosting τ_2, τ_7, τ_9 and τ_{10} $(0.32+0.32+0.4+0.186 = 1.23)$ (Line 14). TSeRBA tries to decrease the server bandwidth of the soft tasks allocated on ECU1, however again no schedulable task set on ECU1 is found (Line 15) and ECU1 is eliminated as a choice to host the failed tasks.

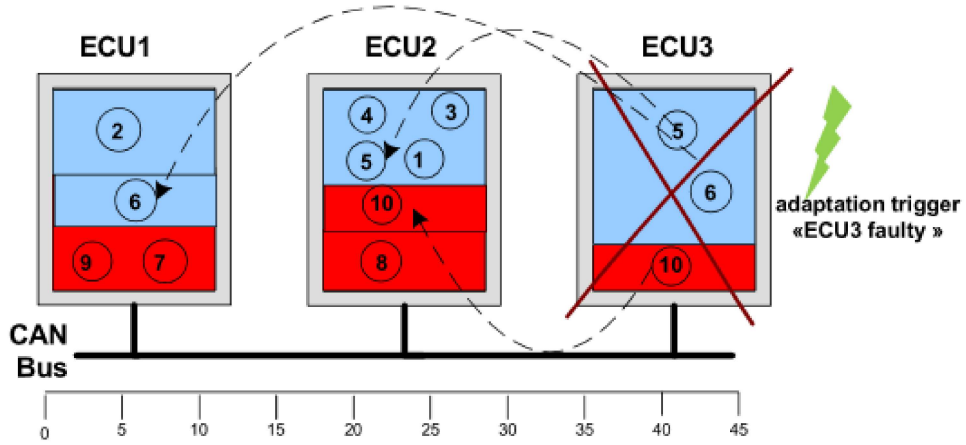

Fig. 5.10 Tasks migration example.

The set of candidate ECUs is updated (Line 5) and TSeRBA is attempted again and selects ECU2 as the target migration node. Since the total processor utilization available for the soft tasks on ECU2 if τ_{10} is migrated on it is equal to $1 - (0.49 + 0.32) = 0.26$. Therefore, the utilization desired by τ_{10} can be made available by decreasing the bandwidth of the soft tasks on ECU2 in proportion to their expected utilities EUs. The values of EU for the allocated soft tasks (i.e., τ_1, τ_3, τ_4) on ECU2 are: 28, 29 and 31, respectively. For instance, the available utilization and the changed bandwidth for τ_1 is computed as follows: $U_{\tau_1} = (28/(28 + 29 + 31)) * 0.26 = 0.09$, $Q^{new}_{\tau_1} = 0.09 * 150 = 12$. In the same way we compute U_{τ_i} and $Q^{new}_{\tau_i}$ with $i \in \{3, 4\}$.Note that if the task to migrate cannot be allocated to any of ECU1 or ECU2, thereby an additional ECU is required (Line 17).

Note that ASLA's algorithms for replacing and removing tasks are given in Appendix 2 Section B.0.3.

ALGORITHM 3: TSeRBA for migrating tasks

As soon as the failure request is triggered (i.e a set of failed ECUs)
for all $\mathsf{SWC_i} \in \Gamma^c$ such that $\chi(\Gamma^c, \mathsf{ECU_{faulty}})$ **do**
  $QoS_{best} = 0$
  get all the utilization of the Op mapped on $\mathsf{ECU_{faulty}}$
  $U_i = getUtilization(\mathsf{SWC_i})$; $\mathsf{SWC_i} \in \Gamma^c$
  for all $\mathsf{ECU_j} \in \mathsf{ECUs}$; $\mathsf{ECU_j} \notin \mathsf{ECU_{faulty}}$ **do**
    Find the best-fit ECUs(which is the one that gives the best QoS)
    $QoS_{currECU_j} = \sum_{\forall \Gamma_k : \chi(\Gamma_k) \in \mathsf{ECU_j}} GetQoS(\Gamma_k)$
    if $QoS_{curr} > QoS_{best}$ **then**
      $ECU_{target} = ECU_j$; $QoS_{best} = QoS_{curr}$
    end if
    Sorts Γ^cin descending order of their utilization, Γ^c with hard requirement first and after Soft.
    if Feasible Mapping **then**
      Migrate($\Gamma^c, \mathsf{ECU_{target}}$)
    else if Picks the biggest Γ^c_{ksoft} ;$\Gamma^c_{ksoft} \in \mathsf{ECU_{target}}$; $AdjustQsProportionally(\mathsf{\Gamma^c}, ECU_j)$ **then**
      $recompute\chi$
      else
| $ECUs \leftarrow ECUs + ECU_{new}$
      end
    end if
    $UndoAdjustQs(ECU_j)$
  end for
end for

5.3 Summary

In this Chapter, we have considered the theoretical and the technical aspect of ASLA framework described in Chapter 4. More specifically, we have formalized and solved the problem faced when tackling runtime adaptation in automotive real-time systems. We have shown how the runtime support to AUTOSAR is realized by TSeRBA algorithm "Tabu Search Reconfiguration and Bandwidth Allocation" as it allows the dynamic allocation of Software components(SWCs) with both Hard and Soft real-time constraints as well as supports the insertion and the migration of SWCs at runtime. In the next Chapter, the theoretical results will be illustrated with examples and test cases.

Chapter 6

Implementation and Evaluation

In previous chapters, we described the theoretical aspects of our approach towards an adaptive AUTOSAR, ASLA. In this chapter, we present the implementation of ASLA components, namely Reconfiguration & Mapping managers. Most importantly, this implementation allows us to demonstrate the feasibility of our approaches through extensive evaluations, which we detail throughout the chapter. Due to the complexity of ASLA scheduling analysis, we have first implemented ASLA's algorithms using Java-based implementation (simulation) and then we moved a part of this implementation on a real platform (more specifically, ASLA's migration algorithm).

6.1 ASLA Reconfiguration Model

This section describes the implementation of ASLA reconfiguration model that reconfigures the system components. Figure 6.1 illustrates a class diagram of the reconfiguration model. The core functional system of ASLA are ReconfigurationManager as well as the ReconfigurationAlgorithm sub-classes. The ReconfigurationManager provides the entry point for adaptations. The ReconfigurationAlgorithm classes implement the TSeRBA algorithm with CBS and EDF scheduling algorithms discussed in Chapter 5.

– ReconfigurationManager a singleton class that is deployed in every ASLA reconfiguration model instance and is created via a static getInstance() operation. This static operation calls the class's constructor of the ReconfigurationManager class, in which fields, such as the reconfiguration scheduling algorithm used are instantiated. The current implementations provide an operation that set the algorithm used manually. The ReconfigurationManager maintains a list of all the components that are deployed in CompList as well as the currently used ReconfigurationAlgorithm algo. Additionally, it contains a list of adaptation triggers triggersList that have occurred and that still need to executed. The reconfiguration manager implements the CompFactory interface that is responsible for the creation and the deletion of

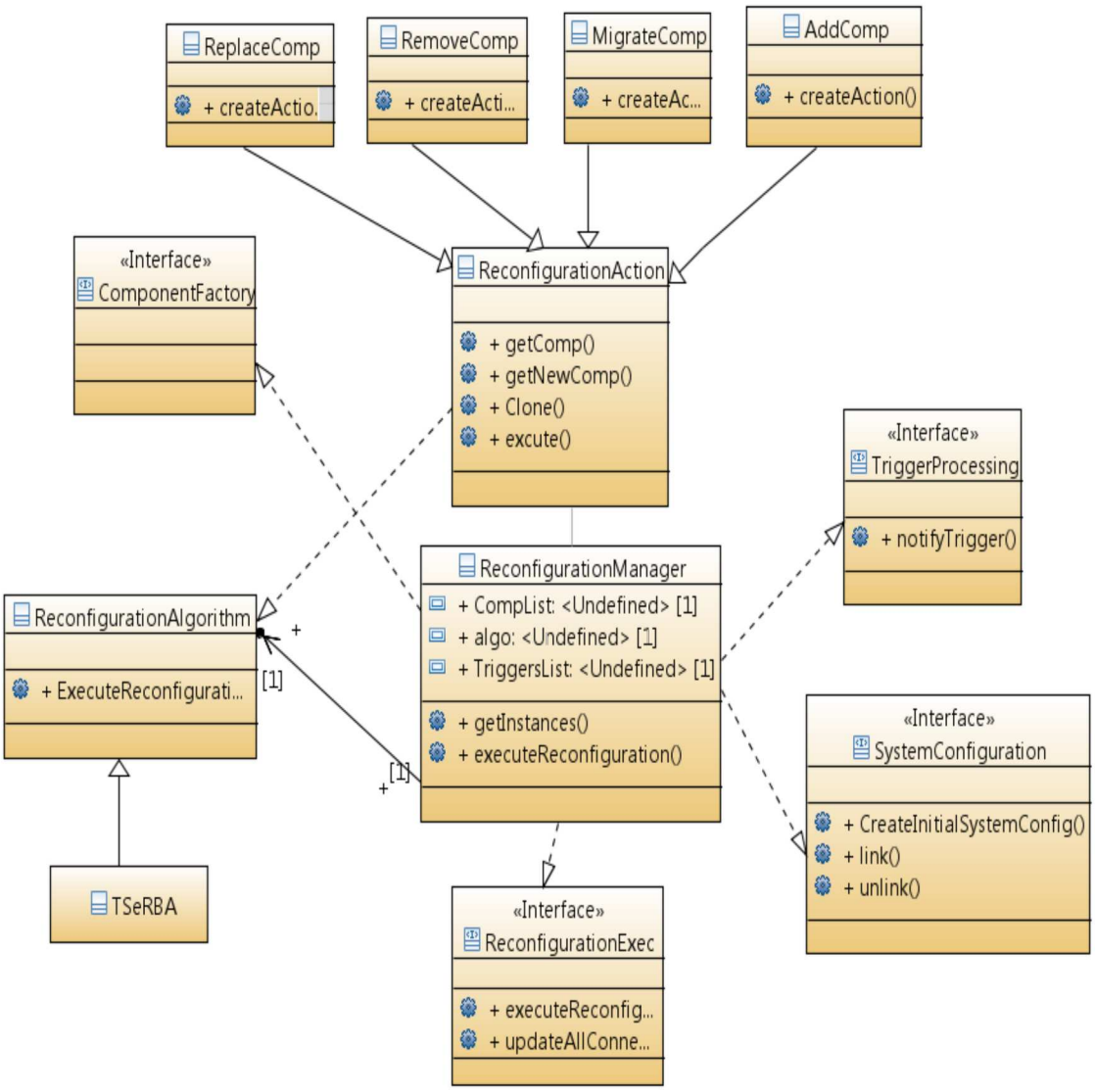

Fig. 6.1 ASLA reconfiguration model implementation.

the components (this is done manually). Additionally the reconfiguration manager, provides the following interfaces :

- SystemConfiguration Interface

  This interface provides operations to change the current system configuration and to execute reconfiguration operations. These include createInitialSystemConfig() method to create an initial system configuration, this system configuration is generated by matlab script GENERATE.INIT.Config.m (see code in Appendix A) and methods to connect(link()) and disconnect(unlink()) components to and from the current system configuration.

- Reconfigurationexcution Interface This interface provides operations that deal with the actual execution of a reconfiguration sequence. The method executeReconfiguration() executes all the reconfiguration actions. The method updateAllConnections()

Listing 6.1 Implemented reconfiguration action classes.

```
class AddComp()
class RemoveComp ()
class ReplaceComp()
class MigrateComp()
```

retrieves all dependent components, i.e., components that are connected to the replaced component via an input port, and updates their input ports accordingly. It is called, for example, when a reconfiguration action needs to be revoked.

- TriggersProcessing Interface This interface provides the notifyTrigger() method that deals with the processing of incoming triggers. This method is called by any component in the system configuration when it receives an incoming reconfiguration triggers. This then starts the reconfiguration scheduling algorithm referenced by the filed algo, which is of type ReconfigurationAlgorithm.

– ReconfigurationAlgorithm. The class ReconfigurationAlgorithm serves as the abstract base class for TSeRBA reconfiguration scheduling algorithms. This class has an abstract executeReconfiguration() method. ASLA also contains the implementation of CBS and EDF scheduling algorithms.

– ReconfigurationActions

All reconfiguration actions extend from the abstract base class ReconfigurationAction. This class contains the unique identifier of the component on which the reconfiguration action is executed. Methods that need to be implemented by the concrete sub-classes include accessor methods to retrieve the affected component, such as getComp() or methods to retrieve the new component getNextComp(). Also, a concrete sub-class must implement the abstract method execute(), which contains the actual implementation of the respective action. A more detailed description of how the execute() method is realized for the different actions is presented next subsection.

Listing 6.1 summarizes the concrete reconfiguration action classes, which represent the available reconfiguration actions identified in Chapter 5, Section 5.1.6. Note that the reconfiguration action classes link and unlink are only indirectly realized as actions, since they are part of the reconfiguration manager, or more specifically, part of the reconfiguration manager's systemconfiguration interface.

– Reconfiguration actions at runtime. A reconfiguration is triggered by calling the executeReconfiguration() method of the reconfiguration manager. Reconfigurations can be either triggered explicitly by a reconfiguration designer or can be

triggered by changes in the system conditions. However, the reconfiguration manager is oblivious to the cause of the trigger.The executeReconfiguration() method gets as parameter the list of reconfiguration actions to execute. The AddComp reconfiguration action class realizes the execute() method by calling the static createComp() method of the reconfiguration manager, with the type identifier of the component to be added as parameter. The RemoveComp action class contains the unique identifier of the component to be removed. The implementation of execute() results in a call to the removeComp() method of the reconfiguration manager, that then executes the actual object removal. Replace and Migrate reconfiguration actions contain the identifier the component to be replaced and the type identifier of the replacement component. In both cases, the execute() method first creates the component, which replaces the old component, by calling the static createComp() method of the reconfiguration manager. The reconfiguration actions link and unlink are realized by the reconfiguration manager. link is realized by calling the link() method of the reconfiguration manager, which takes as parameters the two components to be connected and their input and output ports. Likewise, the unlink() method of the manager takes as parameters the components to be disconnected and their ports. Take note that, in our current implementation we were interested in implementing ASLA's algorithms, having a complete implementation of ASLA classes is left as future work.

6.2 Simulation Based Evaluation Environment

ASLA's algorithms presented in Chapter 4 and 5 are implemented using Java and MATLAB[1] on a 2.3 GHz Intel Core i5 machine running Ubuntu Linux 10.04 (8G RAM). It is noteworthy to mention that we chose Java as a programming language given its robustness, portability, large suit of available libraries and the widespread community of Java developers. Nonetheless, we acknowledge the fact that Java and related frameworks employed while implementing our solution are mostly suitable for systems with high processing and computation capabilities. However, the solution can be ported to other languages (e.g., C, C++) that are suitable for real-time systems. The central part in the implementation of the algorithms described in Chapter 5 (i.e. TSeRBA) is the cost function (see equation 5.3). In the next section, we describe the implementation details:

The cost function is implemented as a Java function. The input of the Java function is a mapping of tasks to the ECUs and their allocated budgets (Q_i values). It outputs the value of the cost function. The TABU search heuristic searches through the solution space of such values of cost function for various mappings and tries to find the best mapping and

[1] https://fr.mathworks.com/products/simulink.html

best budget allocations. The ECUs, tasks (both hard and soft real-time) are represented on disk in an XML file (see Listing 6.2).

The hard real-time tasks are characterized by a task id, a worst case execution time, a period and a deadline. The soft real-time tasks are characterized by a task id, the PDFs of computation times (multiple PDFs corresponding to various mappings), the period of the tasks, a deadline which should be satisfied to give the best possible quality of service and a weight which represents how important is it to satisfy the QoS of this task. The weight is a fraction with values between $[0 - 1]$.

We have used Apache commons maths library to implement various routines (all the theory about cost function calculation described in Chapter 5 Section 5.1.4). Various steps in the calculation of cost function are given below:

- **Step 1**–*Reading XML file* The processing elements, tasks (both hard and soft real-time) are read from the XML file and they are converted to an in-memory representation. This in-memory representation is constructed using the Interface RealMatrix as defined in Apache Commons api.org.apache.commons.math.linear. This interface defines a real-valued matrix on which some basic algebraic operations can be performed.

- **Step 2** – *Constructing the Marcov matrix*

  The Markov matrix M' is constructed in two steps. First, the worst possible length of the queue is found out. In our case we have restricted the maximum number of jobs to 100. This computed length is the truncation point for the infinite Markov matrix. Thus, we generate the Markov matrix having dimensions equal to the worst possible length of the queue.

- **Step 3**–*Solving the Markov matrix for stationary solution.*

  We are generating the stationary solution by raising M' to higher and higher powers until all columns are almost identical. A variable measures the error by calculating the quadratic distance between the elements of the first and the last column. The iterations are stopped when this measured error is less than a certain pre-decided value. This value corresponds to the desired level of accuracy. Any column can be used as a steady state vector since they are all equal and correspond to a stationary solution. The steady state vector is then normalized so that the sum of the elements becomes 1.

- **Step 4**–*Generating the Cumulative Distribution Function (CDF)*

  The CDF $QoS(\delta_i)$ is then calculated from the given steady state vector using the method described in Chapter 5.

Listing 6.2 Example of application description.

```
<?xml version="1.0" encoding="UTF-8"?>
 <input>
<PARAMS>
 <EXPNO> 1 </EXPNO>
 <PDFLEN> 50 </PDFLEN>
 <NC> 100 </NC>
 <TABULEN> 100 </TABULEN>
 <MAX_ITER> 8000 </MAX_ITER>
 <LAST_IMPRV_CNT> 50 </LAST_IMPRV_CNT>
 <BUS_BW> 2.000000 </BUS_BW>
 <TOT_FAULTS> 2 </TOT_FAULTS>
 </PARAMS>
<ECU> <ID> ECU1 </ID> </ECU>
<ECU> <ID> ECU2 </ID> </ECU>
<task_SRT>
 <TID> 1 </TID>
 <CPDF>    0.007810    0.012313    0.003486    0.015320    0.016111    0.007229
       0.013019    0.019692    0.012083    0.018671    0.020155    0.022963
     0.017444    0.024728    0.018166    0.019347    0.026893    0.024261
     0.027072    0.025218    0.022833    0.027270    0.030258    0.023530
     0.019951    0.025495    0.023168    0.029968    0.021092    0.025485
     0.021558    0.019881    0.019079    0.022833    0.025172    0.026560
     0.020570    0.022825    0.017249    0.017169    0.018913    0.022026
     0.018240    0.016288    0.018930    0.021573    0.016870    0.013552
     0.012540    0.018430 </CPDF>
 <EXP> 26.240276 </EXP>
 <CPDF>    0.000000    0.007881    0.007694    0.014759    0.013345    0.016354
       0.010878    0.014253    0.016396    0.020932    0.010923    0.012577
     0.013794    0.023443    0.022570    0.019238    0.023526    0.017286
     0.027132    0.026045    0.020880    0.024943    0.027748    0.022039
     0.025154    0.024189    0.019146    0.020508    0.024245    0.028744
     0.029168    0.022293    0.023442    0.018908    0.020788    0.025859
     0.023639    0.019775    0.019107    0.027283    0.020515    0.021385
     0.018202    0.023872    0.018638    0.015002    0.022644    0.024630
     0.020156    0.015272 </CPDF>
 <EXP> 27.304875 </EXP>
 <TPDF> 150 </TPDF>
 <DDLN> 124 </DDLN>
 <WGHT> 1 </WGHT>
 <CHKP_OVER> 2 </CHKP_OVER>
 <NUM_CHKP> 4 </NUM_CHKP>
 <SFC> 1 </SFC>
 </task_SRT>
 <task_HRT>
 <TID> 4 </TID>
 <WCT> 6 </WCT>
 <PER> 30 </PER>
 <DDL> 30 </DDL>
 <CHKP_OVER> 1 </CHKP_OVER>
 <NUM_CHKP> 3 </NUM_CHKP>
 <SFC> 1 </SFC>
 </task_HRT>
<task_HRT>
 <TID> 5 </TID>
 <WCT> 10 </WCT>
 <PER> 50 </PER>
 <DDL> 50 </DDL>
 <CHKP_OVER> 1 </CHKP_OVER>
 <NUM_CHKP> 3 </NUM_CHKP>
 <SFC> 1 </SFC>
 </task_HRT>
 </input>
```

– **Step 5**–*Cost function calculation*

After we have got $QoS(\delta_i)$ for all soft real time tasks ($\tau_i^s = 1, 2, ...M_s$), the value of the cost function is calculated by multiplying the task's weights with their probabilistic guaranties $QoS(\delta_i)$and adding them up.

Take note that the initial system configuration and the description of the system given to TSeRBA algorithm are automatically generated by Matlab code (see Appendix A).

6.2.1 Empirical Evaluation

In this section, we present an empirical investigation, examining the effectiveness of our analysis techniques and TSeRBA scheme itself. We report on a set of experiments undertaken for tasks set with both hard and soft tasks. We will first introduce an experiment designed to compare the performance of TSeRBA and the non-adaptation algorithm (i.e., TSMBA). Then, we will explore others experiments.These are sufficient to provide a clear evaluation.

Task set parameter generation. We performed a number of simulations by varying the number of ECUs E, the number of tasks n, the total utilization U and the task parameters: deadline D_i, period T_i, PDF, bandwidth Q_i and execution time C_i. We consider tasksets with 6, 10, 16, 26, 37...83 tasks. The number of ECUs to allocate these tasksets vary from 3 to 20 ECUs. For each one of these cases, we let the total utilization vary from 0.025 to 0.975. For each configuration (n, E, U), we generated 1000 task sets. The taskset parameters used in our experiments were randomly generated as follows: Task utilizations ($U_i = C_i/T$) were generated using the Uunifast algorithm [35]. Task deadlines D_i were assigned according to a uniform distribution, in the range $[C_i, T_i]$. Periods T_i were generated using a Log uniform algorithm [92]. The execution time WCET C_i for each hard task is accordingly computed using the generated T_i and U_i: $C_i = \lceil T_i U_i \rceil$. The server bandwidth Q_i is randomly generated with uniform distribution similar to [30]. The soft tasks utilization is ($U_i = Q_i/T$). The Server period P is computed as $P = Q_i/U_i$. The PDFs were generated using a Matlab script. These PDFs were generated using WCETs in the range $[3, 19ms]$ to match the shape of real-life benchmarks. The message lengths were assigned randomly within 1 to 4 bytes ranges. Since the bus bandwidth is assumed to be 10 Mbps, the transmission times on the bus are in between 1 to 4 ms. We are assuming that there can be at most 1 transient fault per execution segment of the application. We are assuming a hard task τ_i has to recover before the end of the next period, i.e., $2 \times T_i$. The checkpointing overhead and number of checkpoints for these tasks are between 1 to 3 ms and 2 to 8, respectively. We computed The number of preemptions $N^{pr}(\tau^j)$ that occur with TSeRBA to the j-th generated task sets τ^j in the time interval $[0, 1.5 * 10^4]$.

Experiments results. In the experiments presented below we performed the schedulability analysis test presented in Chapter 5 comparing the TSeRBA and TSMBA (the baseline approach presented in [99]) schemes. A first experiment has been carried out to measure the schedulability of the different tests at taskset utilizations. Figure 6.2 plots the percentage of tasksets generated that were deemed schedulable for a system with 8 ECUs and 33 tasks, with on average 50% of those tasks are hard. We observe that both algorithms the static and the dynamic one are capable to find some schedule solutions in all the cases. However, TSeRBA finds schedulable solutions much earlier than TSMBA. This is expected as TSeRBA tries to reconfigure (migrate) an already allocated tasks which may improve scheduling. It can be seen also, that both algorithms are able to find good solutions even if the utilization of the system increases, however TSeRBA has better performance than TSMBA when U $>$ 0.65 and TSMBA degrades much faster with increase in the utilization.

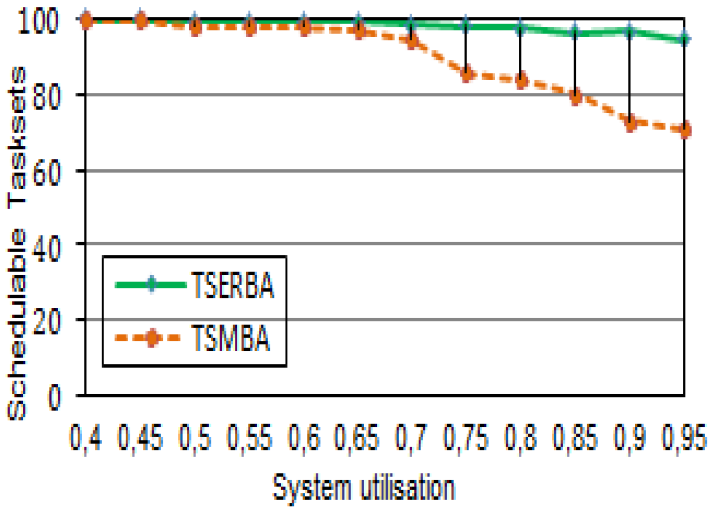

Fig. 6.2 Percentage of schedulable tasksets.

Figure 6.3 illustrates the result of a second experiment aimed at measuring the migration cost for different task set sizes. Clearly, the cost of migrating the task on our target ECUs linearly increases with the size of the tasksets. However, in Figure 6.3, the task migration cost also includes the computation cost of TSeRBA algorithm. From this figure, we can see that the task migration cost increases slowly with the taskset size when it is under 35. However, when the taskset size is bigger than 35, the task migration cost has a near linear relationship with the taskset size. The reason is that when the taskset size is small the task migration cost is dominated by the computation cost of TSeRBA algorithm. However, when the taskset size increase to a certain amount, the computation cost dominates the task migration cost.

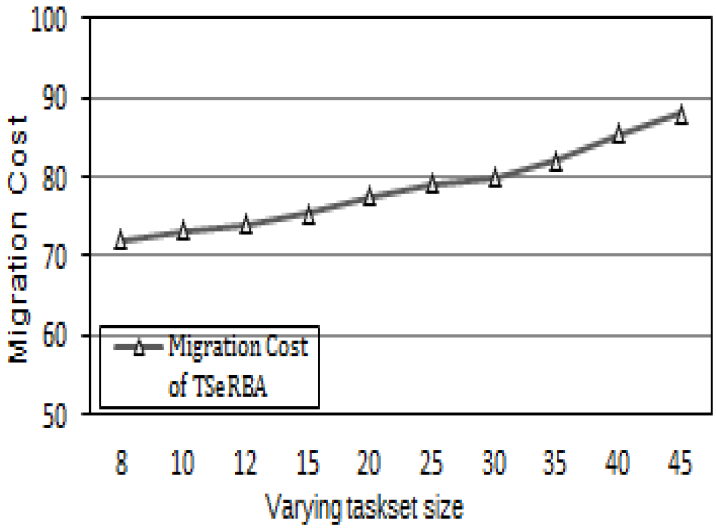

Fig. 6.3 Varying the number of tasksets.

6.2.2 Additional Experimental Results

For each considered configuration (n, E, U) described in Section 6.2.1; we have computed the average number of task preemptions as a function of the taskset utilization U, for all considered values of $n \in \{4, 8, 12\}$ (See Figure 6.4, 6.5 and Figure 6.6.). The average number of preemption $N^{pr}_{avg}(\tau^j)$ is counted as the sum, among all 1000 generated task sets, of the total number of preemptions experienced by each task set during interval$[0, 1.5 * 10^4]$, divided by the number of generated task sets (i.e., $\Sigma_{j=1}^{1000} N_{pr}(\tau^j)/1000$). From the above Figure. 6.4, it clearly appears that TSeRBA schedules tasksets with a significantly small number of preemptions, at all the system utilizations. This property becomes more evident considering tasksets composed of a large number of tasks. For n = 12, our algorithm has an average number of preemptions that is less than 15%. A well known property of EDF scheduling is that the number of preemptions in a given interval is bounded by the number of jobs in the same interval [37]. Therefore, increasing the number of tasks while keeping constant the total U, the number of tasks increases,as does the number of preemptions. In our experiments, the $N^{pr}_{avg}(\tau^j)$ with TSeRBA remains more or less constant while varying n. TSeRBA shows a very limited number of preemptions even at system utilizations close to 1; with the considered parameters, the $N^{pr}_{avg}(\tau^j)$ is never higher than $1.5 * 10^4$ preemptions every 10^4 time units, for all considered values of n .

We provide the number of migrated tasks as another metric of the migration overhead, which is reported in Fig 6.7. We have varied the number of faults from 1 to 3 faults (which can happen during one execution cycle), depending on the number of ECUs (from 3 to 20 ECUs) in the system, for taskset cardinalities of 10, 16, 21, 29, 33, 37, 49, 69, 80, 83 tasks. We observe that the number of tasks to migrate are similar regardless of which ECU fails. It implies that the entire workload of the system is kept quite well distributed over the available ECUs even after the failure.

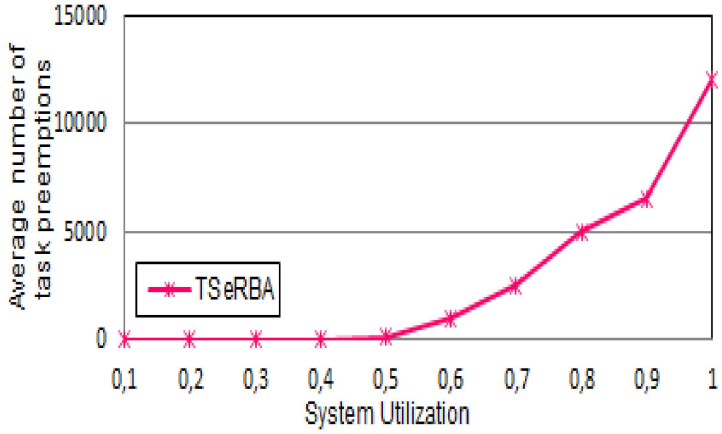

Fig. 6.4 Average number of preemptions for n=12.

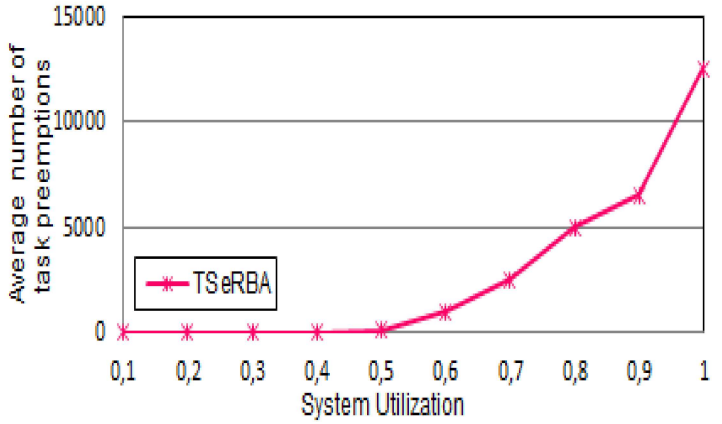

Fig. 6.5 Average number of preemptions for n=8.

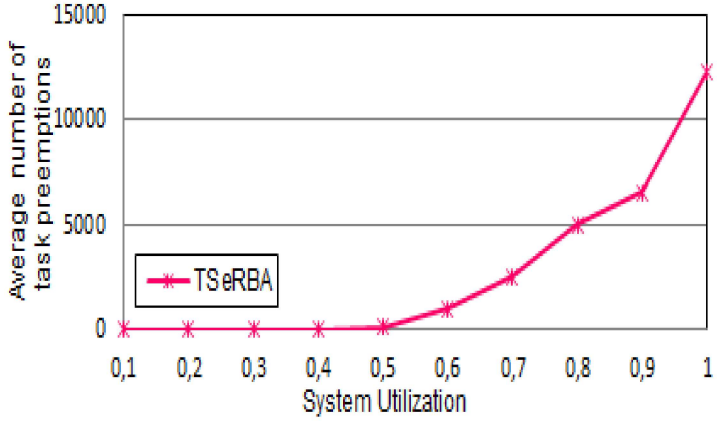

Fig. 6.6 Average number of preemptions for n=4.

6.3 Second Implementation– Ongoing Demonstrator Platform

In order to create a rapid prototype of ALSA, we built an experimental platform (in Figure. 6.10). Three ARM-based STM32FDiscovery boards are used as representatives for more powerful control units that are expected in the future. The scenario we would like to show is described as follows: “upon the receipt of an adaptation trigger that is,

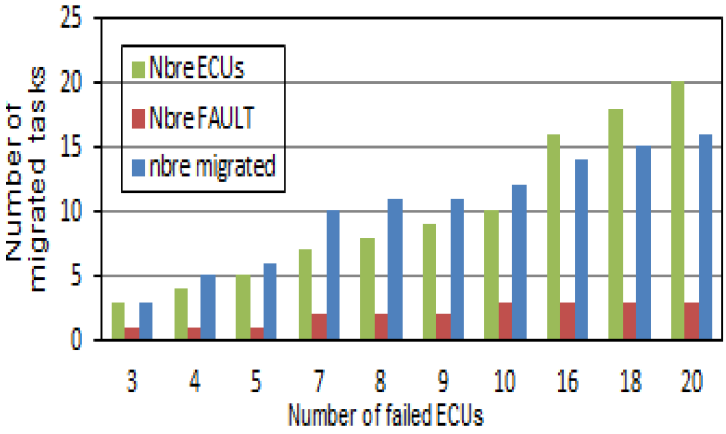

Fig. 6.7 Number of tasks migration.

one of the ECU fails, migrate tasks from this ECU to the other operational ECUs". This section describes the software architecture in detail.

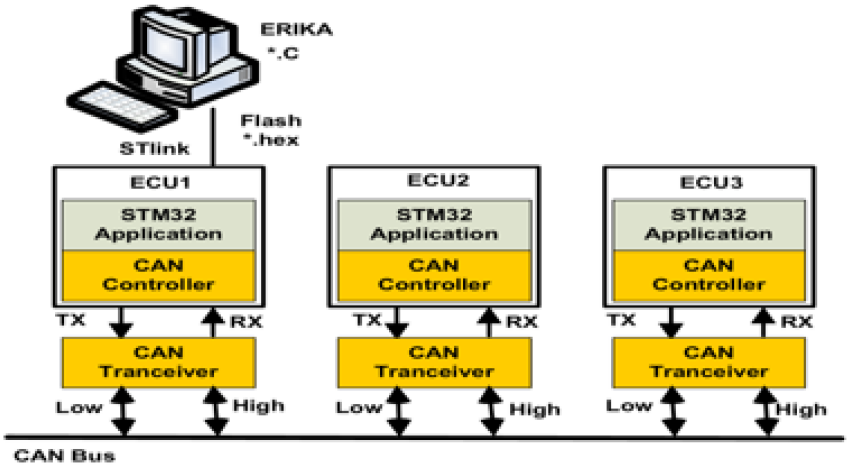

Fig. 6.8 Block diagram of the experimental platform.

6.3.1 Hardware

For the use case described above, we at least need 3 ECUs; one of the 3 ECUs acts as a system gateway and is connected to the PC using an USB cable. This ECU acts as the fault injection module and also to collect data from the system. The Fault in our case is "shutdown or restart of ECUs". This fault and the collected data are controlled using the PC interface. Figure. 6.8 shows the block diagram of our platform. For the demonstration purpose, we consider 3 applications with varying types of criticality namely, Steer-By-Wire (SBW) [41], Wiper Status (WS) and Door Locks (DL).

Steer-By-Wire application. Enables the electric steering of the vehicle by sensing the driving and steering wheel angles, calculating the intended wheel angles, and actuating the change of direction via motors to the front axis.

Wiper Status application. The wiper status application has three different modes: it can be off, constantly on or operated in an interval. The period of this interval depends on the amount of water on the windshield that is detected by a rain sensor.

Door Locks application. Enables automatic locking and unlocking of doors in the vehicle. In case of an accident the door lock needs to be unlocked without a tangible delay; therefore, a fast wake of a sleeping door module is necessary. This application takes as input the Door state request with four valid states (Close, Open, Lock and Unlock). This function decides whether to accept the change depending on its current state. In our current implementation, each application is implemented on a separate ECU and it contains one runnable, considering several runnables is left as future work. The different applications and ASLA have been flashed using ERIKA enterprise [49] and STLINK as follows: ECU1 and ECU2 host WS, DL respectively. Whereas ECU3 executes SBW application as described in Table 1.

Runnable	Task	Bound ECU	CPU Utilization	CAN msg	Msg Lenght (bytes)	Bandwidth(Q)
SBW-Runnable	SBW-task	ECU3	0.32	SBW, ECU3-health-msg	2	-
WiperStatus-Runnable	WS-Task	ECU1	0.9	WS, ECU1-health-msg	2	58
DoorLock-Runnable	DL-Task	ECU2	0.49	DL, ECU2-health-msg	2	91

Table 1: System tasks and Allocation

6.3.2 Software

The basic software running on the hardware includes the ERIKA-OS [49] operating system and generated code from RT-Druid [49]. The code generated conforms to AUTOSAR 4.x specification and produce the minimum required implementation to produce a working system. An OIL file for the configuration of ERIKA enterprise is also generated for each ECU. This generated code includes the necessary code modifications required as part of the runtime adaptation support described in the previous chapters. The GCC compiler for the STM32Discovery board is used along with ERIKA OS configuration tool to produce an executable for each ECU. The CAN message identifiers are generated for each message on the bus. The various Runnables, Tasks and Messages used in the experimental system are provided in Table 1. The runnable parameters like WCET, Period and ECU were provided by the system designer. Message sizes and CAN configuration (125kps) were also specified at the design phase. Specific CAN identifiers are assigned to specific ECUs. For example, if the WS application is running on ECU1, the CAN identifier assigned to its message is 101, etc.

6.3.3 Test Plan and Preliminary Results

In this section, we present how ASLA deals with the case of ECU failure through task-level adaptation as depicted in Figure.6.9.

Test Procedure

1. Switch on all the three ECUs.
2. Switch OFF ECU1 when one of the other operational ECUs has greater free CPU utilization
3. Capture the logs.

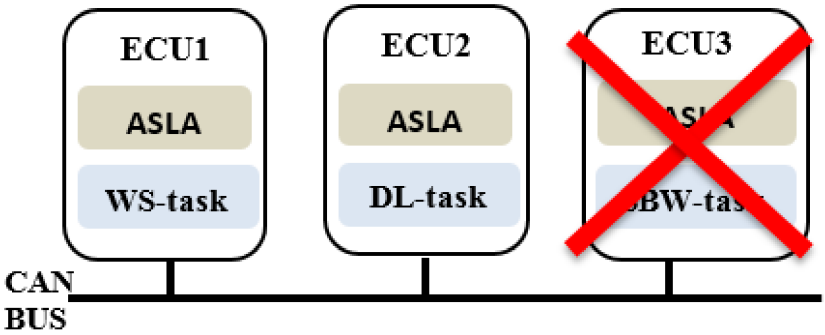

Fig. 6.9 Task migration example

Results

Upon the receipt of the adaptation trigger "ECU3 fails", the RM tries to find the target ECU to host the failed tasks on ECU3 (i.e., SBW task) (as shown in Figure. 6.9). ECU1 is determined as an infeasible solution since the combined utilization on the ECU1 would exceed 100% if SBW-task would be allocated on ECU1 already hosting WS-task. Even by reducing the bandwidth of the WS-task to zero, the SBW-task would not meet its deadline (because the resulting task set on ECU1 is not schedulable). Hence, the RM updates the set of candidate ECUs and selects ECU2 as the target migration node, since the total CPU utilization available for DL-task on ECU2 if SBW-task is migrated on it is equal to $(1 - (0.49 + 0.32) = 0.26$. Therefore, the utilization desired by SBW-task can be made available by decreasing the bandwidth of the DL–task on ECU2. For instance, the available utilization and the changed bandwidth for DL-task are equal to $U_{(DL-task)} = 0.27$ and $Q^{new}_{DL-task} = 56$.

6.4 Summary

We presented in this chapter the implementation details of ASLA's algorithms followed by a set of experiments to assess the validity of our approach, leaving the real-implementation

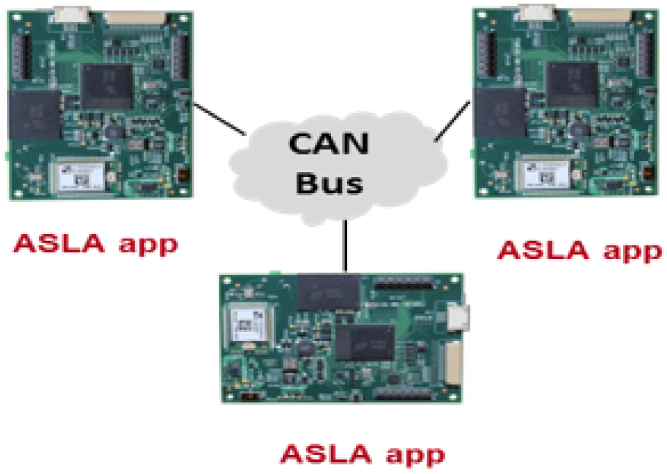

Fig. 6.10 Cars electronics

for our future work. To assess the validity of our approach, we computed the schedulability analysis of TSeRBA and we computed the migration cost for different tasks sets. In addition, we computed the number of preemptions caused by our algorithm. Finally, we computed the overhead caused by the our task migration algorithm.

Chapter 7

Conclusion and Future perspectives

Runtime adaptation of embedded systems was not a main concern in many safety critical application domains up to now. The economic pressure, for time-to market reasons, but also the new challenges posed by the Internet of Things (IoT), imposes a rapid evolution of embedded systems. Smart cities are an example of a specific IoT where cars become connected objects. These new trends with the recent innovation concerning autonomous vehicles and ADAS (Advanced Driver Assistance Systems) raise the problem of evolution of safety critical systems. This was one of the main motivations behind this dissertation: Is it useful to make AUTOSAR adaptive? Our work shows that the AUTOSAR architecture as it is today does not offer enough flexibility to perform runtime adaptation and hence does not comply with the new trends discussed above. We have shown that with the proposition of a new AUTOSAR architecture, i.e., ASLA, that runtime adaptation, is however, possible and this is the focus of the Adaptive AUTOSAR platform, a recent initiative from the AUTOSAR consortium.

In this chapter, we present a summary of the concepts and ideas presented in this thesis. We recall in this context the main contributions of our work for the ASLA approach. Further, we want to discuss how the ASLA approach could be integrated into today's development process. Finally, we want to present some perspectives, how the ASLA approach might be improved for use with future automotive systems.

7.1 Summary of Contributions

The central goal of my thesis is to extend the AUTOSAR framework in order to support runtime adaptation. This challenge involves an important number of facets to be addressed. We recall that runtime adaptation in automotive systems involves task mapping, scheduling of adaptation. Thus, the contributions of my thesis are threefold: (i) resource allocation and scheduling for automotive systems, (ii) proposing TSeRBA algorithm "Tabu search Reconfiguration and Bandwidth Allocation" which we used for task mapping, bandwidth

allocation and reconfiguration for mixed criticality distributed systems, and (iii) runtime adaptation support for AUTOSAR. The contributions are summarized as follows:

(i) Resource allocation and scheduling for automotive systems. We have developed a task-partitioning strategy for allocating SWCs to the ECUs based on the work of [99]. In [99], the authors propose TSMBA "Tabu Search Mapping and Bandwidth Allocation" algorithm, which provides a comprehensive solution that allocates mixed-criticality SWCs (hard and soft) to a distributed architecture and performs processor bandwidth reservation for guaranteeing timing requirements even in the presence of faults. TSMBA [99] cannot be used directly for our goal; specifically it cannot be used directly in systems with task dependencies (i.e., pipeline task model). Therefore, we propose a new allocation algorithm based on TSMBA. So, to incorporate task dependencies, we have defined an abstraction called Operational Chains, which enables timing analysis. An ECU assignment methodology called O-TSMBA (Operational chains- Tabu Search Mapping and Bandwidth allocation) is proposed to consider task dependencies. A key observation is that consolidating tasks that communicate with each other can save computing resources as the consolidation reduces the processing overhead. TSMBA is also adapted to an AUTOSAR-compliant platform to see how the proposed algorithm can be used in the automotive context.

(ii) Proposing an algorithm for task mapping, bandwidth allocation and reconfiguration for mixed-critical hard/soft distributed systems. Conventional real-time theories are in general applicable to automotive systems; however, they do not incorporate highly dynamic attributes and hence do not provide tight schedulability analyses. To the best of our knowledge, TSeRBA is the first algorithm that integrates runtime adaptation, tasks allocation and scheduling within AUTOSAR.

(iii) Runtime adaptation support for AUTOSAR. We propose a layer called ASLA (Adaptive System-Level in AUTOSAR) to incorporate tasks-level adaptation features in AUTOSAR. ASLA aims at extending the AUTOSAR architecture starting from the application layer down to the operating system layer (task model and RTE)(i.e., extending AUTOSAR ECU Software architecture).

7.2 Future Work

Our work offers opportunities both in the short term for investigating extensions and enhancements to the approach, and in the long term for exploring new research directions.

This work opens up many directions for future research. Primarily, we intend to continue validating the results in the experimental platform, but also move it to an industrial setting and try it in a real vehicle. In addition, we envisage an extension to our approach to support mode change protocols for operational mode changes.

During my Master work, my research revolved around decision making in self adaptive systems [28] [27] [25] [26]. I believe that bringing together my Master and my PhD works can be a first step towards safer autonomous vehicles. Hence, ASLA can be easily extended to support machine learning techniques (aka. dynamic decision network). The new architecture is illustrated in Figure 7.1

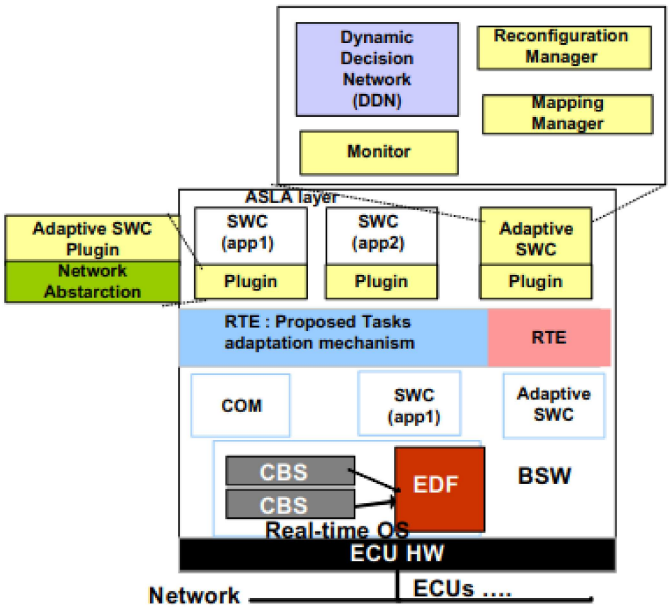

Fig. 7.1 The evolution of ASLA architecture towards autonomous vehicle.

References

[1] Autosar partnership. specification of the rte level,february 2013, http://www.autosar.org/download/R4.2/AUTOSAR_SWS_RTE.pdfs,2013.

[2] Autosar specification of operating system, 2.1.1 edition, feb.2014,http://www.autosar.org/download/R4.2/AUTOSAR_SWS_OS.pdf.

[3] Abeni, L., Buttazzo, G., Superiore, S., and Anna, S. (1998). Integrating multimedia applications in hard real-time systems. In *In Proceedings of the 19th IEEE Real-time Systems Symposium*, pages 4–13.

[4] Abeni, L. and Buttazzo, G. C. (2001). Stochastic analysis of a reseveration based system. In *Proceedings of the 15th International Parallel & Distributed Processing Symposium (IPDPS-01), San Francisco, CA, April 23-27, 2001*, page 92.

[5] Abeni, L., Palopoli, L., Scordino, C., and Lipari, G. (2009). Resource reservations for general purpose applications. *IEEE Transactions on Industrial Informatics*, 5(1):12–21.

[6] Anderson, J. H., Bud, V., and Devi, U. C. (2005). An edf-based scheduling algorithm for multiprocessor soft real-time systems. In *Proceedings of the 17th Euromicro Conference on Real-Time Systems*, ECRTS '05, pages 199–208, Washington, DC, USA. IEEE Computer Society.

[7] Anthony, R., Leonhardi, A., Ekelin, C., Chen, D., Törngren, M., de Boer, G., Drüke, I., Burton, S., Redell, O., Weber, A., and Vollmer, V. (2006). A future dynamically reconfigurable automotive software system. In *Moderne Elektronik im Kraftfahrzeug*, Dresden, Germany.

[8] Audsley, N., Burns, A., Richardson, M., Tindell, K., and Wellings, A. J. (1993). Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8:284–292.

[9] Audsley, N., Burns, A., Richardson, M. F., and Wellings, A. J. (1991). Hard real-time scheduling: The deadline-monotonic approach. In *in Proc. IEEE Workshop on Real-Time Operating Systems and Software*, pages 133–137.

[10] AUTOSAR (03-2017). Autosar adaptive platform, march 2017, https://www.autosar.org/standards/adaptive-platform/.

[11] AUTOSAR (2003). Autosar development cooperation. http://www.autosar.org.

[12] AUTOSAR (2011). Autosar consortium " guide to mode management". http://www.autosar.org/download/R4.2/AUTOSAR_mode_management.pdfs.

[13] AUTOSAR (2014). Autosar methodology. http://www.autosar.org/specifications/release-41/methodology-and-templates/templates/AUTOSAR.

[14] Axelsson, J. and Kobetski, A. (2013). On the conceptual design of a dynamic component model for reconfigurable autosar systems. *SIGBED Rev.*, 10(4):45–48.

[15] Balasubramanian, J., Tambe, S., Lu, C., Gokhale, A., Gill, C., and Schmidt, D. C. (2009). Adaptive failover for real-time middleware with passive replication. *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 0:118–127.

[16] Baruah, S. and Goossens, J. (2003). Scheduling real-time tasks: Algorithms and complexity.

[17] Baruah, S. and Vestal, S. (2008). Schedulability analysis of sporadic tasks with multiple criticality specifications. In *Real-Time Systems, 2008. ECRTS '08. Euromicro Conference on*, pages 147–155.

[18] Baruah, S. K., Bonifaci, V., D'Angelo, G., Li, H., Marchetti-Spaccamela, A., van der Ster, S., and Stougie, L. (2012). The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems. In *24th Euromicro Conference on Real-Time Systems, ECRTS 2012, Pisa, Italy, July 11-13, 2012*, pages 145–154.

[19] Baruah, S. K., Burns, A., and Davis, R. I. (2011). Response-time analysis for mixed criticality systems. In *Proceedings of the 32nd IEEE Real-Time Systems Symposium, RTSS 2011, Vienna, Austria, November 29 - December 2, 2011*, pages 34–43.

[20] Baruah, S. K., Howell, R. R., and Rosier, L. E. (1993). Feasibility problems for recurring tasks on one processor. *Theoretical Computer Science*, 118(1):3 – 20.

[21] Baruah, S. K., Mok, A. K., and Rosier, L. E. (1990). Preemptively scheduling hard-real-time sporadic tasks on one processor. In *In Proceedings of the 11th Real-Time Systems Symposium*, pages 182–190. IEEE Computer Society Press.

[22] Becker, B., Giese, H., Neumann, S., Schenck, M., and Treffer, A. (2009). Model-Based Extension of AUTOSAR for Architectural Online Reconfiguration. In Van Baelen, S., Weigert, T., Ober, I., and Espinoza, H., editors, *Proceedings of the 2nd International Workshop on Model Based Architecting and Construction of Embedded Systems (ACES-MB 2009)*, volume 507 of *CEUR Workshop Proceedings*, pages 123–137. CEUR-WS.org.

[23] BELAGGOUN, A. and Issarny, V. (2016). Towards adaptive autosar : a system-level approach. In *FISITA 2016 World Automotive Congress*, Busan, South Korea.

[24] BELAGGOUN, A., Radermacher, A., and Issarny, V. (2015). ASLA: Adaptive System-Level in AUTOSAR . JRWRTC 2015: 9th Junior Researcher Workshop on Real-Time Computing . Poster - In conjunction with the 23rd International Conference on Real-Time and Network Systems (RTNS 2015).

[25] Bencomo, N. and Belaggoun, A. (2013). Supporting decision-making for self-adaptive systems: From goal models to dynamic decision networks. In *Requirements Engineering: Foundation for Software Quality - 19th International Working Conference, REFSQ 2013, Essen, Germany, April 8-11, 2013. Proceedings*, pages 221–236.

[26] Bencomo, N. and Belaggoun, A. (2014). A world full of surprises: bayesian theory of surprise to quantify degrees of uncertainty. In *36th International Conference on Software Engineering, ICSE '14, Companion Proceedings, Hyderabad, India, May 31 - June 07, 2014*, pages 460–463.

[27] Bencomo, N., Belaggoun, A., and Issarny, V. (2013a). Bayesian artificial intelligence for tackling uncertainty in self-adaptive systems: The case of dynamic decision networks. In *2nd International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering, RAISE 2013, San Francisco, CA, USA, May 25-26, 2013*, pages 7–13.

[28] Bencomo, N., Belaggoun, A., and Issarny, V. (2013b). Dynamic decision networks for decision-making in self-adaptive systems: A case study. In *Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '13, pages 113–122, Piscataway, NJ, USA. IEEE Press.

[29] Berger, C. and Tichy, M. (2012). Towards transactional self-adaptation for autosar on the example of a collision detection system.

[30] Biondi, A., Buttazzo, G. C., and Bertogna, M. (2015). Supporting component-based development in partitioned multiprocessor real-time systems. In *Real-Time Systems (ECRTS), 2015 27th Euromicro Conference on,IEEE*, pages 269–280.

[31] Broy, M. (2006). Challenges in automotive software engineering. In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pages 33–42, New York, NY, USA. ACM.

[32] Broy, M., Kruger, I. H., Pretschner, A., and Salzmann, C. (2007). Engineering automotive software. *Proceedings of the IEEE*, 95(2):356–373.

[33] Burns, A. (2013). The application of the original priority ceiling protocol to mixed criticality systems. In George, L. and Lipari, G., editors, *ReTiMiCS, RTCSA*, pages 7–11.

[34] Burns, A. (2014). System mode changes - general and criticality-based. In Cucu-Grosjean, L. and Davis, R., editors, *Proc. 2nd Workshop on Mixed Criticality Systems (WMC), RTSS*, pages 3–8.

[35] Burns, A. and Davis, R. I. (2014). Adaptive mixed criticality scheduling with deferred preemption. In *Proceedings of the IEEE 35th IEEE Real-Time Systems Symposium, RTSS 2014, Rome, Italy, December 2-5, 2014*, pages 21–30.

[36] Burns, A. and Davis, R. I. (2016). Mixed criticality systems: A review. Technical report, Department of Computer Science, University of York,.

[37] Buttazzo, G. (2006). Research trends in real-time computing for embedded systems. *SIGBED Rev.*, 3(3).

[38] Buttazzo, G. and Santinelli, L. (2015). Adaptive mechanisms for component-based real-time systems". In *Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems (ASH 2015), June 15-18, 2015, Montreal, QC, Canada.*

[39] Buttazzo, G. C. (2004). *Hard Real-time Computing Systems: Predictable Scheduling Algorithms And Applications (Real-Time Systems Series).* Springer-Verlag TELOS, Santa Clara, CA, USA.

[40] Buttazzo, G. C., Lipari, G., and Abeni, L. (1998). Elastic task model for adaptive rate control. In *Proceedings of the IEEE Real-Time Systems Symposium.*

[41] Chaaban, K., Leserf, P., and Saudrais, S. (2009). Steer-by-wire system development using autosar methodology. In *2009 IEEE Conference on Emerging Technologies Factory Automation*, pages 1–8.

[42] Cheng, B. H. and et al., L. (2009). Software engineering for self-adaptive systems. chapter Software Engineering for Self-Adaptive Systems: A Research Roadmap, pages 1–26. Springer-Verlag, Berlin, Heidelberg.

[43] Cheng, B. H., Lemos, R., Giese, H., Inverardi, P., Magee, J., Andersson, J., Becker, B., Bencomo, N., Brun, Y., Cukic, B., Marzo Serugendo, G., Dustdar, S., Finkelstein, A., Gacek, C., Geihs, K., Grassi, V., Karsai, G., Kienle, H. M., Kramer, J., Litoiu, M., Malek, S., Mirandola, R., Müller, H. A., Park, S., Shaw, M., Tichy, M., Tivoli, M., Weyns, D., and Whittle, J. (2009). Software engineering for self-adaptive systems. chapter Software Engineering for Self-Adaptive Systems: A Research Roadmap, pages 1–26. Springer-Verlag, Berlin, Heidelberg.

[44] Davis, R. and Bertogna, M. (2012). Optimal fixed priority scheduling with deferred pre-emption. In *Real-Time Systems Symposium (RTSS), 2012 IEEE 33rd*, pages 39–50.

[45] Davis, R. I., Burns, A., Bril, R. J., and Lukkien, J. J. (2007). Controller area network (can) schedulability analysis: Refuted, revisited and revised. *Real-Time Syst.*, 35(3):239–272.

[46] de Normalización, O. I. (2011). *ISO 26262: Road Vehicles : Functional Safety*. ISO.

[47] Di Natale, M., Perillo, D., Chirico, F., Sindico, A., and Sangiovanni-Vincentelli, A. (2016). A model-based approach for the synthesis of software to firmware adapters for use with automatically generated components. *Software & Systems Modeling*.

[48] Emberson, P. and Bate, I. (2008). Extending a task allocation algorithm for graceful degradation of real-time distributed embedded systems. In *Proceedings 29th Real-Time Systems Symposium (RTSS 08)*, pages 270–279.

[49] ERIKA (2010). Erika entreprise. http://erika.tuxfamily.org/drupal/.

[50] Francis Cottet, E. G. (2005). *Systèmes temps réel de contrôle-commande : Conception et implémentation*. Dunod.

[51] Fritsch, S. and Clarke, S. (2008). Timeadapt: Timely execution of dynamic software reconfigurations. In *Proceedings of the 5th Middleware Doctoral Symposium*, MDS '08, pages 13–18, New York, NY, USA. ACM.

[52] Fritsch, S., Senart, A., and Schmidt, D. C. (2008). Time-bounded adaptation for automotive system software. ICSE '08, pages 571–580, New York, NY, USA. ACM.

[53] García-Valls, M. and Basanta-Val, P. (2012). A practical solution for functional reconfiguration of real-time service based applications through partial schedulability. In *REACTION*.

[54] García-Valls, M. and Basanta-Val, P. (2013). A real-time perspective of service composition: Key concepts and some contributions. *Journal of Systems Architecture - Embedded Systems Design*, 59(10-D):1414–1423.

[55] García-Valls, M., Basanta-Val, P., Marcos, M., and Estevez, E. (2013). A bi-dimensional qos model for SOA and real-time middleware. *Comput. Syst. Sci. Eng.*, 28(5).

[56] García-Valls, M., Lopez, I. R., and Fernandez-Villar, L. (2013). iland: An enhanced middleware for real-time reconfiguration of service oriented distributed real-time systems. *IEEE Trans. Industrial Informatics*, 9(1):228–236.

[57] Garcia-Valls, Marisol, R. J. C. (2014). scheduling component replacement for timely execution in dynamic systems. *Software: Practice & Experience, January 2013*, 44(1):889–910.

[58] Giese, H., Burmester, S., Schäfer, W., and Oberschelp, O. (2004). Modular design and verification of component-based mechatronic systems with online-reconfiguration. In *Proceedings of the 12th ACM SIGSOFT Twelfth International Symposium on Foundations of Software Engineering*, SIGSOFT '04/FSE-12, pages 179–188, New York, NY, USA. ACM.

[59] Guangming, Q. (2009). An earlier time for inserting and/or accelerating tasks. *Real-Time Syst.*, 41(3):181–194.

[60] He, L., Chen, G. Y., and Zheng, H. Y. (2015). Fault tolerant control method of dual steering actuator motors for steer-by-wire system. *International Journal of Automotive Technology*, 16(6):977–987.

[61] Holzl, M., Rauschmayer, A., and Wirsing, M. (2008). Software-intensive systems and new computing paradigms. chapter Engineering of Software-Intensive Systems: State of the Art and Research Challenges, pages 1–44.

[62] Horn, W., S. (1974). ome simple scheduling algorithms. *Naval Research Logistics Quarterly.*, 21(1).

[63] Jain, R., Hughes, C. J., and Adve, S. V. (2002). Soft real-time scheduling on simultaneous multithreaded processors. In *Real-Time Systems Symposium, 2002. RTSS 2002. 23rd IEEE*, pages 134–145.

[64] Juarez-Dominguez, A. L., Day, N. A., and Joyce, J. J. (2008). Modelling feature interactions in the automotive domain. In *Proceedings of the 2008 International Workshop on Models in Software Engineering*, MiSE '08, pages 45–50, New York, NY, USA. ACM.

[65] Kavi, K. M., Giorgi, R., and Arul, J. (2001). Scheduled dataflow: Execution paradigm, architecture, and performance evaluation. *IEEE Trans. Comput.*, 50(8):834–846.

[66] Kim, J. ((2014). Dissertations. Paper 386.). *"Dependable Cyber-Physical Systems"*. PhD thesis, CMU University.

[67] Kloukinas, C. and Yovine, S. (2011). A model-based approach for multiple qos in scheduling: from models to implementation. *Autom. Softw. Eng.*, 18(1):5–38.

[68] Kopetz, H. (1997a). *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, Norwell, MA, USA, 1st edition.

[69] Kopetz, H. (1997b). *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, Norwell, MA, USA, 1st edition.

[70] Kopetz, H. and Bauer, G. (2003). The time-triggered architecture. In *PROCEEDINGS OF THE IEEE*, pages 112–126.

[71] Kopetz, H., Tindell, K., Wolf, F., and Ernst, R. (2003). Safe automotive software development. *Messe Munich, Germany*.

[72] Kramer, J. and Magee, J. (2007). Self-managed systems: An architectural challenge. In *2007 Future of Software Engineering*, FOSE '07, pages 259–268, Washington, DC, USA.

[73] L.Abeni and Buttazzo, G. (2001). Stochastic analysis of a reservation based system. In *in Proceedings of 15th International Parallel and Distributed Processing Symposium, pp.946-952*.

[74] Lakshmanan, K., Bhatia, G., and Rajkumar, R. (2010). Integrated end-to-end timing analysis of networked autosar-compliant systems. In *Design, Automation and Test in Europe, DATE 2010, Dresden, Germany, March 8-12, 2010*, pages 331–334.

[75] Lardieri, Patrick, S. e. a. (2007). A multi-layered resource management framework for dynamic resource management in enterprise dre systems. *J. Syst. Softw.*

[76] Lehoczky, J., Sha, L., and Ding, Y. (1989). The rate monotonic scheduling algorithm: exact characterization and average case behavior. In *Real Time Systems Symposium, 1989., Proceedings.*, pages 166–171.

[77] Lehoczky, J. P. (1990). Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *RTSS*, pages 201–209. IEEE Computer Society.

[78] Liu, C. L. and Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61.

[79] M. Jan, L. Z. and Pitel, M. (2013). Maximizing the execution rate of low-criticality tasks in mixed criticality system. In *In Proceedings of Workshop on Mixed Criticality, IEEE Real-Time Systems Symposium (RTSS), pages 43?48,.*

[80] Makowitz, R. and Temple, C. (2006). Flexray - a communication network for automotive control systems. In *2006 IEEE International Workshop on Factory Communication Systems*, pages 207–212.

[81] Martorell, H., Fabre, J., Roy, M., and Valentin, R. (2014). Improving adaptiveness of AUTOSAR embedded applications. In *Symposium on Applied Computing, SAC 2014, Gyeongju, Republic of Korea - March 24 - 28, 2014*, pages 384–390.

[82] Metzger, A. (2004). Feature interactions in embedded control systems. *Comput. Netw.*, 45(5):625–644.

[83] Mitzlaff, M. and Kapitza (2010). Enabling mode changes in a distributed automotive system. In *Proceedings of the CARS*, CARS '10, pages 75–78, New York, NY, USA. ACM.

[84] Narasimhan, P., Dumitraş, T. A., Paulos, A. M., Pertet, S. M., Reverte, C. F., Slember, J. G., and Srivastava, D. (2005). Mead: Support for real-time fault-tolerant corba: Research articles. *Concurr. Comput. : Pract. Exper.*, 17(12):1527–1545.

[85] Navet, N. and Simonot-Lion, F. (2008a). *Automotive Embedded Systems Handbook*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition.

[86] Navet, N. and Simonot-Lion, F. (2008b). *Automotive Embedded Systems Handbook*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition.

[87] Nelis, V. and Andersson, B. e. a. (2010). Global-edf scheduling of multimode real-time systems considering mode independent tasks. In *Proceedings of the ECRTS*, ECRTS11, pages 205–214, Washington, DC, USA. IEEE.

[88] OSEK/VDX (2005). Osek group. osek/vdx operating system (release 2.2.3). http://portal.osek-vdx.org/.

[89] Pant, A. (2011). A comparison between fcfs and mixed scheduling.

[90] Pedro, P. (1999). . *Schedulability of mode changes in flexible real-time distributed systems. Ph.D. thesis,*. PhD thesis, University of York, Department of Computer Science.

[91] Pedro, P. and Burns, A. (1998). Schedulability analysis for mode changes in flexible real-time systems. In *ECRTS*, pages 172–179.

[92] P.Emberson, R. and R.Davis (2010). Techniques for the synthesis of multiprocessor tasksets. In *Proc. of the 1st Int l Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2010) ,ECRTS10*.

[93] Prasad, K. V., Broy, M., and K, I. (2010). Scanning advances in aerospace & automobile software technology. *Proceedings of the IEEE*, 98(4):510–514.

[94] Pretschner, A., Broy, M., Kruger, I. H., and Stauner, T. (2007). Software engineering for automotive systems: A roadmap. In *2007 Future of Software Engineering*, FOSE '07, pages 55–71, Washington, DC, USA. IEEE Computer Society.

[95] Rasche, A. and Polze, A. (2005). Dynamic reconfiguration of component-based real-time software. In *Proceedings of the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*, WORDS '05, pages 347–354, Washington, DC, USA. IEEE Computer Society.

[96] Rasche, A. and Polze, A. (2008). Redac dynamic reconfiguration of distributed component-based applications with cyclic dependencies. In *Proceedings of the 2008 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing*, ISORC '08, pages 322–330, Washington, DC, USA. IEEE Computer Society.

[97] Real, J. and Crespo, A. (2004). Mode change protocols for real-time systems: A survey and a new proposal. *Real-Time Syst.*, 26(2):161–197.

[98] Sangiovanni-Vincentelli, A. and Di Natale, M. (2007). Embedded system design for automotive applications. *Computer*, 40(10):42–51.

[99] Saraswat, P. K., Pop, P., and Madsen, J. (2010). Task mapping and bandwidth reservation for mixed hard/soft fault-tolerant embedded systems. In *Proceedings of the 2010 16th IEEE ,RTAS*.

[100] Schwabl, W., Reisinger, J., and Grunsteidl, G. (1989). A survey of mars. Technical report, Research Report 16/89, Institut fur Technische Informatik, Technische Universitat.

[101] Sha, L., Abdelzaher, K.-E., Cervin, A., Baker, T., Burns, A., Buttazzo, G., Caccamo, M., Lehoczky, J., and Mok, A. K. (2004). Real time scheduling theory: A historical perspective. *Real-Time Syst.*, 28(2-3):101–155.

[102] Sha, L., Rajkumar, R., Lehoczky, J. P., and Ramamritham, K. (1989). Mode change protocols for priority-driven preemptive scheduling. *Real-Time Systems*, 1(3):243–264.

[103] Silberschatz, A., Galvin, P. B., and Gagne, G. (2008). *Operating System Concepts*. Wiley Publishing, 8th edition.

[104] software empiricist, K. W. K. T. (1973). *SIGMETRICS Perform. Eval. Rev.*, 2(2).

[105] Stoimenov, N. and al. (2009). Reliable mode changes in real-time systems with fixed priority or edf scheduling. DATE '09, pages 99–104.

[106] Stringfellow, M. V., Leveson, N. G., and Owens, B. (2010). Safety-driven design for software-intensive aerospace and automotive systems. *Proceedings of the IEEE*, 98(4):515–525.

[107] Su, H. and Zhu, D. (2013). An elastic mixed-criticality task model and its scheduling algorithm. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, pages 147–152.

[108] Tanenbaum, A. S. and Steen, M. v. (2006). *Distributed Systems: Principles and Paradigms (2Nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[109] Tindell, K. and Alonso, . (1996). A very simple protocol for mode changes in priority preemptive systems. Technical report, Universidad Polit?cnica de Madrid.Technical report,.

[110] Tindell, K., Burns, A., and Wellings, A. J. (1992). Mode changes in priority pre-emptively scheduled systems. In *Proceedings of the RTSS , USA, December*.

[111] Vandewoude, Y. and Berbers, Y. (2002). Run-time evolution for embedded component-oriented systems. In *International Conference on Software Maintenance, 2002. Proceedings.*, pages 242–245.

[112] VECTOR (2016). Vector webinar slides: Introduction to autosar. https://vector.com/.

[113] Vestal, S. (2007). Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Proceedings of the 28th IEEE International Real-Time Systems Symposium*, RTSS '07, pages 239–243, Washington, DC, USA. IEEE Computer Society.

[114] Vincent Nelis, Joel Goossens, B. A. (2009). "two protocols for scheduling multi-mode real-time systems upon identical multiprocessor platforms",. In *ECRTS, 2009, 2012 24th Euromicro Conference on Real-Time Systems, 2012 24th Euromicro Conference on Real-Time Systems 2009, pp. 151-160, doi:10.1109/ECRTS.2009.27*.

[115] Wagner, S., Schatz, B., Puchner, S., and Kock, P. (2010). A case study on safety cases in the automotive domain: Modules, patterns, and models. In *Proceedings of the 2010 IEEE 21st International Symposium on Software Reliability Engineering*, ISSRE '10, pages 269–278.

[116] Wahler, Michael, E. A. (2011). Non-disruptive large-scale component updates for real-time controllers. In Abiteboul, S., B?hm, K., Koch, C., and Tan, K.-L., editors, *ICDE Workshops*, pages 174–178. IEEE.

[117] Wolf, F., Balasubramanian, J., Tambe, S., Gokhale, A., and Schmidt, D. C. (2010). Supporting component-based failover units in middleware for distributed real-time and embedded systems. *Journal of Software Architectures: Embedded Software Design, Special Issue on Embedded and Real-time*.

[118] Xu, J. and Parnas, D. L. (1993). On satisfying timing constraints in hard-real-time systems. *IEEE Trans. Softw. Eng.*, 19(1):70–84.

[119] Zeeb, A. (2012). Plug-and-play-lösung für autosar-software-komponenten. *ATZelektronik*, 7(1):28–33.

[120] Zeller, Marc; Prehofer, C. (2012). Timing constraints for runtime adaptation in real-time, networked embedded systems. In *(SEAMS)(ICSE)*.

[121] Zeller, M., Prehofer, C., Weiss, G., Eilers, D., and Knorr, R. (2011). Towards self-adaptation in real-time, networked systems: Efficient solving of system constraints for automotive embedded systems. In *5th IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2011, Ann Arbor, MI, USA, October 3-7, 2011*, pages 79–88.

Appendix A

TSeRBA Algorithm Implementation

A.0.1 Cost Function Implementation

```
/**
 * @author Amel BELAGGOUN
 * @date 11-12-2014
 */

import org.apache.commons.math.linear.*;

@SuppressWarnings("deprecation")
public class CostFuncCalc {
public static double getCostFuncValue(double[][] array) {
        RealMatrix M = new RealMatrixImpl(array);
        double Total_Cost_Value = 0;
for (int i = 0; i < M.getRowDimension(); i++) {

int DDLINE = (int) M.getEntry(i, 1);
double WGHT = (double) M.getEntry(i, 2);
int Q = (int) M.getEntry(i, 3);
//System.out.println("\nn+i+" "+DDLINE+" "+WGHT+" "+Q+"\n");
 double[] V = M.getRow(i);
double[] U =  new double [XMLRead.getPDFLen()];
for (int k = 5, j = 0; k < V.length; k++ ,j++) {
        U[j]= V[k];
        }
        RealMatrix MRKV;
    RealMatrix PI = null;
        MRKV = genMarkovMatrix(U, Q);
         PI = MRKV;
        for (int lm = 0; lm <= 100; lm++) {
        PI = PI.multiply(PI);
if (calc_err(PI) < 1E-30)
```

```
        break;}
        double sum = 0;
for (int jj = 0; jj < PI.getRowDimension(); jj++)          {
        sum = sum + PI.getEntry(jj, 0);
}
PI = PI.scalarMultiply(1 / sum);double [] CDF = new
    double[PI.getColumnDimension()];
        CDF [0]= PI.getEntry(0, 0);
for (int kk = 1; kk < PI.getColumnDimension(); kk++) {
        CDF[kk] = CDF[kk - 1] + PI.getEntry(kk, 0);}              double
            CF_value;
        CF_value = CDF[DDLINE]; Total_Cost_Value = Total_Cost_Value +
            WGHT*CF_value;
        }
  System.out.println(" \n The TOtal COst VaLue is  " + Total_Cost_Value +
      "\n\n");
 return (Total_Cost_Value/XMLRead.getNumSRT())*100;
        }
private static double calc_err(RealMatrix PI) {
// TODO Auto-generated method stub
 double err = 0;
         for (int i = 0; i < PI.getColumnDimension(); i++) {
err = err + (PI.getEntry(i, 0) - (PI.getEntry(i, PI.getRowDimension()-1)));
                }
                return err;
        }
private static RealMatrix genMarkovMatrix(double[] U, int Q) {
                // TODO Auto-generated method stub

int NUMJOBS = 500;
double WCL = 0.0;
for (int k = 0; k < (int) U.length; k++) {
          WCL = WCL + NUMJOBS* U[k]* (k-Q) ;

}
int WCLength = (int)  200;
if (WCLength < Q) {

        WCLength = Q + 1;
        }

if (WCLength < U.length) {
WCLength = U.length;
                }

double[] UA = new double[WCLength];
```

```
for (int i = 0; i < WCLength; i++) {
        if (i < (int) U.length) {
                UA[i] = U[i];
        }
else

UA[i]= 0;
                    }

 double[][] M = new double [WCLength][WCLength];
    for (int i = 0; i < WCLength; i++) {
        for (int k = 0; k < Q+1; k++) {
         M[i][k] = UA[i];
        }
                }
   for (int i = 0; i < WCLength; i++) {
        for (int k = Q+1; k < WCLength; k++) {
                        M[i][k] = 0;
                        }
                  }
  for (int i = 0; i < WCLength; i++) {
        for (int k = Q + 1, l = 1; k < WCLength; k++, l++) {
                if (i + l < WCLength) {
        M[i + l][k] = UA[i];
                }
        }
        }
    RealMatrix MRKV = new RealMatrixImpl(M);
         return MRKV;
        }
}
```

A.0.2 TABU Search Implementation

```
/**
 * @author Amel BELAGGOUN
   @date 15-01-2015
 */
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.text.DateFormat;
```

```
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.HashSet;
import java.util.Random;
import java.util.Set;
import java.util.StringTokenizer;


public class TabuSearch_MB {



        private static int NC ;
        private static int TL;
        private static double GLOBAL_BEST_COST;
        private static double[][] GLOBAL_BEST_SOL;
        private static double[][] CURR_SOL;
        private static double[][][] TABU;
        private static double[][] SYS;
        private static int[][] COMM_ARR;
        private static int TABU_LENGTH;
        private static int LAST_IMPROVED;
        private static int ITR_COUNT;
        private static double[][] CHOSEN_MOVE;
        private static double NEXT_SOL_COST;
        private static int MAX_ITERATIONS;
        private static int LAST_IMPROVED_COUNT;




        public static void main(String[] args) throws IOException {
                // TODO Auto-generated method stub


XMLRead.getSystem()
String Filename = "Results_MB_".concat(getDateTime().concat(".txt"));
FileWriter outFile = new FileWriter(Filename);
PrintWriter out = new PrintWriter(outFile);

        double[][] INIT_SOL = readInitialSOL("init_sol.txt");
 COMM_ARR = readComm("comm_info.txt");
NC = XMLRead.getNC();
 TL = XMLRead.getTabuLen();
 TABU_LENGTH = 0;
```

```
  LAST_IMPROVED = 0;
ITR_COUNT = 0 ;
 MAX_ITERATIONS = XMLRead.getNumIter();
 LAST_IMPROVED_COUNT = XMLRead.getLastImprvCnt();
 double[][][][] CM = new
     double[NC][XMLRead.getSysArrRows()][XMLRead.getNumECUs()];
          TABU = new
              double[TL][XMLRead.getSysArrRows()][XMLRead.getNumECUs()];

            SYS = XMLRead.getSystem();

                              System.out.println("\n--------O-TSeMBA AND
                                  TSeRBA----------\n");
            System.out.println("\n\n STATISTICS : " +NC +" "+TL+ " "+
                MAX_ITERATIONS + " "+LAST_IMPROVED_COUNT);
            out.println("\n # LIST OF PARAMETERS " +
                    " \n # NUM_ELEM_CAND_SET : \t "+NC+
                      " \n # LAST_IMPR_CNT: \t  "+ LAST_IMPROVED_COUNT +
                    " \n # NUM_ITERATIONs: \t  "+ MAX_ITERATIONS+ "\n\n" );
                  out.flush();
System.out.println("\n IS IT A VALID MAPPING SOLUTION \n" +
    isValidSol(INIT_SOL));

          if (isValidSol(INIT_SOL) == 1) {
           CURR_SOL = INIT_SOL;
            GLOBAL_BEST_SOL = INIT_SOL; GLOBAL_BEST_COST =
                CostFuncCalc.getCostFuncValue(createParameterArray(INIT_SOL));
                System.out.println("\n INTITIAL COST FUNCTION VALUER \n"+
                GLOBAL_BEST_COST);
LAST_IMPROVED= 0;
         ITR_COUNT = 1;
           CM = CreateCandidateMoves(INIT_SOL);
              double[][][][] NCM = compactCandSet(CM);
              CHOSEN_MOVE = chooseMove(NCM);
         NEXT_SOL_COST =
             CostFuncCalc.getCostFuncValue(createParameterArray(CHOSEN_MOVE));
 System.out.println("\n-----------------THE CHOSEN---------");
                printMoreDetails(CHOSEN_MOVE);
  System.out.println(" COST OF CHOSEN ONE------\n"+ NEXT_SOL_COST );
            while (ITR_COUNT< MAX_ITERATIONS) {
           while (LAST_IMPROVED < LAST_IMPROVED_COUNT) {
System.out.println("\n\n ########################################ITERATION
    NO################\n\n"+ ITR_COUNT);
System.out.println("\n COST @ ITERATION "+ ITR_COUNT+" est EGAL " +
    GLOBAL_BEST_COST);
System.out.println("\n THe CURRENT SOLUTION IS : \n ");
```

```
        printMoreDetails(GLOBAL_BEST_SOL);
        System.out.println("\n COMMUNICATION COST :" +
            getCommunicationCost(COMM_ARR,GLOBAL_BEST_SOL));
out.println("\n " +ITR_COUNT+","+GLOBAL_BEST_COST+"    ");
out.flush();
System.out.println("the current solution \n");
printSOL(CURR_SOL);
addToTabu(CURR_SOL);

LAST_IMPROVED++;
ITR_COUNT++;
CURR_SOL = CHOSEN_MOVE;
if (NEXT_SOL_COST > GLOBAL_BEST_COST) {
GLOBAL_BEST_COST = NEXT_SOL_COST;
GLOBAL_BEST_SOL = CURR_SOL;
LAST_IMPROVED = 0;
System.out.println("\n ---------The cHosen SOLUTION-------------");
printMoreDetails(CURR_SOL);
System.out.println("COST OF THE CHOSEN ONE- \n "+NEXT_SOL_COST);
out.println(", Q\n ");

}else {System.out.println("\n THERE IS NO GOOD SOLUTION THIS TImE \n");}

CM = CreateCandidateMoves(CURR_SOL);
double[][][][] NCM1 = compactCandSet(CM);
        CHOSEN_MOVE = chooseMove(NCM1);
    NEXT_SOL_COST =
        CostFuncCalc.getCostFuncValue(createParameterArray(CHOSEN_MOVE));
        out.println(" \n");
        }
 CHOSEN_MOVE = moveSomeWhereElse(CURR_SOL);
         if (isValidSol(CHOSEN_MOVE) == 1) {
NEXT_SOL_COST =
    CostFuncCalc.getCostFuncValue(createParameterArray(CHOSEN_MOVE));
LAST_IMPROVED = 0;
out.println(", M  \n");
out.println(" \n");
ITR_COUNT++;
                }
        }//end while externe
            }
System.out.println("\n\n-----------------THE END OF THE DESIGN SPACE
   EXPLORATION ------------\n\n");
System.out.println("\n COST"+ GLOBAL_BEST_COST);
                printSOL(GLOBAL_BEST_SOL);
        printMoreDetails(GLOBAL_BEST_SOL);
```

```
                out.close();
        }
        private static double[][] readInitialSOL(String Filename) {
                // TODO Auto-generated method stub
                double [][] INIT_SOL = new double
                    [XMLRead.getSysArrRows()][XMLRead.getNumECUs()];
                try {
        System.out.println("READ the INTIAL SYSTEM DESCRIPTION FROM the
            File \n");
        FileReader input = new FileReader(Filename);
BufferedReader buffread = new BufferedReader(input);
String Line;
int count = 0;
Line = buffread.readLine();
count ++;
int col = 0;
while (Line != null) {
        col = 0;
System.out.println(count+": "+Line);
  { StringTokenizer st = new StringTokenizer(Line);
     while (st.hasMoreTokens()) {
        String tok =  st.nextToken();
Float f2 = new Float(tok);
double d2 = f2.doubleValue();
//       System.out.println(" *"+col+" * "+d2);
    INIT_SOL[count-1][col]= d2;
col++;
   }

    }

Line = buffread.readLine();
                count++;
            }


 buffread.close();

} catch (ArrayIndexOutOfBoundsException e) {
                        // TODO: handle exception
                        System.out.println(" CANNOT FIND THE FILE ");
                }catch (IOException e) {
                        // TODO: handle exception
                 e.printStackTrace();
                }
```

```
            return INIT_SOL;
      }
public static double[][] moveSomeWhereElse(double[][] CURR_SOL)
                  {
                                                      return GLOBAL_BEST_SOL;
                                              }


public static int isTabu(double[][] SOL) {
          double err = 0;
          for (int k = 0; k < TABU_LENGTH; k++) {
             err = 0;
                for (int i = 1; i < XMLRead.getSysArrRows(); i++) {
           for (int j = 0; j < XMLRead.getNumECUs(); j++) {
             err = err + Math.abs(TABU[k][i][j] - SOL[i][j]);
              System.out.print("  what is the value of err"+err);
                 }

         }
               if (err == 0)
               return 1;

           }
               return 0;
        }



      public static int addToTabu(double[][] SOL) {

         if (TABU_LENGTH < TL)
          {
              TABU[TABU_LENGTH] = SOL;
              TABU_LENGTH++;
          }
          else if(TABU_LENGTH==TL)
          {
              for (int i= 0; i < TABU_LENGTH-2; i++)
          {
                      TABU[i]=TABU[i+1];
          }
          TABU[TABU_LENGTH-1]= SOL;
          }

              return 0;
```

```
        }


    public static int printTabu() {
         for (int k = 0; k < TABU_LENGTH; k++) {
              System.out.println("\n ITEM NUMERO\n" +k);
             for (int i = 1; i < XMLRead.getSysArrRows(); i++) {
                             System.out.println("\n");
                             for (int j = 0; j < XMLRead.getNumECUs();
                                 j++) {
                        System.out.println("THIS Is the TaBU LIst"+
                            TABU[k][i][j]);
                             }
                        }
                        }
             return 0;
             }


        /**
         * @param SOL
          * @return
         */
  private static double[][][] CreateCandidateMoves(double[][] SOL) {
             // TODO Auto-generated method stub
System.out.println("\n ############## Creating Candidate Moves - both
    Mapping and alloc ################ \n");

 double[][] MOV;
 int M_param = 0;
     int Q_param = 0;
 int skip_step = 1;
     double[][][] CM = new
         double[NC][XMLRead.getSysArrRows()][XMLRead.getNumECUs()];
 int CM_index = 0;
     for (int k = 0; k < NC; k++) {

     System.out.println("\n -- what contains-CM-" + CM_index +
         "------------");

MOV = geMove(SOL, Q_param, M_param, skip_step );

     if (isValidSol(MOV) == 1) {
     CM[CM_index] = MOV;
     CM_index++;
     System.out.println("\n  the Solution is");
```

```
        printMoreDetails(MOV);
                printSOL(MOV);
                        }
                }
                return CM;
        }




       private static double[][][][] compactCandSet(double[][][][] CM) {
         // TODO Auto-generated method stub

           double [][][][] NCM = deleteNullElements(CM);
          System.out.println("\n AFTER DEL NULL\n");
           printCandSet(NCM);


        double [][][][] NCM1 = deleteDuplicateElements(NCM);
        System.out.println("\nAFTER DEL DUP\n");
        printCandSet(NCM1);

                return NCM1;
                }


         private static double[][][][] deleteDuplicateElements(double[][][][]
            CM) {
            // TODO Auto-generated method stub
           Set<double[][]> NCM_set = new HashSet<double [][]>();

          for (int k = 0; k < CM.length; k++) {
            if (isAlreadyPresent(CM[k], NCM_set)==0) {
//                      System.out.println("\n\n  Not Present\n "+k);
                NCM_set.add(CM[k]);
                        }
                }
  double [][][][] NCM = new double
     [NCM_set.size()][XMLRead.getSysArrRows()][XMLRead.getNumECUs()];
          NCM_set.toArray(NCM) ;
                return NCM;
                }
```

```
private static int isAlreadyPresent(double[][] SOL, Set<double[][]>
    NCM_set) {
                // TODO Auto-generated method stub
                double err = 0;
double [][][][] NCM = new double[NCM_set.size()]
    [XMLRead.getSysArrRows()][XMLRead.getNumECUs()];
                NCM_set.toArray(NCM);

 if (NCM_set.size() == 0) { return 0;};
  for (int k = 0; k < NCM_set.size(); k++) {
                                err = 0;
                for (int i = 1; i < XMLRead.getSysArrRows(); i++) {
                for (int j = 0; j < XMLRead.getNumECUs(); j++) {
                                err = err + Math.abs(NCM[k][i][j]-
                                    SOL[i][j]);
                            }
                    }
                                if (err == 0)
                                        return 1;

                        } // end for k

                return 0;
            }




  private static double[][][] deleteNullElements(double[][][] CM) {
      // TODO Auto-generated method stub
    int NUM = 0;

    for (int k = 0; k < NC; k++) {

    double SUM = 0 ;
   for (int i = 1; i < XMLRead.getSysArrRows(); i++) {
      for (int j = 0; j < XMLRead.getNumECUs(); j++) {
           SUM = SUM + CM[k][i][j];
                }
           }
         // System.out.println(" SUM "+SUM);
          if (SUM != 0)
               NUM++;

          }
        System.out.println(" No of non void entries"+NUM);
```

```
double [][][][] NCM = new
    double[NUM][XMLRead.getSysArrRows()][XMLRead.getNumECUs()];
                            System.arraycopy(CM, 0, NCM, 0, NUM);

                            return NCM;
                    }

 private static void printCandSet(double[][][][] KCM) {
          // TODO Auto-generated method stub
           for (int k = 0; k < KCM.length; k++) {
                  System.out.println(" \nCandidate Set # : "+k+"\n");
                  printMoreDetails(KCM[k]);

                            }

 }
  public static double [][] geMove(double [][][]ARR,int M_param, int
     Q_param,int skip_step) {

Random generator = new Random(System.currentTimeMillis());
                int rand = generator.nextInt(2);

    double [][] MOV_ARR = new double
        [XMLRead.getSysArrRows()][XMLRead.getNumECUs()];
              if (rand == 0) {
System.out.println("\n BW move");
         MOV_ARR = geBandwidthMove (ARR,M_param,Q_param,skip_step);
             System.out.println("\n IS a VALID mapping
             splution"+isValidSol(MOV_ARR));
          printSOL(MOV_ARR);
System.out.println("AAAAAAAAAAA");
System.out.println("\n MaPping MOOovvVEeee");
MOV_ARR = geMappingMove(ARR, M_param,Q_param,skip_step);
System.out.println("\n IS a VALID mapping solution" +isValidSol(MOV_ARR));
          printSOL(MOV_ARR);
System.out.println("AWWWWWWWWWWWW"); }

                  return MOV_ARR;
                   }



private static double[][] geBandwidthMove(double[][] ARR, int M_param, int
    Q_param, int skip_step) {
                  // TODO Auto-generated method
                  Random generator = new Random(System.currentTimeMillis());
```

```
        double [][] MOV_ARR = new double
            [XMLRead.getSysArrRows()][XMLRead.getNumECUs()];
                do {
        int randchange = generator.nextInt(XMLRead.getNumSRT()) + 1;

for (int j = 0; j < XMLRead.getNumECUs(); j++) {
        for (int i = 1; i < XMLRead.getSysArrRows(); i++) {
         MOV_ARR[i][j] = ARR[i][j];
                  if (SYS[i][1]== 1) {
                        if (ARR[i][j] != 0) {


if (i == randchange) {
int Rand1 = generator.nextInt(2);
int Rand2 = generator.nextInt((int) ((SYS[i][4]-ARR[i][j])/skip_step));
Rand2 = Rand2*skip_step;
System.out.println(" QM Randchange "+randchange+" RAND1 "+Rand1+ " RAND2
    "+Rand2+"\n");
 if (Rand1 == 0) {
double Q_Temp = ARR[i][j];
        double Q =  ARR[i][j];
        if (Rand2 < Q) {
         Q = Q- Rand2;
        System.out.println("\n -- Exp Q  : " +XMLRead.getExpection(i, j,
            SYS));
  if(Q < XMLRead.getExpection(i, j, SYS))
                   {
Q = Q_Temp;
          }
           }else {
                  Q = Q_Temp;

                  MOV_ARR[i][j] = Q;
        System.out.println("\n -- Changed Q  : " +MOV_ARR[i][j]);
                  }else if (Rand1 == 1) {
                   double Q_Temp = ARR[i][j];
                            double Q =  ARR[i][j];
                   Q = Q + Rand2;
if(Q >= SYS[i][4])
                  {                          Q = Q_Temp;}

        MOV_ARR[i][j]=Q;
        System.out.println("\n -- Changed Q  : " +MOV_ARR[i][j]);
                  }
                          }
                  }
```

```
                        }}      }

                        } while (isValidSol(MOV_ARR) == 0)
                                        return MOV_ARR;
                        }

private static double[][] geMappingMove(double[][] ARR,
        int M_param, int Q_param, int skip_step) {
                // TODO Auto-generated method stub
        Random generator = new Random(System.currentTimeMillis());
double [][] MOV_ARR = new double
    [XMLRead.getSysArrRows()][XMLRead.getNumECUs()];
for (int j = 0; j < XMLRead.getNumECUs(); j++) {
for (int i = 1; i < XMLRead.getSysArrRows(); i++) {
MOV_ARR[i][j] = ARR[i][j];}}
 do {
int Rand1 =  generator.nextInt((int) XMLRead.getSysArrRows()+ 1);
for (int i = 1; i < XMLRead.getSysArrRows(); i++) {
if (i == Rand1)  { int randomIndex =  generator.nextInt((int)
    XMLRead.getNumECUs());
System.out.println("\n ## The Mapping MOve MM
    RANd"+Rand1+"RANDomINDEX"+randomIndex+"\n" );
 if (SYS[i][1]== 0) {
for (int j = 0; j < XMLRead.getNumECUs(); j++) MOV_ARR[i][j]= 0 ;
MOV_ARR [i][randomIndex] = 1;
}else if(SYS[i][1] == 1) {
double old_value = 0;
for (int j = 0; j < XMLRead.getNumECUs(); j++)
if (MOV_ARR[i][j] != 0) {
old_value = MOV_ARR[i][j];
MOV_ARR[i][j] = 0;
                }
   MOV_ARR[i][randomIndex] = old_value;             }
 }

}while (isValidSol(MOV_ARR) == 0);

        return MOV_ARR;
                }

private static void printSOL(double[][]SOL) {
// TODO Auto-generated method stub
        for (int i = 1; i < XMLRead.getSysArrRows(); i++) {
        System.out.println("\n");
        for (int j = 0; j < XMLRead.getNumECUs(); j++) {
```

```
        System.out.println("The SoLUTions is  " + SOL[i][j]);

        }

        }

        }



private static void printMoreDetails(double[][] SOL) {
        // TODO Auto-generated method stub
        System.out.println(" \n ECU1 | ECU 2 ");
        System.out.println("-----------------");
        for (int i = 1; i < XMLRead.getSysArrRows(); i++) {
                System.out.println("\n");
                for (int j = 0; j < XMLRead.getNumECUs(); j++) {
                        if (SYS[i][1]== 0) {
                        System.out.format(" %.3f (%.3f) |", SOL[i][j],
                            SOL[i][j]
                                 * (float) SYS[i][4] / SYS[i][5]);
                        }
                        if (SYS[i][1]== 1) {
                        //System.out.print(" "+SOL[i][j]+" (
                            "+(float)SOL[i][j]/SYS[
                        // i][4]+" ) \t");
                        System.out.format(" %.3f (%.3f) |", SOL[i][j],
                        (float) SOL[i][j] / SYS[i][4]);
                        }
                        }
                }
                System.out.println("\n");
        }



        private static double[][] chooseMove(double[][][] NCM) {
                // TODO Auto-generated method stub
                double local_best_cost = 0;
                int chosen_move_index = 0;
                int chosen_move_index_oth = 0;
                int total_tabus = 0;
                                if (NCM.length == 0) {

                        System.out.println("\n NO ELEMENTs Length
                            :"+NCM.length);
```

```
                        return GLOBAL_BEST_SOL;
                }
        System.out.println("\n Printing candidate set \n");
                printCandSet(NCM);
                System.out.println("\n ############## Choosing Move
                   ################ \n"+NCM.length);
                for (int k = 0; k < NCM.length; k++) {

// Case 1.

             if (isTabu(NCM[k])== 0) {
                double [][] array = createParameterArray(NCM[k]);
                double CV = CostFuncCalc.getCostFuncValue(array);
                        if (CV > local_best_cost) {
                        local_best_cost = CV;
                chosen_move_index = k;
                                                        }
                System.out.println("\n This is CaSE 1 \n "+k);
                                        }
                    // Case 2.

   if (isTabu(NCM[k])== 1){
                total_tabus++;
   double [][] array1 = createParameterArray(NCM[k]);
                   double CV1 = CostFuncCalc.getCostFuncValue(array1);
                      if (CV1 > GLOBAL_BEST_COST) {
        local_best_cost = CV1 ;
                GLOBAL_BEST_COST = CV1;
                                chosen_move_index = k;
             }else if (CV1 > local_best_cost) {
                        local_best_cost = CV1;
                                chosen_move_index_oth = k ;
                                                        }
                          System.out. println("\n This is CaSE 2 \n  "+k);
                                        }


                                chosen_move_index = k;
                        }


                                  // Case.3
                        if (total_tabus == NCM.length) {
                                System.out.println("\n  This CaSE 3\n");
                                return NCM[chosen_move_index_oth];
        }
                                else
```

```
                                return NCM[chosen_move_index] ;
                }




        private static double[][] createParameterArray(double[][] SOL) {
                // TODO Auto-generated method stub
                double [][] param_array = new
                    double[XMLRead.getNumSRT()][XMLRead.getPDFLen() + 5];
                int param_index_row = 0;

     for (int j = 0; j < XMLRead.getNumECUs(); j++) {
for (int i = 1; i < XMLRead.getSysArrRows(); i++) {

                        if (SYS[i][1]== 1) {
        if (SOL[i][j] != 0) {
                double [] pdf = XMLRead.getPDF(i, j, SYS);
         System.out.print("\n\n i: "+ i +" j: "+(j+1));

                 for (int l=0;l< XMLRead.getPDFLen();l++)

         System.out.print("   | "+pdf[l]);

                        param_array[param_index_row][0]= i;
                                param_array[param_index_row][1] =
                                    SYS[i][2] ;
                                        param_array[param_index_row][2] =
                                            SYS[i][3];
                        param_array[param_index_row][3]= SOL[i][j];
                        param_array[param_index_row][4] = SYS[i][4];

                                        System.arraycopy(pdf, 0,
                                            param_array[param_index_row],
                                            5, XMLRead.getPDFLen());
                                        param_index_row++;
                                }
                         }
                }
                  }
                return param_array;
        }
```

```
        private static int isValidSol(double[][] SOL) {
                // TODO Auto-generated method stub
                int isValid = 1;
for (int j = 0; j < XMLRead.getNumECUs(); j++) {
                double Util = 0;
                for (int i = 1; i < XMLRead.getSysArrRows(); i++) {

                        if (SYS[i][1] == 0) {
                        Util = Util + SOL[i][j] * (SYS[i][4] / SYS[i][5]);

          System.out.println(" \nH SOL[i][j]    "+SOL[i][j]+" SYS[i][4]
              "+SYS[i][4]+ " SYS[i][5] "+SYS[i][5]);


                                }
                        if (SYS[i][1] == 1) {
                Util = Util + SOL[i][j] / SYS[i][4];
        System.out.println(" \n SOL[i][j]    "+SOL[i][j]+ " SYS[i][4]
            "+SYS[i][4]);

                                }
                        }
                        System.out.println(" \n " +Util);

                        if (Util > 1) {
                                isValid = 0;
                                System.out.println("\n Util Invalid");
                        }

                }


                        if(getCommunicationCost(COMM_ARR, SOL) > 1)
                        {
                                isValid = 0;
                                System.out.println("\n Bus Invalid");
                        }


                        if(isValid_Reconfig(SOL) == 0)
                        {
                                isValid = 0;
                                System.out.println("\n reconfig Invalid
                                    :(");
                        }
```

```
                    isValid =1;

                return isValid;
    }


public static double getTotalUtil(double[][] SOL) {
    return 0;
}



  private static int isValid_Reconfig(double[][] SOL) {
          // TODO Auto-generated method stub
          int isValid = 1;

           for (int j = 0; j < XMLRead.getNumECUs(); j++) {
                  double Util = 0.0;
                  double max_wcet = 0;
                  int max_wcet_index = 0;
                  int max_srt_index = 0;
                  double max_srt_et =0;


                  for (int i = 1; i <XMLRead.getSysArrRows(); i++) {

                          if ((SYS[i][1] == 0)&(SYS[i][8]== 1)) {
                                  if (SYS[i][4]> max_wcet) {
                                        max_wcet= SOL[i][j];
                                        max_wcet_index = i;
                                   }
                              }

                          if ((SYS[i][1] == 1) & (SYS[i][8] == 1))
                              {
                                 if (SOL[i][j] > max_srt_et){
                                        max_srt_et = SOL[i][j];
                                        max_srt_index = i;
                                 }
                          }

                  }

           for (int i = 1; i < XMLRead.getSysArrRows(); i++) {

           if ((SYS[i][1] == 0) & (SYS[i][8]== 1)) {
```

```
                        int  new_wcet = (int) ( SYS[i][4] + ((SYS[i][7] -
                            1)*SYS[i][6]) );
                        Util = Util + SOL[i][j] * ( new_wcet / SYS[i][5]);
                                System.out.println(" \n HHHAAARD SOL[i][j]
                                      "+new_wcet+ " SYS[i][4] "+SYS[i][4]);
                                }

          if ((SYS[i][1] == 1) & (SYS[i][8] == 1)) {
                        int new_Q = (int) ( SOL[i][j] + ((SYS[i][7] - 1)*
                           SYS[i][6]));
                Util = Util + new_Q / SYS[i][4];

        System.out.println("\n SSSOOOFT SOL[i][j]  "+ new_Q + " SYS[i][4]
            "+ SYS[i][4]);
                                }

                        }

                        System.out.println(" index HARD task  " +
                            max_wcet_index +  "  index SoFT TASK " +
                            max_srt_index);
                        double reconfiguration_time_hrt = 0;
                        double reconfiguration_time_srt = 0;


                         if (max_wcet_index != 0) {
                        reconfiguration_time_hrt =
                            XMLRead.getNumFaults()*((
                            SYS[max_wcet_index][4]/
                            SYS[max_wcet_index][7])/SYS[max_wcet_index][5]);
                                }
                         if (max_srt_index !=0) {
reconfiguration_time_srt = XMLRead.getNumFaults()*((
   SOL[max_srt_index][j]/SYS[max_srt_index][7])/SYS[max_srt_index][4]);
                        }
                         Util = Util + reconfiguration_time_hrt
                            +reconfiguration_time_srt;
                         System.out.println(" \n Util \n " + Util);

                         if (Util > 1) {
                                isValid = 0;
                        }
                }

                return isValid;
        }
```

```
private static double getCommunicationCost(int[][] COMM_ARR,
    double[][][] SOL) {
        // TODO Auto-generated method stub
        double BusUtil = 0;
        double totalCost = 0;

        for (int j = 1; j < XMLRead.getSysArrRows(); j++) {

                for (int i = 1; i < XMLRead.getSysArrRows(); i++) {
                        if ((isMappedOn(j, SOL)!= isMappedOn(i,
                            SOL)& isMappedOn(i, SOL)!= -1 &
                            isMappedOn(j, SOL)!= -1))
                        {
                                //System.out.print("\n "+j+" is
                                    mapped on "+isMappedOn(j,SOL)+"
                                    , "+i+" is mapped on "
                                    +isMappedOn(i,SOL)+
                                //"Comm Cost"+COMM_ARR[j][i]);
                                totalCost = totalCost +
                                    COMM_ARR[j][i];

        if (SYS[j][1] == 0) {    double msgSize =
            COMM_ARR[j][i]/XMLRead.getBusBW();
        BusUtil = BusUtil + msgSize /SYS[j][5];
System.out.print("\n BusUtil "+BusUtil);
                                }

                if (SYS[j][1] == 1) {
        double msgSize = COMM_ARR[j][i]/XMLRead.getBusBW();
                        BusUtil = BusUtil + msgSize /SYS[j][4];
                                        System.out.print("\n
                                            BusUtilSSSSSS
                                            "+BusUtil);
                                }

                        }
                }

        }

        System.out.println(" \n \n COMMUNICATION COST
            "+totalCost+ " BUS UTIL  "+BusUtil);
```

```
        return BusUtil;
    }




    private static int isMappedOn(int i, double[][] SOL) {
        // TODO Auto-generated method stub
         int ecu =-1;
        for (int i1 = 1; i1 < XMLRead.getSysArrRows(); i1++) {
                for (int j1 = 0; j1 < XMLRead.getNumECUs(); j1++) {
                        if (i1 == i & SOL[i1][j1]!= 0) {
                                ecu = j1;
                        }

                }

        }

        return ecu;
    }



    public static int[][] readComm(String Filename) {

        int [][] COMM_ARR = new
            int[XMLRead.getSysArrRows()][XMLRead.getSysArrRows()];
        try {
                System.out.println("reading the communication
                    CONFIg from the file \n");
                FileReader input = new FileReader(Filename);
                BufferedReader buffRead = new
                    BufferedReader(input);
                String Line ;
                 int count = 0;
                 Line = buffRead.readLine();
                 count++;
                 int col =1;

              while (Line != null){
                            col =1;
           System.out.println(count+": "+Line);
                                {
```

```
                        StringTokenizer st = new StringTokenizer(Line);
                                while (st.hasMoreTokens()) {
                                String tok = st.nextToken();

                               Integer f2 = new Integer(tok);
                                int d2 = f2.intValue();


                                        COMM_ARR[count][col]= d2;

                                            col++;

                                        }

                                    }
                                   Line = buffRead.readLine();
                                 count++;
                               }

                                  buffRead.close();
} catch (ArrayIndexOutOfBoundsException e) {
                        // TODO: handle exception
                        System.out.println("I cannot find the
                            COMMUNICATION FILE CONFIG");

                }catch (Exception e) {
                        // TODO: handle exception
                        e.printStackTrace();
                }


                return COMM_ARR;

        }


        private static String getDateTime() {
                // TODO Auto-generated method stub

                    DateFormat dateFormat = new
                        SimpleDateFormat("yyyy_MM_dd_HH_mm_ss");
                Date date = new Date();
                return dateFormat.format(date);
        }

}
```

A.0.3 Uunifast Algorithm Implementation

```
/*
 * Uunifast.cpp
 *  Created on: Nov 24, 2015
 *      Author: AMel BELAGGOUN
 */
#include "Uunifast.h"
#include<iostream>
#include<math.h>
#include<vector>
#include<stdlib.h>

using namespace std;

    std::vector<double> UUniFast(int n, double U_target)
    {
       // std::uniform_real_distribution<double> dist(0,1);

        std::vector<double> v;
        double sum = U_target;
        for (int i=1; i<=n-1; i++) {

            double next = sum*pow(rand(), 1/double(n-i));
            v.push_back(sum - next);
            sum = next;
        }
        v.push_back(sum);
        return v;
    }

    int  main() {
          std::vector<double> z;
        z=  UUniFast(41, rand());
        for (std::vector<double>::const_iterator i = z.begin();
        i != z.end(); ++i)
            std::cout << *i << "\n";
    return 0;
        }
```

A.0.4 Matlab Script for Generating Initial System Configuration

```
function GENERATE_INIT_Config(sys_util,sys_q,n_srt,n_hrt,n_ECU)
```

```
fid2 = fopen('initial_config.txt','w+');


ECU_UTIL = zeros(1,n_ECU);


SYS_ARR = zeros(n_srt + n_hrt,n_ECU);


for i = 1:n_srt+n_hrt

    [C I] = min(ECU_UTIL)    ;

        if(ECU_UTIL(1,I) + sys_util(i,I)<=2)

            if(i<=n_hrt)
            SYS_ARR(i,I) = 1;
            else
            SYS_ARR(i,I) = sys_q(i,I);
            end

            ECU_UTIL(1,I)=  ECU_UTIL(1,I) + sys_util(i,I);

        else

            SYS_ARR
            pause
            while(ECU_UTIL(1,I) + sys_util(i,I)>2)
                I = ceil(n_ECU * rand());
            end

       SYS_ARR(i,I) = sys_q(i,I);
        ECU_UTIL(1,I)=  ECU_UTIL(1,I) + sys_util(i,I);
        end

end


for i = 1 :n_srt+n_hrt
    sys1(n_hrt+n_srt-i+1,:) = SYS_ARR(i,:);
end


SYS_ARR=sys1;

for k= 1:n_ECU
```

```
    fprintf(fid2, ' 0 ');
end
fprintf(fid2, '\n');
for   i = 1:n_srt+n_hrt
for k= 1: n_ECU
    fprintf(fid2, ' %d ',SYS_ARR(i,k));
end
fprintf(fid2, '\n');
end




SYS_ARR

ECU_UTIL

(sum(ECU_UTIL)/n_ECU)*100


fclose(fid2);
```

Appendix B

B.0.1 AUTOSAR Software Development

1. **System configuration:** The system consists of two SWCs as shown in Figure B.1. We concentrate on an extract of the configuration and some parts of the source code for one SWC, thus, the ECU configuration is not shown in this example.

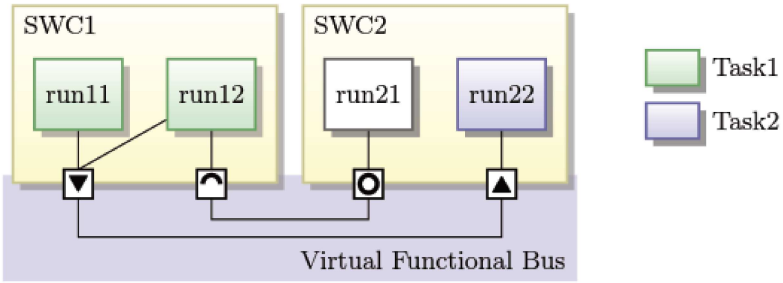

Fig. B.1 System Configuration.

Take note that the whole configuration of the system can be found in appendix A. In this configuration every SWC has two runnables run11,run12 and run21,run22 respectively. . The runnable run11 is triggered by a timing event that occurs every 100ms. Runnable run12 is triggered by another timing event that occurs every 50ms. run11, run121 are mapped to the task Task1 and they are running in the context of the same task. The other two runnables of SWC2 i.e. run22 and run21 also triggered by an timing event every 50ms and by the invocation of a server call at the server port respectively. We assume that Both SWC1 and SWC2 are mapped on the same ECU. The definition of the data types used in this example is shown in Code Listing B.2. As we can see the first type is the integer type Int16 with a specified range of valid values. The second type is a string type, which has a length of 8 characters and the symbol String8. In order to reference a type, the names of the packages have to be used as path. For example the path /types/Int16 can be used to reference the integer type.

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns="http://autosar.org/4.1.2">
<TOP-LEVEL-PACKAGES>
<AR-PACKAGE>
<SHORT-NAME>types</SHORT-NAME>
<ELEMENTS>
<INTEGER-TYPE>
<SHORT-NAME>Int16</SHORT-NAME>
<LOWER-LIMIT INTERVAL-TYPE="CLOSED">-32768</LOWER-LIMIT>.
<UPPER-LIMIT INTERVAL-TYPE="CLOSED">32767</UPPER-LIMIT>
</INTEGER-TYPE>
<STRING-TYPE>
<SHORT-NAME>String8</SHORT-NAME>
<ENCODING>utf8</ENCODING>
<MAX-NUMBER-OF-CHARS>8</MAX-NUMBER-OF-CHARS>
</STRING-TYPE>
</ELEMENTS>
</AR-PACKAGE>
</TOP-LEVEL-PACKAGES>
</AUTOSAR>
```

Fig. B.2 Data types description.

In the same way the interface of the sender–receiver port needs to be described as shown in the Code Listing B.3. Here, the interface SR Int16 consists of two data

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns="http://autosar.org/4.1.2">
<TOP-LEVEL-PACKAGES>
<AR-PACKAGE>
<SHORT-NAME>interfaces</SHORT-NAME>
<ELEMENTS>
<SENDER-RECEIVER-INTERFACE>
<SHORT-NAME>SR Int16</SHORT-NAME>
<IS-SERVICE>false</IS-SERVICE>
<DATA-ELEMENTS>
<DATA-ELEMENT-PROTOTYPE>
<SHORT-NAME>intValue1</SHORT-NAME>
<TYPE-TREF DEST="INTEGER-TYPE">/types/Int16</TYPE-TREF>
<IS-QUEUED>false</IS-QUEUED>
</DATA-ELEMENT-PROTOTYPE>
<DATA-ELEMENT-PROTOTYPE>
<SHORT-NAME>intValue2</SHORT-NAME>
<TYPE-TREF DEST="INTEGER-TYPE">/types/Int16</TYPE-TREF>
<IS-QUEUED>false</IS-QUEUED>
</DATA-ELEMENT-PROTOTYPE>
</DATA-ELEMENTS>
</SENDER-RECEIVER-INTERFACE>
</ELEMENTS>
</AR-PACKAGE>
</TOP-LEVEL-PACKAGES>
</AUTOSAR>
```

Fig. B.3 Interface description.

types intValue1 and intValue2, which are both of type Int16, and it provides a not queued communication.

In the Code Listing B.4, the SWC1 description is given. SWC1 has two runnables and ports that are accessed by the runnables via DataSendPoints and a SynchronousServer-CallPoint This is needed to generate the API and to provide consistency for the communication. The runnable run11 writes both values to the sender port pport11, whereas run12 only writes the value intValue1. Both runnables are triggered by the specified timing events.

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns="http://autosar.org/4.1.2">
<TOP-LEVEL-PACKAGES>
<AR-PACKAGE>
<SHORT-NAME>swc root</SHORT-NAME>
<ELEMENTS>
<ATOMIC-SOFTWARE-COMPONENT-TYPE>
<SHORT-NAME>swc1</SHORT-NAME>
<PORTS>
<P-PORT-PROTOTYPE>
<SHORT-NAME>pport1</SHORT-NAME>
<PROVIDED-INTERFACE-TREF y
DEST="SENDER-RECEIVER-INTERFACE">/interfaces/SR Int16</PROVIDED-INTERFACE-TREF>
</P-PORT-PROTOTYPE>
<R-PORT-PROTOTYPE>
<SHORT-NAME>rport1</SHORT-NAME>
<REQUIRED-INTERFACE-TREF y
DEST="CLIENT-SERVER-INTERFACE">/interfaces/CS string to int</REQUIRED-INTERFACE-
TREF>
</R-PORT-PROTOTYPE>
</PORTS>
</ATOMIC-SOFTWARE-COMPONENT-TYPE>
</ELEMENTS>
</AR-PACKAGE>
</TOP-LEVEL-PACKAGES>
</AUTOSAR>
```

Fig. B.4 Software Component description.

2. **Source code generation:** The next step after the SWCs description is calling the RTE generator with that configuration in order to produce auto generated code of the RTE. This code contains the API for the SWCs. The extract for SWC1 is shown in the Code listing **??**. In the last two lines the functions for the runnables are declared. These functions are called when a runnable is executed (i.e. when the timing events occur). Developers must implement first these functions in order to implement the functional behavior of the SWC. In AUTOSAR a sender port with a DataSendPoint is accessed via the API specified in the following form: Rte_Write_<p>_<o>(data). Where <p> is the name of the port and <o> is the name of the data element which is accessed. In other words the value which is passed to the call with the parameter data, is written to the port. It can be seen from the example that r12 uses the specified function call SynchronousServerCallPoint to invoke an operation at the server. So the function does not return until the operation on the server is finished and the return values are available. For the SynchronousServerCallPoint, the API Rte_Call_rport1_parse (string , value) is expected and created by the RTE generator. The operation parse takes one string as argument and returns one integer value. This API is implemented using a #**define** directive to a function; which directly accesses the global variables Rte_RxBuf_1 and Rte_RxBuf_2. This function is implemented also in the generated file Rte.c. The source code for the tasks is created by the RTE generator. We present the extract for Task1 in the code listing B.6. As we said in the example specification it can be seen that r11 is triggered every 100ms whereas r12 every 50ms. Once the task is released the runnable is executed. Releasing tasks is done in the operating system configuration, the latter is created by the RTE generator.

```
#define RTE_E_CS_string_to_int overflow (42)
#define RTE_E_CS_string_to_int_underflow (43)

#define Rte_Call rport1 parse(array, sum) (Rte_Call_swc1_rport1_parse(array, sum))
FUNC(Std_ReturnType, RTE_CODE)
Rte_Call_swc1_rport1_parse(CONSTP2VAR(String8, AUTOMATIC, RTE_APPL_DATA),
CONSTP2VAR(Int16, AUTOMATIC, RTE_APPL_DATA));

/* Inline Write optimization; Rte_Write_pport1_intValue2_to_direct_access */
extern VAR(Int16, RTE DATA) Rte_RxBuf_1;
#define Rte_Write_pport1_intValue2(data)
Rte_WriteHook_swc1_pport1_intValue2_Start(data), (Rte_RxBuf_1 = data),
Rte_WriteHook_swc1_pport1_intValue2 Return(data), RTE_E_OK )

/* Inline Write optimization; Rte_Write_pport1_intValue1_to_direct_access */
extern VAR(Int16, RTE_DATA) Rte_RxBuf_0;
#define Rte_Write pport1 intValue1(data)
Rte_WriteHook _swc1_pport1_intValue1_Start(data), (Rte_RxBuf_0 = data),
Rte_WriteHook_swc1_pport1_intValue1_Return(data), RTE_E_OK )

FUNC(void, RTE_APPL_CODE) run11(void);
FUNC(void, RTE_APPL_CODE) run12(void);
```

Fig. B.5 Example header file for SWC1.

```
TASK(Task1)
{
Rte RECount_Task1_divby2 _0;
if ( Rte_RECount_Task1_divby2_0 == 0 )
{
run11();
}
{
run12();
}
if ( Rte_RECount_Task1_divby2 0 == 0 )
{
Rte_RECount Task1_divby2_0 = 2;
}
TerminateTask();
}
```

Fig. B.6 Example code for task1

3. **Runnables implementation:** The last step in developing the SWC is the runnable implementation this step is shown at the end of code listing B.7. In which The

```
FUNC(void, RTE_APPL_CODE) run12(void)
{
String8 val1;
Int16 val2;
...
 Rte_Call_rport1_parse(val1, &val2);
...
Rte_Write_pport1 _intValue1(val2);
...
}
```

Fig. B.7 Example code for runnable21.

created API can be used to access the ports. Here it is just necessary to write the correct API to the source code. This code can then be compiled with the generated header file. Instead of the global variables for the senderreceiver communication shown in this example, also a function call can be used. The runnable would not

get noticed about it. This also shows the first problem for sharing object code. If a component is compiled against such a header file with global variables, it is expected that a generator, which creates the whole API, does not create the same global variable. So the object code cannot be used. Therefore function calls have to be created for sharing object code.But this causes an overhead for the function invocation when accessing a port.

B.0.2 The Characteristics of AUTOSAR Runnables

Table B.1 shows the relevant characteristics of a runnable that need to be taken into account when addressing runtime adaptation, i.e. Category, activation, events. The most important characteristics are *Activation Mode* and *Category*.

Table B.1 Characteristics of AUTOSAR Runnables.

Concept	**Characteristic**	**Details**
Activation mode	Periodic / Sporadic	A runnable is either periodic or triggered by events.
Communication	Input and Output Data and corresponding access mode	Corresponds to communication needs of the runnable. Required to plan ahead the proper connections. Linked to the runnable category.
Category	1: No wait point 2: Wait Point	A wait point is the moment when a runnable wait for an external event to resume its execution

B.0.3 ASLA's Task Adaptation Algorithm–The Case for Replacing tasks

Similarly to Algorithm 3, ASLA's reconfiguration manager uses Algorithm 4 to replace a SWC with another one that have an improved version; the new SWC has the same interface, WCET and the same communication pattern however it has an improved internal behavior (i.e., runnables). Note that the task need to be in suspended State in order to replace safely the SWC .

ALGORITHM 4: TSeRBA for replacing SWC_{old} by SWC_{new}

As soon as the replacement request is triggered.
O-TSMBA (Γ^c, ECUs)
add ($\mathsf{SWC_{new}}$, ECUs)
for all $\mathsf{SWC_i} \in \Gamma^c$ **do**
  if feasible mapping **then**
    if link($\mathsf{SWC_i}$, $\mathsf{SWC_j}$) **then**
      update input port $\mathsf{P^{out}_{swc_j}}$ of $\mathsf{SWC_j}$ to connect to $\mathsf{P^{out}_{swc_{inew}}}$ of $\mathsf{SWC_{inew}}$
      suspend($\mathsf{SWC_{old}}$)
      State-transfer($\mathsf{SWC_{old}}$, $\mathsf{SWC_{new}}$)
      Link($\mathsf{SWC_{inew}}$, $\mathsf{SWC_j}$)
      Unlink ($\mathsf{SWC_{iold}}$, Γ^c)
      Remove ($\mathsf{SWC_{iold}}$, Γ^c)
    end if
  else
    Adjust Qs Proportionally($\mathsf{\Gamma^c}$, ECU_j)
    $recompute\chi$
  end if
end for

B.0.4 ASLA's Task Adaptation Algorithm–The Case for Removing Tasks

Similarly to Algorithm 3. ASLA's reconfiguration manager uses Algorithm 5 to remove task from the ECUs.

ALGORITHM 5: TSeRBA for removing tasks

As soon as the deletion request is triggered
O-TSMBA (Γ^c, ECUs)
$unlink$($\mathsf{SWC_{ri}}$, $\mathsf{SWC_{rj}}$, $\mathsf{P^{out}_{ri}}$, $\mathsf{P^{in}_{rj}}$)
Remove (SWC, $\mathsf{\Gamma^c}$)