Design and Analysis of Task Models for Mixed-Criticality Systems with Practical Considerations

Vijaya Kumar Sundar

School of Computer Science & Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

2022

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

10-01-2022

. . . . . .

Date

. . . . . .

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

10-01-2022

. . . . . .

Date

. . . . . .

Authorship Attribution Statement

This thesis contains material from three papers published in the following peer-reviewed journals / from papers accepted at conferences in which I am listed as an author.

Chapter 4 is published as *Vijaya Kumar Sundar, Arvind Easwaran. A Practical Degradation Model for Mixed Criticality Systems. 23rd IEEE International Symposium on Real-Time Distributed Computing (ISORC), Valencia, Spain.*

The contributions of the co-authors are as follows:

- Assoc. Prof. A. Easwaran provided the initial project direction, feedback on methodology and edited the manuscript drafts.
- I designed the model, derived the equations, designed and implemented experiments, and wrote the manuscript.

Chapter 5 is under review as *Vijaya Kumar Sundar, Saravanan Ramanathan and Arvind Easwaran. Design and analyses of functional mode changes for mixed criticality systems, Real-Time Systems Journal.*

The contributions of the co-authors are as follows:

- Assoc. Prof. A. Easwaran provided the initial project direction, feedback on methodology and edited the manuscript drafts.
- Saravanan Ramanathan provided strategy for task set generation.
- I designed the model, derived the schedulability test and algorithms, conducted experiments, generated the results and wrote the manuscript.

Chapter 6 is published as *Vijaya Kumar Sundar and Arvind Easwaran. Testbed for practical considerations in mixed criticality systems. Brief Presentations Proceedings 25th IEEE Real-Time and Embedded Technology and Applications Symposium, Montreal, Canada.*

The contributions of the co-authors are as follows:

- Assoc. Prof. A. Easwaran provided the initial project direction.
- I decided on the architecture of the testbed, implemented the applications, conducted the experiments, generated the results and wrote the manuscript.

10-01-2022

. . . . . .

Date

. . . . . .

Acknowledgements

First and foremost, I would like to express my utmost gratitude to my advisor Arvind Easwaran for his unwavering support throughout the course of these four years. His methodical approach to problem-solving elucidated the art of conducting independent research to me. The discussions we had were always insightful and highly enjoyable. This extends beyond academics - well into general topics, especially during our travels. I am extremely thankful to him for empowering me with space and freedom to develop intellectually. It has simply been an honor and a pleasure to work with him. Words alone do no justice.

In addition to my thesis supervisors, I would also like to thank my thesis advisory committee members for the valuable suggestions and encouragement that I received.

I would also like to extend my gratitude to the jury, for reviewing the thesis. Their constructive comments are integral for elevating the quality of this thesis.

It has been a great pleasure to spend four years at Nanyang Technological University. I take this opportunity to thank all members of the Cyber-Physical Systems research group, for which I have thoroughly enjoyed the banter and learn a lot from - Saravanan, Sriam, Ankita, Zahra, Daniel.

Last but not least, I would like to express my heartfelt gratitude to my parents, grand parents and brother for their unconditional love and support especially during the most trying periods of this endeavor. They are my motivation to persevere and persist against the odds.

"If you resist change, you resist life."

— Sadhguru, Jaggi Vasudev

To my dear family

Abstract

The current trend in designing safety-critical real-time embedded systems is to consolidate applications performing functionalities with varied levels of safety integrity requirements onto a common platform. These systems are commonly referred to as Mixed Criticality Systems (MCS) and are found in safety-critical domains such as avionics and automotive. The main goals of MCS are to provide high reliability in execution for safety-critical software and at the same time to strike a balance between the pessimistic resource reservations for safety versus efficient resource utilization for cost-effectiveness. Such systems are also required to adapt their functionality depending on their operating mode. To achieve these goals, the application designer relies on task models to provide parameters such as period, deadline, budgets, and criticality of tasks, to denote the required timing behaviour which is later verified by applying mathematical techniques to ensure that the resource allocated for tasks are sufficient to meet their deadlines, even under the occurrence of system overload due to timing faults such as budget overrun of a task.

Safety standards like ISO 26262 (functional safety standard for automotive) consider four to five levels of criticality. But, to simplify the complexity of the analysis involved in verifying the timing behaviour, a majority of the existing MCS task models consider only two criticality levels (high and low) for tasks. Although such a simplified model can capture fundamental behaviour of MCS, these models face criticisms for their approach to handling system overload by suspending or degrading tasks with lower criticality than the overloading tasks. Such a degradation strategy may not be safe for those systems which support more than 2 criticality levels. As MCS can also undergo functional mode changes at run-time, a system overload can occur due to a budget overrun of a high criticality task and/or due to a spike in the resource consumption pattern leading to a temporary system overload when tasks belonging to different modes execute together for a certain time interval when system switches from one mode to another. Most real-time system models that focus on functional mode changes do not focus on capturing mixed-criticality aspects of the system. To address these issues, graph-based MCS task models were proposed as they can provide a higher level of flexibility in the choice of degradation of tasks and to model functional mode change aspects, but such models are harder to analyse.

In this thesis, as a first step, we present the Context-Aware Mixed Criticality System (CA-MCS)

model that is expressive enough to handle system overload caused due to budget overruns of tasks while operating in a specific mode. The model is motivated from the guidelines provided by ISO 26262 and from case studies related to the automotive domain that clearly show that a task can be degraded in multiple ways, and it is possible to consider performance degradation of higher criticality task instead of suspension or degradation of the core functionality of lower criticality tasks. By considering these possibilities, the CA-MCS model is designed in such a way that it can provide a higher level of flexibility to choose the tasks to be degraded and the specific way in which they can be degraded. Further, the criticality information of a task is used only to decide on the degraded budget when the budget overrun of multiple tasks occur.

As a second step, we focus on MCS that can undergo functional mode changes and present the Multi-Mode Mixed Criticality System (MM-MCS) model. The proposed model is expressive enough to capture parameters to determine the budget, release patterns and degradation of tasks during the mode transition, and considers both task degradation and the notion of offsets to handle the system overload. Further, the MM-MCS model establishes precise rules to handle budget overrun of tasks both within a mode and during mode transitions. Additionally, an algorithm to compute the offset values for new-mode tasks based on their criticality value is also derived.

The fixed-priority based schedulability tests for a uniprocessor system is proposed for CA-MCS and MM-MCS models with complexity still being pseudo-polynomial with respect to the number of tasks. The experimental results based on synthetic task sets for these tests show the benefit of considering offsets and task degradation to improve the schedulability performance. Further, the algorithm presented to compute offsets clearly shows that criticality information can play an important role not only for achieving task degradation but also for determining suitable offset values for higher criticality tasks.

As a third step, we present a realistic automotive testbed that is designed and implemented with automotive applications such as Adaptive Cruise Control (ACC), Steering Control (SC) and Collision Avoidance (CA). The main objective is to observe the impact of different degradation strategies adopted by MCS models on the performance and safety aspects of these applications. Experimental results show that the proposed models can give the ability to degrade the performance of the system in a controlled manner by isolating the effects of degradation between safety applications. The testbed is highly flexible, scalable, can facilitate the implementation of any new MCS task models, and can be included as a lab exercise in the university courses related to real-time systems or automated driving.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ACC	Adaptive Cruise Control
CAA	Collision Avoidance Application
MCR	Mode Change Request
WCET	Worst Case Execution Time
ASIL	Automotive Safety Integrity Level
MS-DRT	Mode Switching Digraph Real Time System
WCRT	Worst Case Response Time
CA-MCS	Context Aware Mixed Criticality System
MM-MCS	Multi Mode Mixed Criticality System
DF	Degradation Factor
CF	Criticality Factor
MCS	Mixed Criticality Systems
BO	Budget Overrun

Chapter 1

Introduction

1.1 Real-Time Embedded Systems

In this era of automation, humans rely heavily on machines to perform their tasks. Technological advancements in the field of mechanical, electrical, electronics and artificial intelligence, has led to the development of systems capable of mimicking a wide range of activities corresponding to physical and intellectual planes of human existence. Further, capabilities of such systems to arrive at smart decisions is made possible by integrating their computation with physical processes by making use of the data not only from their perception capabilities but also from other systems connected via networks. As their behaviour is defined by both cyber and physical parts, such systems are commonly referred as 'Cyber-Physical Systems' [2]. In those lines, embedded and cyber-physical systems rely on a dedicated computing platform to perform a specific functionality and have become a common entity ranging from digital watches to aeroplanes.

Real-time systems are commonly found in automotive and aviation industries to perform both safety-critical and non-critical functionalities. Such systems rely on both logical correctness and the timeliness in their execution to meet the safety requirements [3]. The timeliness here refers to the ability of the system to produce the desired result within a certain time interval normally referred to as the deadline. Depending on the consequence of missing a deadline, these systems can be classified as hard, firm and soft real-time systems. The objective of a hard real-time system is to ensure that no deadlines are missed, but for firm real-time systems infrequent deadline misses are tolerable [4]. In both cases, their results are not useful after their deadline. In soft real-time systems, the usefulness of a result decreases after its deadline, thereby degrading the system's quality of service. An automotive application such

as Collision Avoidance Application (CAA) can be considered as hard real-time since it has to apply maximum possible braking immediately after detecting the possibility of a collision, failing which it might lead to a disastrous effect. A navigation or entertainment system can miss certain values leading to decreased performance. Unlike entertainment systems which are soft real-time, a delayed value is of no use in navigation systems making it firm real-time systems. In this thesis, we focus on the hard real-time system.

1.1.1 Real-Time Task Models

In real-time systems, the functionalities of an application are generally implemented as a set of tasks. Every task requires sufficient computational resources to achieve the desired result. A task model consists of parameters to capture the timing constraints such as period, execution time and deadline. The difference in the existing task models can be found from the definition of the way in which the available time, execution time and deadline follow. A *real-time job* is the simplest task model. In this model, a job, represents a piece of code and is available at a specific time instant termed as its release time, and is required to finish its execution by another specific time termed as its deadline.

Real-time sporadic tasks. An example of real-time task model is sporadic task model. A sporadic task τ_i is specified as τ_i = (C_i, T_i, D_i), where C_i is the worst-case execution time (WCET), T_i is the time period and D_i is the deadline. The specification of the period and the deadline for the task is done considering the required performance of the control algorithm implemented by the task. A sporadic task τ_i can release an infinite sequence of jobs $< J_i^1, J_i^2, J_i^3, \ldots >$ with a constraint that a minimum inter-arrival time between any two consecutive job releases must be equal to the period T_i. The worst-case execution time C_i specifies the known upper bound of the execution time of task τ_i, and is usually measured very conservatively to ensure the safety of the system. Suppose the release time of job J_i is $r(J_i)$. J_i must complete by its absolute deadline $d_i = r(J_i) + D_i$, and in the worst-case it could execute up to C_i time units. All jobs of the task τ_i are required to complete by their deadline, otherwise it is regarded as system failure.

1.1.2 Real-Time Task Scheduling

The scheduling problem is to deal with the allocation of the resources to satisfy the required timing behaviour of tasks (meet deadlines). In the real-time systems literature, task scheduling can be categorized as static and dynamic scheduling. In static scheduling, the scheduling decisions are made off-line.

A schedule is generated off-line based on the knowledge of system parameters. Given a set of jobs $\{J_1, J_2, J_3, \ldots\}$, where $J_i = (r(J_i), C_i, d_i = r(J_i) + D_i)$, static scheduling will construct a schedule in which each job meets its deadline (a feasible schedule). On the other hand, dynamic scheduling is flexible and adaptive, and it determines which job to run next at run-time based on active jobs at that time. For this purpose, several scheduling algorithms that switch tasks and allocate resources at run-time are studied.

Based on the specifications provided in the task model, any scheduling algorithm must assure apriori that all tasks are completed by their deadlines. These guarantees must be analytically evaluated before the actual execution of the system. Hence, these algorithms are classified into two types:

- *Scheduling policies* are algorithms that will be executed along with real-time tasks and control the run-time schedule. Hence, they are also called schedulers. These algorithms make scheduling decisions at run-time based on the time and/or the temporal behavior of real-time tasks. In real-time systems, schedulers are classified into fixed-priority (*e.g.,* Rate Monotonic) and dynamic-priority (*e.g.,* Earliest Deadline First) schedulers. With *fixed-priority schedulers*, priorities are assigned at the task level *i.e.* all jobs of the same task have the same priority. With *dynamic-priority schedulers*, priorities are assigned at the job level *i.e.* different jobs of the same task can have a different priority based on their deadline or any other parameter.

- *Schedulability tests* are the algorithms that are used to verify the temporal correctness of the real-time system at design time. A schedulability test for a scheduling algorithm is said to be exact if it can identify all systems schedulable by that scheduling algorithm, and sufficient if it can identify only some schedulable systems that are schedulable by that scheduling algorithm.

The proposed MCS task models considered in this thesis are motivated from the automotive domain. It is observed that Fixed Priority Scheduling (FPS) is the de facto standard in the automotive industry and it is the scheduling algorithm considered in the industry standard OSEK/AUTOSAR (AUTOmotive Open System ARchitecture) operating system [5]. Therefore, in this thesis, only fixed-priority based sufficient schedulability tests for preemptive systems are considered. *Preemptive* means that at any time, the scheduling policy can interrupt the currently executing job, and choose another job to execute. The additional time due to preemption in the form of context and state saving is assumed to be bounded by the worst-case execution time.

1.1.3 Mode Changes In Real-Time Systems

Sometimes, a real-time system may be exposed to different operational situations and hence the required functionality also depends on the operational situation. These operational situations are termed as modes. A typical example is that of an automotive control system where depending on the vehicle speed, modes are classified as low-speed or high-speed. Functionalities like Anti-lock Braking System (ABS) and Adaptive Cruise Control (ACC) are activated only in high-speed mode, whereas automatic parking assist and the low-speed emergency braking are activated in the low-speed mode.

A *Mode* is defined by a specific set of tasks whose parameters are configured to achieve a unique set of perceivable safety and/or performance functionalities of the system. A system can undergo a transition from the origin mode (old-mode) to the designation mode (new-mode) due to change in environment or internal state of the system. For example, a task monitoring wheel speed sensors may issue Mode Change Request (MCR) for the transition from low-speed to the high-speed mode.

Definition 1 *A Mode Change Request (MCR) is defined as the event that triggers mode transition and is usually issued by a task which has knowledge of environment or system's internal state.*

Sometimes, MCRs are also issued by interrupt service routines whenever a button or switch is pressed. MCR may result in significant changes in the number of tasks and their parameters such as execution time, period, deadline etc., leading to a temporary spike in the resource consumption pattern. Hence, studies [6–9] have dealt with modelling and analysis for mode transitions in a real-time system that is considered necessary to determine whether a task can meet its deadline during the mode transitions. To ensure that tasks get enough resource for their execution during the mode transition, these models also consider the notion of *offsets* which is a time delay applied to the first activation of tasks occurring in the new-mode after the MCR.

1.2 Mixed Criticality Systems

Real-time systems that are designed to perform safety-critical functionalities such as CAA, are subjected to rigorous testing and certification process to avoid unreasonable risks or hazards. In general, hardware and software components of these systems undergo a risk analysis process and are assigned with a specific criticality level. The criticality refers to the level of assurance and safety integrity against failure needed for a component. In practice, up to five such levels are considered in functional safety standards such as IEC 61508, DO-178B, DO-254 and ISO 26262 [10–13]. These levels are referred

to as DALs (Design Assurance Levels), ASILs (Automotive Safety Integrity Levels), and SILs (Safety Integrity Levels). For example, in ISO 26262 [14], a functional safety standard for automotive, ASIL is used as a risk classification scheme to denote the impact level of each possible hazard associated with the failure of the component, which is later used to derive the necessary functional and technical safety requirements. A "quality managed" (QM) rating denotes that the hazard is not severe enough to require specific regulations through the standard, those that are will be given a value of ASIL A through ASIL D. ASIL is established for hardware and software components by performing a risk analysis by looking at classes of severity, probability of exposure and controllability factors of the hazard for a vehicle operating scenario. The decision chart shown in Figure 1.1 shows the ASIL and different classes for these three factors.

Severity	Probability of Exposure	Controllability		
		C1 - Easy	C2 - Normal	C3 - Very Difficult
S1 - Light	E1 (very low)	QM	QM	QM
	E2 (low)	QM	QM	QM
	E3 (medium)	QM	QM	A
	E4 (high)	QM	A	B
S2 - Severe	E1 (very low)	QM	QM	QM
	E2 (low)	QM	QM	A
	E3 (medium)	QM	A	B
	E4 (high)	A	B	C
S3 - Fatal	E1 (very low)	QM	QM	A
	E2 (low)	QM	A	B
	E3 (medium)	A	B	C
	E4 (high)	B	C	D

QM – Quality Managed

Figure 1.1: ASIL Decision Chart.

The current trend in the design of real-time embedded systems is to integrate components (hardware or software) with different levels of criticality on a common hardware platform. Such a system is commonly referred to as Mixed Criticality System (MCS) [15]. The design of MCS is motivated from the need to cope with the increase in the number of safety-critical and non safety-critical functionalities to be implemented with minimal and powerful hardware. Advancements in hardware platforms with increased processing capacity offered by multi/many core architectures with optimal size, weight and power characteristics, have allowed a designer to include additional functionalities. This can be seen from industry based initiatives such as ARINC653 [16], Integrated Modular Avionics (IMA) in avionics and AUTOSA in automotive.

According to functional safety standards such as ISO 26262, a designer has to show evidence for suf-

ficient isolation between components of lower and higher criticality levels using techniques employed at both hardware and software level for the system to get approved by the Certifying Authorities (CAs). Isolation is required to prevent the possibility of migration of failures from low critical to high critical components as they share the same platform. For example, imprecise estimation of the execution time of a software component can lead to an execution overrun at run-time. Such events can lead to starvation of the processor cycles thereby preventing the execution of other safety-critical software. This can lead to catastrophic consequences. Therefore, to certify a system to be correct, the software performing safety-critical functionalities are assessed with very conservative assumptions that are unlikely to occur in practice. Although from a certification perspective such assumptions can ensure that resources allocated for safety-critical software are always sufficient, in reality, resources end up being severely under-utilized.

1.2.1 Modelling Mixed Criticality Systems

The real-time task models for MCS, are motivated from the need to handle conflicting requirements of efficient resource usage and sufficient isolation between components of different criticality. The task model widely used in previous works (e.g., [15, 17–20]) is a simple extension of the classic sporadic task model [21] to a mixed-criticality setting where each task, τ_i, is defined by its period, deadline, WCET and criticality level: (T_i, D_i, C_i, L_i). $L_i \in L$ where $L = \{A, B, C, D\}$ denotes an ordered set of criticality levels with a total order relation $<$ between its elements. D denotes the highest criticality level. These parameters are considered derived by a process recommended by safety standards for the criticality level of tasks. Therefore, when considering execution time for tasks, MCS task models are based on the conjecture that higher criticality levels demand more stringent verification process and hence result in higher values of C_i. An important assumption considered in the MCS task models is that higher criticality task may not execute till its C_i always and therefore to improve the processor utilization, multiple execution time estimates in addition to the WCET for higher criticality tasks were considered. As they were assumed to be estimated at a criticality level lower than the criticality of the task, their values were also assumed to be equal to or lower than the WCET. Therefore, in Vestal's model [15], each task τ_i, is specified by a tuple (T_i, D_i, $\vec{C_i}$, L_i), where $\vec{C_i}$ is a vector, the m-th coordinate of which denotes the execution time estimate of τ_i at criticality level m where $C_i(L_i)$ denotes the WCET of τ_i. At run-time, any task τ_i is not allowed to execute beyond its $C_i(L_i)$. Depending on the execution behaviour of tasks, the system is assumed to execute in a specific criticality level $l \in L$. To assure that each task τ_i will never miss its deadline while the system is executing at a criticality level l, the

analysis for each task τ_i is carried out using execution estimate having the same level of assurance l *i.e.* $C_i(l)$. Some studies such as [22] have extended the notion of criticality based pessimism to other task parameters such as time period and deadline where a task may have a lower time period/deadline estimate if it is considered to be safety-critical rather than non-critical.

The values in $\vec{C_i}$ are such that $C_i(D) \geq C_i(C) \geq C_i(B) \geq C_i(A)$. Therefore, $C_i(D)$ is considered as the WCET for a task in MCS which is its known upper bound on its execution time. All other execution time estimates are used as *budgets* for tasks. Based on this notion of budget, service level for a task can be defined as follows,

Definition 2 (Service Level) *The service level of a task is a metric that denotes the amount of service provided to a task in a real-time system. For a task τ_i with an implicit deadline (deadline equal to period), we assume that its service level (SL_i) is determined by the time period (T_i) and the budget (C_i) allocated to it. SL_i is computed using the formula $SL_i = C_i/T_i$.*

Definition 3 (Service Level Violation) *A task τ_i is said to violate its allocated service level SL_i when the τ_i executes with a time period shorter than its assigned time period T_i or when τ_i executes more than its assigned budget C_i.*

When a task executes beyond its WCET, it is regarded as an error. Vestal's work led to further development in both MCS modelling and fixed-priority analysis. The model in [23] is well known because it captures fundamental mixed-criticality behaviours with minimum parameters. In this model, each task, τ_i, is specified by a tuple (T_i, D_i, $\vec{C_i}$, $\mathcal{L}_i$). T_i denotes the time period, D_i denotes deadline, $\mathcal{L}_i$ denotes criticality and $\vec{C_i}$ denotes a vector of execution time estimates. The criticality of tasks can be either HI or LO. This model assumes that each task with criticality HI (high criticality task) can have a WCET estimated at HI criticality (denoted as $C(HI)$) and an additional execution time estimated at low criticality (denoted as $C(LO)$). Each task with LO criticality (low criticality task) is assigned only with its WCET estimated at LO criticality. If $\mathcal{L}_i = HI$, then $\vec{C_i} = (C(LO), C(HI))$. Otherwise, if $\mathcal{L}_i = LO$, $\vec{C_i} = (C(LO))$ with the constraint: $C(HI) \geq C(LO)$.

Definition 4 (System Overload) *Consider a system represented as S consisting of n tasks denoted as $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$. S is considered to be overloaded if no scheduling algorithm, either off-line or on-line, can meet the deadlines of all the tasks that are submitted to the system S.*

A *system overload* for the safety-critical system can occur either due to service level violation of a task or due to a spike in the resource consumption pattern when tasks belonging to different modes execute together for a certain time interval when the system switches from one mode to another [7].

1.3 Motivation and Problem Statement

1. **Question 1: How to handle system overload due to service level violations, in a mixed criticality system, in a fair manner with a higher level of flexibility in the choice of tasks to be degraded, without compromising on system's safety?** Existing MCS task models (reviewed in Chapter 3) consider different strategies to handle system overload that can occur at run-time when any high criticality task violates its service level. These models try to achieve graceful degradation of the system either by suspension or degradation of tasks with relatively lower criticality than the overloading task. Many issues arise when such a degradation strategy is extended to systems with more than 2 criticality levels. As pointed out in [24], the notion of criticality as the relative importance of a task may be appropriate only for dual-criticality MCS task models (where a task can be either high critical or low critical). For instance, it has been observed that in a system with 4 criticality levels, to handle service level violation of any task at criticality level 4 (highest), suspension or degradation of tasks with criticality level 3 may not be acceptable.

   To improve the flexibility in the choice of tasks and their degradation, the possibility of multiple ways of degrading a task irrespective of their criticality level can be considered. A case study related to the automotive domain discussed in [25] clearly shows that a task can be degraded in multiple ways, In this study, different configurations related to accuracy, resolution of the output of a sensor task performing environment perception functionality were considered which resulted in different degrees of task degradation. Therefore, any reasonable MCS task model should give the flexibility to choose tasks that can be degraded irrespective of their criticality level. In Chapter 4 of this thesis, a new MCS task model to address these issues is discussed.

2. **Question 2: How to handle system overload due to service level violations of tasks both within a mode and during mode transitions? Can there be an MCS task model with higher level of flexibility in the choice of tasks to be degraded for which task specification and the complexity of its timing analysis are similar to the mixed-criticality sporadic task model?** For a system that can exhibit both mixed-criticality and mode change aspects, to handle system overload due to service level violation of tasks, one can consider task degradation and offsets for

tasks. For such systems, precise rules have to specified to determine the budgets that needs to be allocated for tasks belonging old-mode and/or new-mode when any service level violation of tasks occur.

A majority of existing real-time system task models that are designed to capture mode change behaviour do not consider mixed-criticality aspects of the system. The graph-based MCS task models are highly expressive and can model multiple ways of degrading a task both within a mode and during mode transitions. However, with these models, the specification of the degraded budget that needs to be allocated for a task requires higher number of vertices and edges of order $O(2^n)$. This is because, at worst-case, degraded budget for a task for each possible combination of service level violations of tasks need to be specified. Hence, timing analysis for such models tend to become harder and even becomes worse when system has to be modelled for multiple modes and mode transitions.

To handle this, a trade-off can be made between the higher complexity of the graph based models (which suffers from the state explosion problem) and the conservatism of the more standard MCS task model [26]. Expressiveness of the existing MCS task models that are based on the standard sporadic task model can be improved by considering additional parameters to capture the possibilities of multiple ways of degrading a task along with higher flexibility in the choice of degraded budget for tasks both within a mode and during mode transitions. With such a model, for a system with n tasks, the degraded budget that needs to be allocated to a task can be specified as a simple mathematical expression that can be computed at run-time with lower complexity. For example, the complexity required to find degraded budget with minimum/maximum value of a task is of order $O(n)$ only. In Chapter 5 of this thesis, a new model is proposed that can capture both mixed-criticality and mode change aspects of the real-time system.

3. **Question 3: Experiments based on synthetic tasksets are commonly used in existing MCS studies. However, such experiments will be able to show only the impact of a specific degradation strategy on the schedulability of the tasksets considered. What kind of experiments can reveal the impact on the safety and performance (quality of output) of task's functionality for a specific degradation strategy?** As the models presented in this thesis are motivated by considering requirements from automotive real-time systems, it was realized that there was a need to build a realistic automotive platform to show the impact on the safety and performance of the automotive functionalities for the degradation strategy considered by the proposed models. To cater to this need, an experimental platform with representative automotive applications exhibit-

ing different criticality levels was built. In Chapter 6 of this thesis, the details of the architecture and implementation of the testbed are presented.

1.4 Contributions and Organization

This thesis makes the following contributions.

- Chapter 2 presents concepts and terminologies related to ISO 26262, an automotive functional safety standard, that are relevant to develop any mixed criticality system (MCS). Later part of this chapter discusses the syntax and semantics of an MCS task model and a mode change model. This is followed by the discussions related to the fixed-priority based schedulability test for both MCS and mode change models.

- Chapter 3 presents literature review related to task models of MCS and mode change.

- Chapter 4 introduces the Context-Aware Mixed Criticality System (CA-MCS) model that is motivated from ISO 26262 and some existing case studies related to the automotive domain. The novelty of the CA-MCS model is threefold. First, CA-MCS gives the flexibility to choose tasks that need to be degraded irrespective of their criticality level. This allows a designer to consider the degradation of even higher criticality tasks while keeping relatively lower criticality tasks unaffected. Second, CA-MCS supports multiple ways of degrading a task's budget. This allows a designer to choose a specific way of degrading a task's budget depending on the task(s) that have overrun their budget. Third, instead of using criticality to decide tasks to be degraded, CA-MCS uses criticality only to choose the degraded budget (among multiple degraded budgets) of the task when multiple tasks overrun their budgets. For the CA-MCS model, a fixed-priority based sufficient schedulability analysis on preemptive uniprocessors is derived and proved correct.

- Chapter 5 introduces the Multi-Mode Mixed Criticality System model (MM-MCS) that combines task degradation and the notion of offsets to handle system overload due to service level violation and a mode transition. For the MM-MCS model, a fixed-priority based sufficient schedulability analysis on preemptive uniprocessors is derived and proved correct. An algorithm to compute the offset values for tasks based on their criticality value is derived. This algorithm clearly shows that criticality of tasks can play an important role not only for deciding on the task degradation but also for determining offset values. Experimental results with synthetic tasks show that task

degradation and offsets can be an effective means to handle system overload both within a mode and across mode transitions.

- In Chapter 6, the architecture and implementation of an automotive testbed is discussed. The main purpose of this testbed is to observe the effects of task degradation schemes enforced by different MCS task models on the safety and performance aspect of the vehicle which are generally not possible with synthetic taskset experiments. Experimental results show that for the selected driving scenarios, degradation strategy as per the CA-MCS model can ensure safety and can effectively isolate the effects of the degradation between applications of different criticality levels, performing different functionalities while smooth and safe mode transitions can be achieved with MM-MCS model.
- Chapter 7 concludes this thesis and discusses some possible future directions for this work.

Chapter 2

Background

In this chapter, concepts and terminologies related to ISO 26262, an automotive functional safety standard, that is relevant to develop any mixed criticality system (MCS) models are discussed. For this, a brief overview of different types of failures recognized by ISO 26262 and the hazard and risk analysis process for these failures are presented. This is followed by the discussion related to the steps involved in the criticality assignment process for an automotive case study. Later part of this chapter discusses a MCS model and a mode change model along with their timing analysis to show how schedulability test for tasks both within a mode and across mode transitions are done.

2.1 ISO 26262 Functional Safety Standard

In order to address safety aspects of automotive system development, ISO (the International Organization for Standardization) has come up with a functional safety standard called ISO 26262 [14]. Safety-related systems in automotive may include software and hardware components related to one or more electrical and/or electronic (E/E) systems. Therefore, the ISO 26262 is intended to reduce or avert the risks due to systematic and random failures of these components by providing appropriate requirements and processes in their design and development.

There are ten parts covered by ISO 26262. **Part 1** provides the vocabulary of ISO 26262 - terms, abbreviations, acronym, etc. **Part 2** focuses on management of safety requirements from a project and organisational point of view. **Part 3** deals with the 'concept phase' where concerns regarding project definition, the establishment of the safety requirements and criteria for the projects. **Part 4** considers the development at the system level such as detailed requirement analysis, system synthesis, functional

and logical evaluation, validation and verification of systems. **Part 5** and **Part 6** deal with hardware and software aspects of system design and implementation. **Part 7** details requirements for system production, operation, installation, servicing, etc. **Part 8** defines requirements for processes that support the development effort, including change management, documentation standards, etc. **Part 9** deals with requirements and guidance with respect to safety analyses; and aspects related to requirements of Automotive Safety Integrity Levels (ASIL). **Part 10** provides guidance on applying ISO 26262.

The primary focus of this chapter will be on Part 3 and Part 9. Part 3 of ISO 26262 introduces the notion of criticality for automotive functionalities and provides details on the factors that determine the ASIL. Recognising the potential benefits of MCS in automotive, Part 9 of ISO 26262 specifies the criteria for allowing software elements (or tasks) with different ASILs to coexist on the same hardware platform. Hence it is of utmost importance that MCS models for the automotive systems should be aligned with the specifications of ISO 26262. In the following, some important concepts and definitions related to ISO 26262 that are useful in the development of MCS task models for automotive systems are explained.

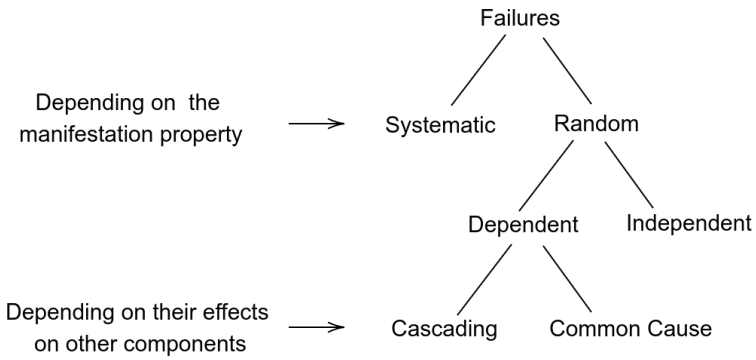

Figure 2.1: Failure Classification.

2.1.1 Faults and Failures

As per ISO 26262, termination of the ability of hardware or software component to perform a function as required can be considered as a failure. Failure can be caused due to faults. ISO 26262 defines a fault as an abnormal condition that can lead to hardware or software component to fail. Faults are classified as permanent, intermittent and transient. A permanent fault occurs and stays until removed or repaired. An intermittent fault occurs occasionally when a component is on the verge of breaking down. Systematic faults are manifested in a deterministic way that and can be prevented by applying process

or design measures. Some systematic faults like timing marginalities of a task could lead to intermittent faults. A transient fault occurs once and subsequently disappears.

Failures can be classified into different types based on their manifestation property and their effects on system behaviour. Manifestation property denotes the ability to precisely determine the cause of the failure and exact circumstances under which they can occur. Depending on their manifestation property, they are classified as systematic or random. Depending on the effect of failures on the system behaviour, they are classified as dependent and independent failures. Dependent failures are further classified as cascading and common cause failures. The details of all these failures are explained with examples below. Figure 2.1 depicts different types of failures considered in ISO 26262.

2.1.1.1 Systematic Failures

ISO 26262 defines a systematic failure as "failure related in a deterministic way to a certain cause, that can only be eliminated by a change of the design or of the manufacturing process, operational procedures, documentation or other relevant factors". Systematic failures are produced by human error during system design and development process. They can be created in any stage of the system's life cycle including specification, design, manufacture, operation and decommissioning. The manifestation of a systematic failure is deterministic in the sense that once a failure has been detected, it will always occur when the circumstances are exactly the same until it is removed. Owing to the difficulty in predicting the occurrence of circumstances causing the systematic failure, predicting the occurrence of the systematic failure and their effect on the safety of a system is also difficult. As these failures cannot be prevented with simple redundancy, to prevent them from occurring, safety standards such as ISO 26262 recommend several rigorous procedures for system and components requiring a higher level of safety integrity.

2.1.1.2 Random Failures

ISO 26262 defines random failures as "failures that can occur unpredictably during the lifetime of a hardware element, and that follows a probability distribution". Although the root cause of a random failure is very difficult to determine, the occurrence of a failure can be represented mathematically. Random failures are further classified into dependent and independent failures.

In case of dependent failures, it is not possible to express the probability of their successive or simultaneous occurrence as the product of the unconditional probabilities of each of them. Dependent

failures X and Y can be characterized when $P_{XY} \neq P_X * P_Y$ where P_{XY} denotes the probability of the simultaneous occurrence of failure X and failure Y; P_X is the probability of the occurrence of failure X and P_Y is the probability of the occurrence of failure Y.

Cascading failure is a type of dependent failure. When the failure of one component causes a set of other components to fail, it is termed as a cascading failure as shown in Figure 2.2. Biased vehicle speed information that affects the behaviour of one or more vehicle functions is an example of cascading failure. When a failure of two or more components is a result of a specific event or root cause, it is termed

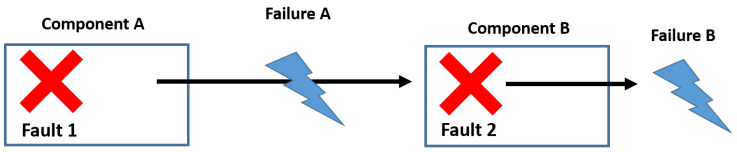

Figure 2.2: Cascading Failure.

as a common cause failure as shown in Figure 2.3. Common cause failures are dependent failures that are not cascading failures. A high-intensity electromagnetic field that causes different electronic devices to fail in a way that depends on the design and use is an example of a common cause failure. With

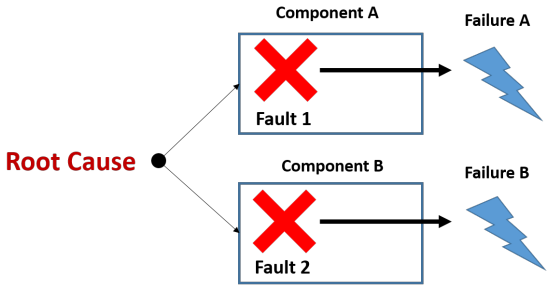

Figure 2.3: Common Cause Failure.

independent failures, it is possible to express the probability of their simultaneous occurrence as the simple product of their unconditional probabilities. Independent failures A and B can be characterized when $P_{XY} = P_X * P_Y$. Consider a software performing lead vehicle detection functionality. For this, a designer may combine radar and an additional sensor (say V2V - Vehicle to Vehicle Communication) as inputs. Since radar and V2V are based on completely different technologies, any failure in radar cannot affect V2V and vice-versa. Hence, they can be considered as independent sensors. Here, the failure of a radar sensor cannot prevent the vehicle to vehicle communication module from detecting the lead vehicle as they are completely independent in terms of sensing mechanism and hardware. Therefore, the

probability of the failure of lead vehicle detection functionality can be expressed as the simple product of the probability of the failure of radar sensor and the probability of the failure of V2V.

	Class			
	S0	**S1**	**S2**	**S3**
Description	No injuries	Light and moderate injuries	Severe and life-threatening injuries . (survival probable)	Life-threatening injuries (survival uncertain), fatal injuries
Examples	Light collision	Rear/front collision at very low-speed	Pedestrian/bicycle accident while turning (city intersection and streets)	Rear/front collision at very medium speed

Table 2.1: Classes of severity as specified in part 3 of ISO 26262.

	Class			
	C0	**C1**	**C2**	**C3**
Description	Controllable in general	Simply Controllable	Normally Controllable	Uncontrollable
Example hazard	Distracting situations like unexpected increase in radio volume	Blocked steering column when starting the vehicle	Failure of ABS during emergency braking	Failure of ABS when braking on low friction road surface while executing a turn

Table 2.2: Classes of controllability as specified in part 3 of ISO 26262.

In order to handle failures, development of safety-critical systems in the automotive industry are usually under the guidance of AUTOSAR (AUTomotive Open System ARchitecture) specification [27] and ISO 26262. AUTOSAR is a de facto standard for automotive basic software specification. The latest versions of the AUTOSAR specification contains a number of functional safety concepts to support the development of safety-related ECUs and to mitigate several failures in accordance with ISO 26262. Some of these concepts are considered to achieve the following:

1. Protection against failures due to memory access violations. The Memory Protection Unit (MPU) of the ECU is used to limit write access to the memory areas of the ECU. The software that controls the memory access and executes the context switch is part of an AUTOSAR operating system and are developed according to requirements stated in ISO 26262.

2. Protection against failures when any task violates its execution sequence and/or its timing be-

	Class				
	E0	**E1**	**E2**	**E3**	**E4**
Description	Incredible	Very low probability	Low probability	Medium probability	High probability
Example situations		Vehicle being towed	Mountain pass with unsecured steep slope	In tunnel In car wash Traffic congestion	Accelerating Braking

Table 2.3: Classes of probability of exposure as specified in part 3 of ISO 26262.

haviour. To enhance the protection against the violations of time behavior and execution sequence, AUTOSAR defines functions for monitoring the defined execution sequence and time intervals between different functions of the ASIL software. In addition to the hardware watchdog, AUTOSAR specification also recommends an additional software feature to detect any violation of requirements and to change the system to a safe state. Although this feature will be able to detect errors, it cannot prevent them. This approach of the AUTOSAR is useful to ensure that failure can be handled within allowable error reaction times.

3. Protection against failures due to end-to-end communication errors. AUTOSAR has defined algorithms for detecting error patterns in the exchange of information by an end-to-end protection. The goal of these algorithms is to detect each occurring error and if required it can change the system to a safe state. Monitoring of an end-to-end communication is performed by inserting sequence numbers and a checksum.

2.1.2 Hazards and Safety Goals

As per ISO 26262, the hazard can be defined as a potential source of harm caused by malfunctioning behaviour of software or hardware components. The combination of a hazard and an operational situation form a hazardous event. The safety goals represent the required safety behaviour that has to be incorporated into the system to avoid unreasonable risks. These goals are derived based on the systematic evaluation of possible hazardous events. For each safety goal, the corresponding safety measures are specified as functional and technical safety requirements. The specification of implementation-independent safety behaviour is termed as functional safety requirements. The specification of the details of the software and/or hardware implementation of associated functional safety is termed as technical safety requirements.

Not all hazards associated with the system have the same impact level on its safety. Hence to clas-

sify the impact level of each possible hazard, ISO 26262 defines an Automotive Safety Integrity Level (ASIL), a risk classification scheme, for each hazard. This helps to define the safety requirements necessary in accordance with the ISO 26262 standard. ASIL is determined by performing a risk analysis by referring to the classes of severity, exposure and controllability factors of the hazard for a vehicle operating scenario. Each hazard is then associated with a safety goal. The functional and technical safety requirements are specified for each safety goal which is later assigned to one or more software and/or hardware components. ASIL for safety goal, safety requirements and the hardware/software components are inherited from ASIL of the hazard. ASIL assignment process is depicted in Figure 2.4.

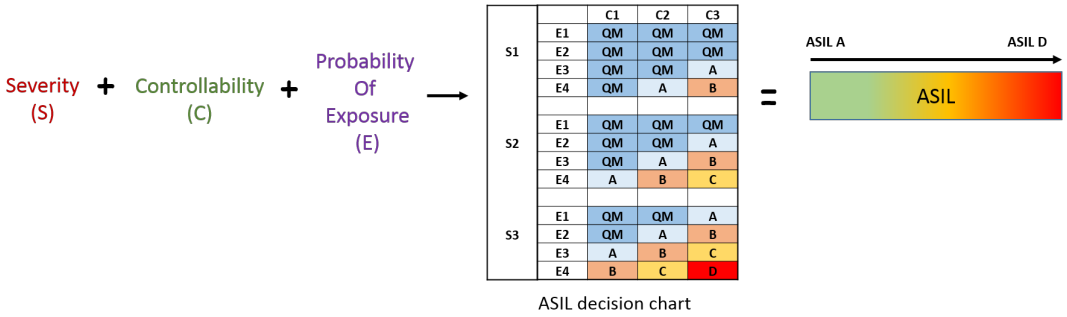

Figure 2.4: ASIL Assignment Process.

Identifier	Operational Situation		Consequence	C	S	E	ASIL
HE1	Highway, speed >30 km/h, collision not imminent	Good conditions	Crash into vehicle in adjacent lane or drive off the road.	C3	S3	E4	D
HE2	Highway, speed <30 km/h	Good conditions	Crash into vehicle in adjacent lane or drive off the road.	C2	S3	E2	A

Table 2.4: Hazardous Events for the Hazard H1.

ASIL of a component can range from A to D. ASIL D being the highest degree of automotive hazard, the components of this level are required to meet highest degree of integrity and safety assurance. Parts 6 and 7 of ISO 26262 specify the safety requirements for ASILs at the hardware and software level of the system. The severity, controllability and the exposure of potential hazard associated with the vehicle and human driver are estimated based for each hazardous event based on a defined rationale and

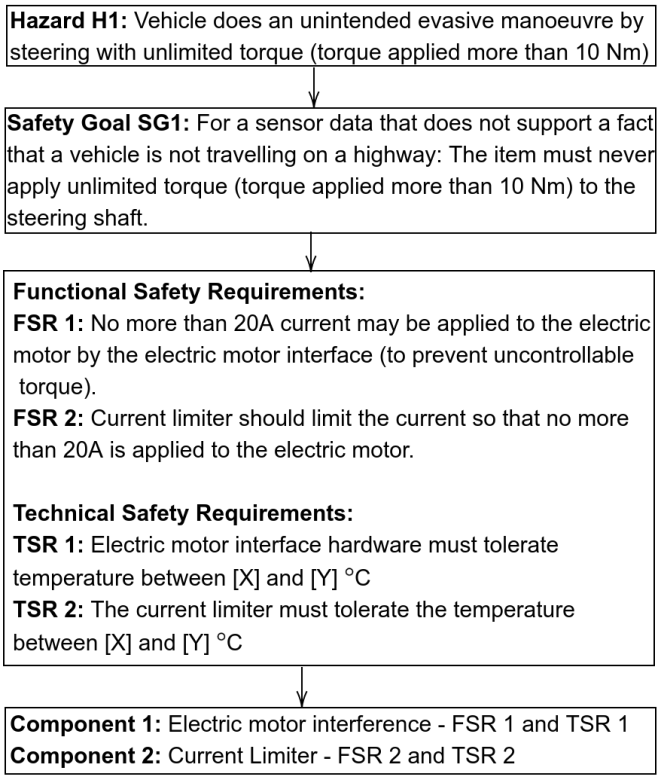

Figure 2.5: ASIL assignment for the components of steer control system.

assigned with one of the classes described in Tables 2.1, 2.2 and 2.3.

2.1.3 ASIL Assignment Process - A Case Study

The process of ASIL assignment is explained by considering steer control (SC) as an example. The hazard and its safety goal along with functional and technical safety requirements for hardware components of SC are shown in Figure 2.5. The hazard and risk analysis leads to the detection of a hazard H1 which threatens the functional safety of SC. For example, H1 here is considered as an unintended manoeuvre by steering with unlimited torque (torque applied more than 10 Nm). Hazardous events HE1 and HE2 of H1 as shown in Table 2.5 corresponds to vehicle velocities greater and lesser than 30 Km/hr respectively. Based on Severity (S), Controllability (C) and Exposure (E) values of HE1 and HE2, they are assigned with ASIL D and ASIL A values. A safety goal SG1 for the hazard is derived considering the hazardous events. SG1 considered here is that when the vehicle is not travelling on the highway, any item that issues control to the steering shaft must never apply unlimited torque (torque

Attribute	Identifier	ASIL
Hazard	H1	D
Hazardous Events	HE1	D
	HE2	A
Safety Goal	SG1	D
Functional Safety Requirements	FSR 1	C(D)
	FSR 2	A(D)
Technical Safety Requirements	TSR 1	C
	TSR 2	A
Allocation of FSR to Components	Electric Motor Interface for FSR 1	C
	Current Limiter for FSR 2	A

Table 2.5: Attributes and ASIL.

applied more than 10Nm). The specifications of Functional Safety Requirements (FSR) and Technical Safety Requirements (TSR) for the SG1 are also derived. The ASIL of SG1, FSR and TSR are inherited from ASIL of the H1 which is ASIL D as shown in Table 2.5. Finally, the FSR and TSR are assigned to hardware and software components of Electric Motor Interface and Current Limiter.

2.1.4 ASIL Decomposition

ASIL decomposition is a method of tailoring the ASIL to software and/or hardware components which implement the safety requirements. This method allows the ASIL of a safety requirement to be apportioned between several components that complies with the same safety requirement addressing the same safety goal. ASIL decomposition provides the following opportunities

- to implement safety requirements by decomposing them into redundant and independent components, and
- to assign a potentially lower ASIL to these independent components

The term independence refers to the absence of propagation of failures from one component to the other implementing the same safety requirement. If the components are not sufficiently independent that is if the failure of one component can affect the other components, then the redundant requirements and architectural components inherit initial ASIL.

ASIL decomposition can be explained considering Adaptive Cruise Control (ACC) [28] as an example application. One of the safety goals of ACC is to detect lead vehicle which can be implemented

either by radar task or vehicle to vehicle (V2V) communication task. The failure to detect the lead vehicle can be considered as a hazard. The hazardous event refers to failure in detecting the lead vehicle while the vehicle is on highway or hill road or during lane change manoeuvre. For example, radar can fail to detect the lead vehicle in curve roads due to the constraints imposed by its geometry and sensing mechanism. As V2V communication relies on the data from other V2V modules, the host vehicle cannot always rely upon its V2V communication to arrive at decisions related to braking and throttle. Since severity, controllability and exposure values to this hazard in highway or city roads are very high, the safety goal of this hazard requires ASIL D assurance. But due to inherent incapability of radar sensor and V2V communication, they can only meet the safety requirements corresponding to ASIL C and ASIL A respectively. However, as per ISO 26262, since they are completely independent in terms of their physical sensors and software, they can together (ASIL C + ASIL A) achieve the required ASIL D requirement. In the example considered in the previous section, ASIL decomposition concept is applied among FSRs by assigning ASIL C for FSR1 and ASIL A for FSR2 to achieve ASIL D requirement of the safety goal. In Table 2.5, the ASIL values in the brackets denote the ASIL of the safety goal.

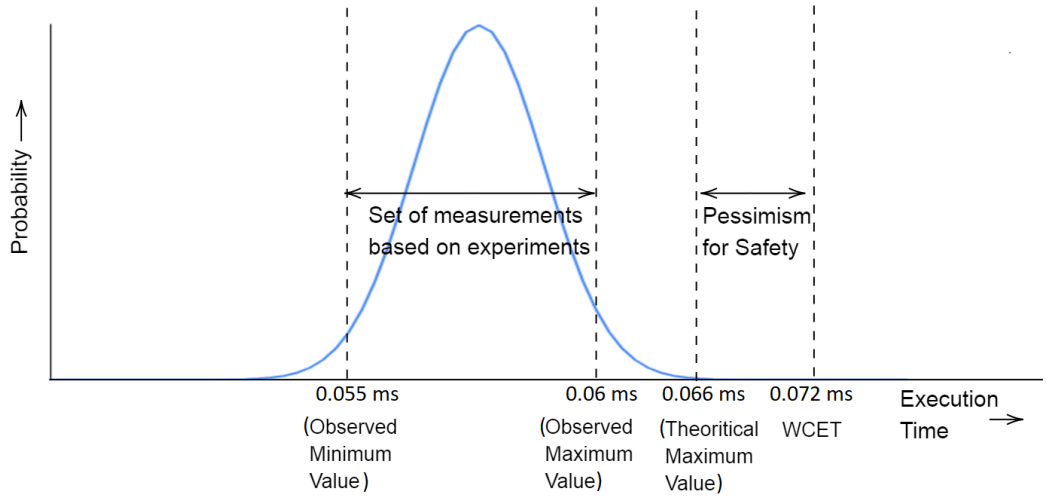

Figure 2.6: Execution time profile of a task.

2.1.5 Freedom from Interference

ISO 26262 uses the term interference to denote the presence of cascading failures from a component with no ASIL assigned, or a lower ASIL assigned, to a component with a higher ASIL assigned leading to the violation of a safety requirement of the component. ISO 26262 requires a designer to adhere to the following conditions to consider the coexistence of software components that have different ASILs assigned or the coexistence of software components that have no ASIL assigned with safety-related ones

- **Condition 1:** - If the evidence for freedom from interference is not feasible then the only possibility is to raise the ASIL of the software component to the highest ASIL among all coexisting components. However, this can lead to increased certification cost.
- **Condition 2:** - If the evidence for freedom from interference is feasible then these components can be allowed to coexist.

Therefore, by providing mechanisms to achieve freedom from interference, a designer can avoid raising the ASIL for some of them to the highest ASIL among the coexisting components. For example, interference due to shared memory can be prevented by having separate memory or storage location for higher ASIL components and constraining the memory operations *i.e.* ASIL D can read from a ASIL C element, but not the other way around. Similarly, timing interference can be caused due to blocking of execution, deadlocks, livelocks, incorrect synchronization between elements (*e.g.* bad clock times), incorrect allocation of budget and incorrect execution of the sequence. ISO 26262 rationale for the freedom from interference is supported by analyses of dependent failures focused on cascading failures.

According to the definitions given in part 1 of ISO 26262, independence is threatened by common cause failures and cascading failures while freedom from interference is only threatened by cascading failures. Therefore, an important challenge in achieving freedom from interference in the design of a real-time system is to carefully handle systematic faults like timing marginalities of a task. These faults if not properly handled can easily manifest as cascading failures. For example, when a task executes more than its allocated budget, it can easily starve the execution of other tasks thereby preventing other tasks to achieve their intended safety goals. Therefore a significant number of studies have been devoted to evaluating the timing variabilities of the task's execution behavior, especially in developing novel techniques to determine their Worst-Case Execution Time (WCET). The difficulty in determining exact WCET value for a task can be attributed to the hardware complexity (*e.g.*, current multi-core architectures, shared resources such as memory, cache memories and pipelines [29] that include speculative mechanisms) and the software complexity (*e.g.*, many functionalities, interoperability, dependencies).

The WCET parameter used for timing verification of tasks is usually a conservative upper bound that exceeds the true WCET as shown in Figure 2.6.

The real-time task models for Mixed Criticality Systems (MCS), first proposed by Vestal [15] and subsequently extended by others, are motivated from the need to achieve the freedom from interference and at the same time efficiently utilize the resources when tasks with different criticality level share common platform (hardware and software). To satisfy this need, several studies have dealt with theoretical problems related to modelling and verification related to the design and implementation of the hardware and run-time software for MCS.

2.2 Task Models for Mode Change and Mixed Criticality Systems

A vast majority of existing MCS task models is a simple extension of the classic sporadic task model [21] to the mixed-criticality setting while few others such as [30] and [31] are an extension of graph-based models [32]. In order to handle system overload conditions, existing MCS task models consider two ways of executing low critical tasks. A majority of MCS models suspend low critical tasks while some of them allow low critical tasks to continue their execution either with reduced budgets or with increased time period. As models proposed in this thesis are based on sporadic MCS model, syntax and semantics for the MCS model proposed in [19] along with its fixed-priority based schedulability tests are discussed. Additionally, an overview of the mixed-criticality elastic task model [1] with an example taskset is also discussed. In this model execution support for low criticality tasks with increased time period is considered. Finally, syntax and semantics for the interference constraint graph (ICG) and a basic mode change model [33] are also discussed.

2.2.1 The Burns and Baruah Model

In the Burns and Baruah model proposed in [23], each task, τ_i, is specified by a tuple (T_i, D_i, $\vec{C_i}$, $\mathcal{L}_i$). T_i denotes the time period, D_i denotes deadline, $\mathcal{L}_i$ denotes criticality and $\vec{C_i}$ denotes a vector of execution time estimates. The criticality of tasks can be either HI or LO. This model assumes that each task with criticality HI (high criticality task) can have a WCET estimated at HI criticality (denoted as $C(HI)$) and an additional execution time estimated at low criticality (denoted as $C(LO)$). Each task with LO criticality (low criticality task) is assigned only with its WCET estimated at LO criticality. If $\mathcal{L}_i = HI$, then $\vec{C_i} = (C(LO), C(HI))$. Otherwise, if $\mathcal{L}_i = LO$, $\vec{C_i} = (C(LO))$ with the constraint: $C(HI) \geq C(LO)$.

2.2.1.1 Adaptive Mixed Criticality Scheduling Policy

In the following, a detailed description of the Adaptive Mixed Criticality (AMC) scheduling policy proposed in [23] for the uniprocessor systems and the AMC based schedulability test for a dual-criticality system are presented. The AMC scheduling policy makes use of the capability of the platform to monitor the amount of time for which individual jobs have executed. The algorithm is provided with a set of mixed criticality sporadic tasks where each task is preassigned with an unique priority.

Definition 5 (AMC scheduling policy) *According to the AMC scheduling policy, jobs are dispatched for execution at run-time according to the following rules:*

R1: There is a execution phase indicator P, initialized to LO-crit.

R2: The system starts its execution in the LO-crit phase where all tasks (with criticality LO or HI) are assigned with their budget equal to their $C(LO)$ values.

R3: While ($P \equiv$ LO-crit), at each time instant the waiting job released by the highest priority task is selected for execution.

R4: If the currently-executing job executes for its $C(LO)$ budget without signalling completion, then $P \leftarrow$ HI-crit.

R5: Once ($P \equiv$ HI-crit), low criticality jobs will be suspended. Henceforth, therefore, at each time instant only the waiting job released by the highest priority task with the high criticality is selected for execution.

From Definition 5, it is clear that in AMC, all low criticality tasks are descheduled if any job of a low criticality task (τ_l) executes for more than $C_l(LO)$. If a job of a high criticality task (τ_h) executes for more than $C_h(LO)$ (but no greater than $C_h(HI)$ all low criticality tasks stop executing. In both of these scenarios, all high criticality tasks continue to meet their deadlines. A standard assumption made in [23] and in all the MCS task models is that when any task τ_i executes till its $C_i(HI)$ but does not signal completion, it is considered as an error.

2.2.1.2 Schedulability Test for AMC

A sufficient schedulability test considered for AMC scheduling policy consists of three steps:

1. Verifying the schedulability of the LO-crit phase,

2. Verifying the schedulability of the HI-crit phase,

3. Verifying the schedulability of the phase change itself.

For steps 1 and 2, the basic response time analysis for fixed priority systems considered in [34] can be applied. The third step is necessary as the schedulability of the phase change cannot be deducted based on the individual schedulability of the LO-crit phase and the HI-crit phase [35].

To say that a task with an implicit deadline (deadline equal to or lower than its time period) is schedulable, its worst-case response time should be equal or lower than its deadline. For this, the equation to compute worst-case response time value for a task considered in [34] can be used. Consider a task τ_i belonging to a taskset Γ such that deadlines of tasks are at most equal to periods *i.e.* $\forall\ i$, $D_i \leq T_i$. Let WR_i denote the worst-case response time value of τ_i. Let tasks are assigned with arbitrary but fixed and unique priority value. The response time for τ_i depends on the release pattern (phasing) of tasks with priority higher than τ_i. Therefore, one has to know the the release pattern of higher priority tasks that can lead to WR_i. The instant where such a release pattern occur is called as critical instant of τ_i. According to [34], a critical instant for any task τ_i occurs when τ_i is released simultaneously with all tasks with priority higher than τ_i and hence the highest amount of pre-emption of a task is found after a simultaneous release of tasks with priority higher than τ_i. For this critical instant, the WR_i is the solution to the following iterative equation:

$$x = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{x}{T_j} \right\rceil C_j \tag{2.1}$$

where $hp(i)$ is the set of tasks with priority higher than that of task τ_i. C_i and C_j denote the computational requirement of τ_i and τ_j respectively. C_i is the smallest positive solution for Equation 2.1. $\left\lceil \frac{x}{T_j} \right\rceil$ denotes the maximum number of pre-emptions of task τ_j in an interval $[0, x)$ by task τ_j whereas $\left\lceil \frac{x}{T_j} \right\rceil C_j$ denotes the maximal preemption time of task τ_j in an interval $[0, x)$ by task τ_j. Intuitively, the LHS of Equation 2.1 denotes the amount of time available (or provided) in $[0, x)$; and the RHS denotes the maximum amount of time requested in $[0, x)$ by τ_i and by tasks with priority higher than τ_i. The iteration is stopped when either the value of x found is same for two successive iterations; or when the value of x exceeds the value of deadline D_i (hence, not schedulable). τ_1 has the highest priority whereas τ_2 has the lowest. In the following, iterations required to calculate WR_3 *i.e.* the worst-case response time for task τ_3 considered in Table 2.6 is presented.

Iteration 1: $WR_3(0)$ = C_3 + $\sum_{j<3} C_j$ = 5 + 3 + 11 = 19.

Task	Time Period (T_i)	Deadline (D_i)	WCET	Utilisation
τ_1	10	10	3	0.3
τ_2	19	19	11	0.58
τ_3	56	56	5	0.09

Table 2.6: An example to illustrate the Equation 2.1.

Iteration 2: $WR_3(1) = C_3 + \sum_{j<3} \left\lceil \frac{WR_3(0)}{T_j} \right\rceil \ C_j = 5 + \lceil \frac{19}{10} \rceil \ 3 + \lceil \frac{19}{19} \rceil \ 11 = 5 + 2.3 + 1.11 = 22.$

Iteration 3: $WR_3(2) = 5 + \lceil \frac{22}{10} \rceil \ 3 + \lceil \frac{22}{19} \rceil \ 11 = 5 + 3.3 + 2.11 = 36.$

Iteration 4: $WR_3(3) = 5 + \lceil \frac{36}{10} \rceil \ 3 + \lceil \frac{36}{19} \rceil \ 11 = 5 + 4.3 + 2.11 = 39.$

Iteration 5: $WR_3(4) = 5 + \lceil \frac{39}{10} \rceil \ 3 + \lceil \frac{39}{19} \rceil \ 11 = 5 + 4.3 + 3.11 = 50.$

Iteration 6: $WR_3(5) = 5 + \lceil \frac{50}{10} \rceil \ 3 + \lceil \frac{50}{19} \rceil \ 11 = 5 + 5.3 + 3.11 = 53.$

Iteration 7: $WR_3(6) = 5 + \lceil \frac{53}{10} \rceil \ 3 + \lceil \frac{53}{19} \rceil \ 11 = 5 + 6.3 + 3.11 = 56.$

Iteration 8: $WR_3(7) = 5 + \lceil \frac{56}{10} \rceil \ 3 + \lceil \frac{56}{19} \rceil \ 11 = 5 + 6.3 + 3.11 = 56.$

It is to be noted that because $WR_3(6) = WR_3(7) = 56 \leq D_3 = T_3$, $WR_3 = 56$.

With the AMC scheduling policy, the system remains in LO-crit phase when no tasks execute beyond their $C(LO)$ budgets. Hence, to determine schedulability of the LO-crit phase, only the $C(LO)$ can be considered as execution requirement (or budgets) for all tasks. The value of the worst-case response time for τ_i in the LO-crit phase is denoted as R_i^{LO} and can be computed by substituting $C(LO)$ as budgets for tasks in the equation 2.1. Hence, R_i^{LO} is given by the solution to the following iterative equation,

$$R_i^{LO} = C_i(LO) + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i^{LO}}{T_j} \right\rceil \ C_j(LO) \tag{2.2}$$

where $hp(i)$ is the set of tasks with priority higher than that of task τ_i.

In HI-crit phase only tasks with HI criticality task are allowed to execute upto their $C(HI)$ budgets. Hence, to determine schedulability of the HI-crit phase, only the $C(HI)$ budget can be considered as execution requirement for all tasks. The value of the worst-case response time for τ_i in the HI-crit phase is denoted as R_i^{HI} and can be computed by substituting $C(HI)$ as budgets for tasks in the equation 2.1.

Hence, R_i^{HI} is given by the solution to the following iterative equation,

$$R_i^{HI} = C_i(HI) + \sum_{\forall j \in hpH(i)} \left\lceil \frac{R_i^{HI}}{T_j} \right\rceil C_j(HI) \tag{2.3}$$

where $hpH(i)$ is the set of high critical tasks with priority higher than that of task τ_i.

When any task τ_j executes for more than $C_j(LO)$ at any arbitrary time s, then phase change is triggered. Therefore, in step 3, to verify schedulability of the phase change, all possible time instants where phase change can be triggered should be considered. If a phase change triggered by τ_j has any impact on task τ_i, then $s < R_i^{LO}$, and the priority of τ_j must be greater than that of τ_i. Otherwise, τ_i will have completed before τ_j. Therefore, the value of s can be some time between 0 and the R_i^{LO}.

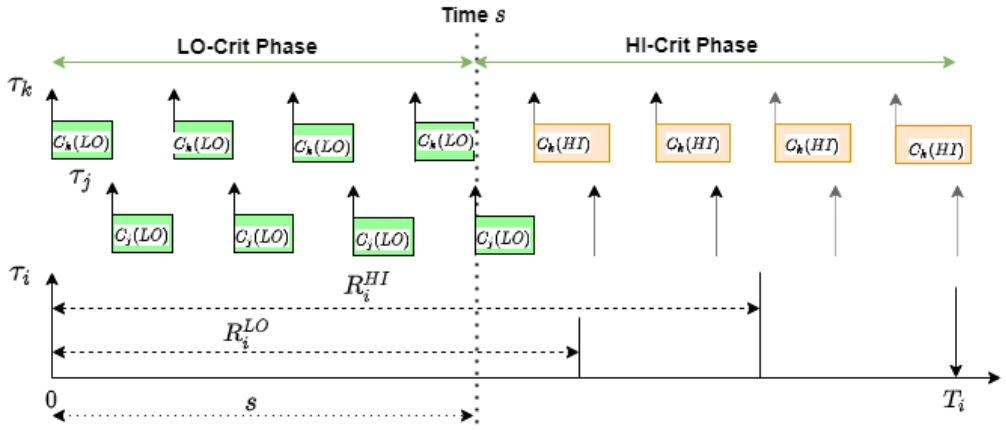

Figure 2.7: Illustration of the Adaptive Mixed Criticality scheduling policy.

Let us consider a task τ_i with HI criticality. Let τ_j and τ_k be tasks with priority higher than τ_i. Let the criticality of τ_j and τ_k be LO and HI respectively. Figure 2.7 illustrates the interference for τ_i from τ_j and τ_k. Since τ_k is a high criticality task, it can interfere τ_i with its $C_k(LO)$ (low criticality budget) in the LO-crit phase and with $C_k(HI)$ (high criticality budget) in HI-crit phase. Since τ_j is a low criticality tasks, with AMC scheduling policy, τ_j can interfere τ_i only in LO-crit phase but gets suspended in HI-crit phase. Therefore, two kinds of sources of interference from τ_k must be accounted for the selected value of s. For a given s, the interference from low criticality tasks in a time interval of length t is denoted as $I_L(t, s)$, whereas the interference from high criticality tasks is denoted as $I_H(t, s)$.

$I_L(t,s)$ is calculated using the following expression,

$$I_L(t,s) = \begin{cases} \sum\limits_{\forall j \in hpL(i)} \left\lceil \frac{t}{T_j} \right\rceil \; C_j(LO), & t < s, \\ \sum\limits_{\forall j \in hpL(i)} \left(\left\lfloor \frac{s}{T_j} \right\rfloor + 1 \right) C_j(LO), & t \geq s \end{cases} \tag{2.4}$$

where t denotes the response time of the task. The value of t is updated in each iteration of the algorithm. $hpL(i)$ is the set of low critical tasks with priority higher than that of task τ_i. If the value of the time interval t is such that $t < s$, then the expression $\left\lceil \frac{t}{T_j} \right\rceil$ upper bounds the number jobs of low criticality tasks that can interfere τ_i with low criticality budgets. However, if the value of the time interval t is such that $t \geq s$, then the expression $\left\lfloor \frac{s}{T_j} \right\rfloor + 1$ gives the upper bound number of jobs interfering with low criticality budgets. Note that this expression accounts even those releases occurring at s.

To compute $I_H(t,s)$, a function M is first defined to compute the upper bound number of jobs from any higher priority task τ_j with high criticality as the following,

$$M(j,s,t) = min(\left\lceil \frac{t-s}{T_j} \right\rceil + 1, \left\lceil \frac{t}{T_j} \right\rceil) \tag{2.5}$$

In the above expression, for lower values of s, $\left\lceil \frac{t-s}{T_j} \right\rceil + 1$ can lead to higher pessimism than $\left\lceil \frac{t}{T_j} \right\rceil$. Therefore, only the minimum value among these two terms is considered. Given the value returned by M, the value of $I_H(t,s)$ can be given by the following equation:

$$I_H(t,s) = \sum_{\forall j \in hpH(i)} \left\{ M(j,s,t)\, C_j(LO) + \left(\left\lceil \frac{t}{T_j} \right\rceil - M(j,s,t) \right) C_j(HI) \right\} \tag{2.6}$$

where $hpH(i)$ is the set of high critical tasks with priority higher than, or equal to, that of task τ_i. Finally, the worst-case response time for τ_i for a given s is given by the solution to the following recursive equation:

$$R_i(s) = C_i(HI) + I_L(R_i(s), s) + I_H(R_i(s), s) \tag{2.7}$$

The worst-case response time for τ_i is given by the following expression,

$$R_i = \max(R_i(s)), \forall s \tag{2.8}$$

2.2.2 The Interference Constraint Graph (ICG) model

The Interference Constraint Graph (ICG) is a graph based MCS task model presented in [30] that gives higher level of flexibility for a designer to specify tasks to be affected when a certain task executes above a certain execution time threshold. With this model, there is no need to suspend all tasks of a particular criticality level as it is possible to ensure the schedulability of the high criticality tasks even when certain low criticality tasks continue executing. The allowed interferences between tasks are specified as an Interference Constraint Graph (ICG). In this graph, each task maps to a node, and an edge between nodes quantifies the interference between a pair of tasks. Therefore, the ICG model can be considered more expressive than majority of the existing sporadic based mixed criticality models.

2.2.2.1 System model

Consider a set of n sporadic tasks denoted as $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$ executed on a uniprocessor. Each task τ_i is characterized by a minimum inter-arrival time T_i between any two successive jobs and a relative deadline D_i where $D_i \leq T_i$. An important assumption considered by the ICG model is that the actual execution time of a job of a task is not known until runtime *i.e.* until when the job signals that it has finished executing. Therefore, a scenario $s(t)$ is defined as the tuple $(c_1(t), c_2(t), ..., c_n(t))$, where $c_i(t)$ is the maximum of the run times of all jobs of task τ_i till time t. Each scenario is considered to have at least one trace associated with it. A trace $tr(t)$ for a scenario $s(t)$ is characterized by the deadlines, arrival and finishing times of jobs till time t. As finishing time of a job depends on the scheduling strategy, $tr(t)$ also depends on the chosen scheduling strategy. In general, a taskset and scheduling policy may exhibit at runtime many scenarios and associated traces. In the following, a formal definition of an ICG established in [30] is presented.

Definition 6 *An Interference Constraint Graph (ICG) is a directed graph $G(V, E, \sigma)$, where V is a set of vertices which corresponds to the set of tasks,* i.e. $V \stackrel{def}{=} \Gamma$*, E is a set of directed edges with $e_{i,j} \in E$ being an edge from τ_i to τ_j,* i.e. $e_{i,j} = (\tau_i, \tau_j)$*, and $\sigma(e)$ is a function $\sigma : E \longrightarrow \mathbb{R}^+$ defined for all edges $e \in E$.*

If there exists an edge $e_{i,j} = (\tau_i, \tau_j)$, then we say that task τ_i can interfere with task τ_j , meaning that if during runtime the execution time of a job of task τ_i exceeds $\sigma(e_{i,j})$ at time $t \geq 0$, then after time t, jobs of task τ_j do not need to be executed anymore. For all $e_{i,j} \in E$, we have $\sigma(\tau_i, \tau_j) \leq D_i$. In the following, a formal definition for ICG schedulability established in [30] is presented.

Definition 7 ***ICG-Schedulability:*** *Given a taskset Γ and a corresponding ICG G, the taskset is ICG schedulable with a particular scheduling strategy if in any scenario $s(t)$ and any corresponding trace $tr(t)$ that can be produced by the taskset and the strategy, jobs of any task τ_j with absolute deadlines smaller or equal to t must meet their deadlines if for no task τ_i with $e_{i,j} \in E$. we have that:*

$$c_i(t) > \sigma(\tau_i, \tau_j)$$

ICG-Schedulability requires that deadlines of a task τ_j to be met in any scenario $s(t)$, only if each interfering task τ_i does not execute for more than $\sigma(\tau_i, \tau_j)$ in this scenario. If a task τ_j does not have any interfering tasks, *i.e.* there are no edges $e_{i,j}$, then jobs of task τ_j should meet their deadlines in any scenario $s(t)$. In the following, an example to illustrate the construction of ICG for the mixed-criticality system model presented in subsection 2.2.1 is discussed.

2.2.2.2 ICG specification of a mixed criticality taskset

Consider the Burns and Baruah model discussed in the subsection 2.2.1 where each task, τ_i, is specified by a tuple (T_i, D_i, $\vec{C_i}$, $\mathcal{L}_i$) where $\vec{C_i}$ is a vector, the m-th coordinate of which denotes the execution time estimate of τ_i at criticality level m where $C_i(\mathcal{L}_i)$ denotes the WCET of τ_i. For a scenario $s(t)$, its criticality level is determined as the smallest value l that satisfies the inequality $c_i(t) \leq C_i(l)$, $\forall i, 1 \leq i \leq n$. It is assumed that l always exists, otherwise the scenario is considered erroneous.

Scheduling conditions for a mixed-criticality taskset under ICG: For a given runtime scenario $s(t)$ with criticality level l and corresponding trace $tr(t)$, a mixed-criticality taskset is considered schedulable with a particular scheduling strategy if the strategy will give sufficient execution to each job that belongs to a task with $\mathcal{L}_i \geq l$ and has a deadline smaller or equal to t, such that the job can signal finishing before its deadline.

A taskset which is subject to the above system model and schedulability conditions can be specified with an ICG graph $G = (V, E, \sigma)$ as follows:

1. $V = \Gamma$
2. For every task, τ_i, add edge $e_{i,i} = (\tau_i, \tau_i)$, with $\sigma(e_{i,i}) = C_i(\mathcal{L}_i)$.
3. For every pair of tasks τ_i and τ_j with $\mathcal{L}_i > \mathcal{L}_j$, add an edge $e_{i,j} = (\tau_i, \tau_j)$ with $\sigma(e_{i,j}) = C_i(\mathcal{L}_j)$.

In the following, an example taskset is considered to illustrate the results for above steps and to

show that it is possible to capture commonly used mixed-criticality system model using an ICG. The parameters of tasks are shown in Table 2.7.

Task (τ_i)	**Time period** (T_i)	**Deadline** (D_i)	**Criticality** ($\mathcal{L}_i$)	$C_i(3)$	$C_i(2)$	$C_i(1)$
τ_1	16	16	3	5	3	1
τ_2	24	24	2	-	5	3
τ_3	13	13	1	-	-	2

Table 2.7: Task parameters for the example.

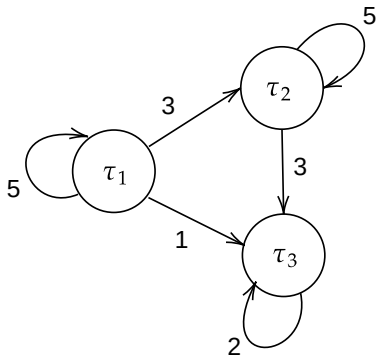

Figure 2.8: ICG representation for the example considered in Table 2.7.

In the above example, the criticality of τ_1 is higher than τ_2 and τ_3. As per the conventional mixed-criticality model [23], if the priority of τ_3 is lower than τ_2 and τ_1 then τ_3 is guaranteed to meet its deadline only if interfering tasks τ_1 and τ_2 do not execute beyond $C_1(1)$ and $C_2(1)$ respectively. In Figure 2.8, the value specified in the edge $e_{1,3}$ as 1 unit means that τ_1 is allowed to interfere τ_3 only upto 1 units (corresponding to $C_3(1)$) beyond which τ_3 is not allowed to execute. Similarly, the value specified in the edge $e_{2,3}$ as 3 units means that τ_2 is allowed to interfere τ_3 only upto 3 units (corresponding to $C_2(1)$) beyond which τ_3 is not allowed to execute. Also, the criticality of τ_1 is higher than τ_2. Hence, the value specified in the edge $e_{1,2}$ as 3 units means that τ_1 is allowed to interfere τ_2 only upto 3 units (corresponding to $C_1(2)$) beyond which τ_2 is not allowed to execute.

2.2.2.3 Schedulability test for ICG

In the following, an ICG-Schedulability test under fixed-priority preemptive scheduling is presented. Given a taskset Γ and a corresponding ICG $G = (V, E, \sigma)$, Γ is ICG-Schedulable if $\forall \tau_i \in \Gamma$, we have that $R_i \leq D_i$, where R_i is a fixed-point solution of the following recurrence equation:

$$R_i = \sigma(\tau_j, \tau_j) + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil \min\{\sigma(\tau_j, \tau_j), \sigma(\tau_j, \tau_i)\} \tag{2.9}$$

where $hp(i)$ denotes a set of tasks with priority higher than τ_i. According to the ICG schedulability condition presented in Definition 7, a task τ_i can be guaranteed to meet its deadline if each interfering task τ_j does not execute for more than $\sigma(\tau_j, \tau_i)$. We can consider two cases:

1. If a task τ_j with priority higher than τ_i is allowed to interfere τ_i then it is enough to assume that τ_j can execute for not more than $\min\{\sigma(\tau_j, \tau_j), \sigma(\tau_j, \tau_i)\}$ because:
   - If $\sigma(\tau_j, \tau_j) \geq \sigma(\tau_j, \tau_i)$, τ_i is dropped if the runtime of τ_j exceeds $\sigma(\tau_j, \tau_i)$.
   - If $\sigma(\tau_j, \tau_j) < \sigma(\tau_j, \tau_i)$, it is not possible for τ_j to trigger the dropping of τ_i, since it is not allowed to execute for more than $\sigma(\tau_j, \tau_j)$.
2. If a task τ_j with criticality higher than τ_i is not allowed to interfere τ_i, then its worst-case execution time is assumed to be $\sigma(\tau_j, \tau_j)$ when determining the schedulability of τ_i. Both cases can be compactly represented as $\min\{\sigma(\tau_j, \tau_j), \sigma(\tau_j, \tau_i)\}$.

2.2.2.4 An example

Consider a taskset consisting of four tasks whose parameters are given in Table 2.8 with $D_i = T_i$, $\forall i$. The corresponding ICG is shown in Figure 2.9.

Task (τ_i)	**Time period** (T_i)	**Deadline** (D_i)
τ_1	16	16
τ_2	24	24
τ_3	13	13
τ_4	6	6

Table 2.8: Taskset parameters

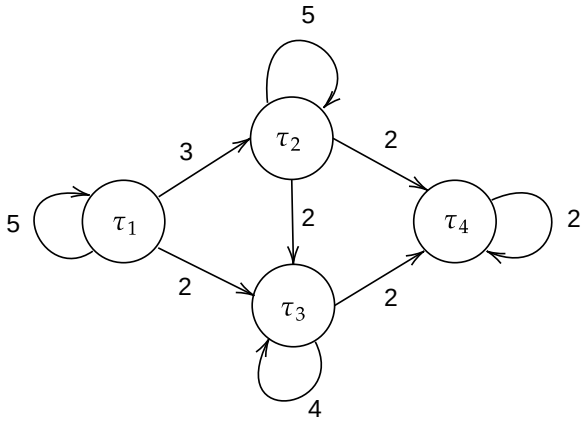

Figure 2.9: ICG for the taskset in Table 2.8.

In Figure 2.9, task τ_1 is not allowed to interfere directly with τ_4. Hence, when considering the schedulability of τ_4, τ_1 must be assumed to take 5 units of execution time, if the schedulability of τ_4 depends on the execution time of τ_1 (*e.g.* τ_1 is assigned a higher priority than τ_4). In contrast, τ_1 is allowed to interfere with τ_2, hence when testing the schedulability of τ_2, τ_1 only needs to be assumed executing for 3 time units.

Let us consider a particular priority ordering where τ_4 has the highest priority: $\tau_4 > \tau_1 > \tau_2 > \tau_3$. For the schedulability test of τ_3, the fixed-point of the following recursive equation considered in Equation 2.9 gives the worst-case response time of τ_3:

$$R_3 = 4 + \left\lceil \frac{R_3}{16} \right\rceil 2 + \left\lceil \frac{R_3}{24} \right\rceil 2 + \left\lceil \frac{R_3}{6} \right\rceil 2$$

Here, the execution times of τ_1 and τ_2 are assumed to be 2 units each, since τ_3 is dropped and therefore does not need to be guaranteed when τ_1 and τ_2 execute more than 2 units. Task τ_4 is assumed to execute its worst-case execution time of 2 units since it is not allowed to interfere with τ_3. Equation 2.9 has a fixed-point of 12, hence τ_3 is schedulable on the lowest priority level. In fact, this taskset is ICG-Schedulable with the priority assignment given above.

2.2.3 Elastic Task Models

In [36], a new task model is proposed to model periodic tasks that can intentionally change their execution rate to achieve different quality of service. For a task τ_i, its execution rate is the defined as the ratio of its WCET and its time period *i.e.* $U_i = C_i/T_i$. C_i denotes the WCET and T_i denotes the time period of τ_i. When any task requires higher execution rate, other tasks can automatically adapt their time periods to keep the system under-loaded. Therefore, an elastic task τ_i is represented by five parameters as the following:

$$\tau_i = (C_i, T_{i,0}, T_{i,min}, T_{i,max}, e_i):$$

where C_i is the WCET, $T_{i,0}$ is the nominal period, $T_{i,min}$ is the minimum period, $T_{i,max}$ is the maximum period, and $e_i \geq 0$ is an elastic coefficient. The value of e_i specifies the flexibility of the task to vary its time period during overload conditions. The greater the value of e_i, the more elastic the task. When $e_i = 0$, it means that the time period of τ_i cannot be varied by the system during load reconfiguration. When $e_i > 0$, it means that the time period of τ_i can be varied depending on its needs within its specified range. However, any such variation is subject to an elastic guarantee and is accepted only if there exists a feasible schedule in which all other periods are within their range.

In [1], an Elastic Mixed Criticality (E-MC) task model is proposed by adopting the idea of variable periods in elastic scheduling [36]. To handle system overload due to service level violation of tasks, low critical tasks are assigned with their largest possible time period corresponding to their minimum service requirement. Additionally, the model also considers the possibility of releasing low criticality tasks early and more frequently at run-time to improve their service levels without sacrificing the timeliness of high criticality tasks. With respect to task parameters, the major difference between the E-MC task model and the mixed-criticality task model considered in [23] is in the way low criticality tasks are represented. Here, an mixed-criticality elastic task τ_i is modelled as the following:

$$\tau_i = (\mathcal{L}_i, \vec{C_i}, T_i(LO), T_i(HI), P_{ER})$$

where $\mathcal{L}_i$ denotes criticality and $\vec{C_i}$ denotes a vector of execution time estimates. The criticality of tasks can be either HI (high) or LO (low). C_i denotes the WCET of τ_i. Similar to [23], this model also assumes that if $\mathcal{L}_i = HI$, then $\vec{C_i} = (C_i(LO), C_i(HI))$. Otherwise, if $\mathcal{L}_i = LO$, $\vec{C_i} = (C_i(LO))$ with the constraint: $C_i(HI) \geq C_i(LO)$. The value of $T_i(LO)$ is such that $T_i(LO) \leq T_i(HI)$ denotes the desired time period which is mainly used in existing mixed criticality scheduling algorithm and $T_i(HI)$ denotes the time period of τ_i corresponding to its minimum service requirement in the HI-crit phase.

Task	Task Parameters				
	Criticality	C_i	$T_i(LO)$	$T_i(HI)$	$P_{i,ER}$
τ_1	HI	$C_1(LO) = 3$ $C_1(HI) = 10$	25	25	-
τ_2	HI	$C_2(LO) = 2$ $C_2(HI) = 4$	10	10	-
τ_3	LO	$C_3(LO) = 2$ $C_3(HI) = 2$	8	16	$\{8\}$
τ_4	LO	$C_4(LO) = 3$ $C_4(HI) = 3$	30	40	$\{30\}$

Table 2.9: An example taskset for the elastic mixed criticality model considered in [1].

Moreover, the low criticality task τ_i has a set of k_i possible early-release points $P_{i,ER} = \{p_i^1, p_i^l, ..., p_i^k\}$ where $C_i(LO) < p_i^1 < p_i^2 < ... < p_i^k$. The early-release points of low criticality tasks enable them to release early and improve their execution frequencies at run-time through appropriate reclamation of several idle intervals (*i.e.* slack).

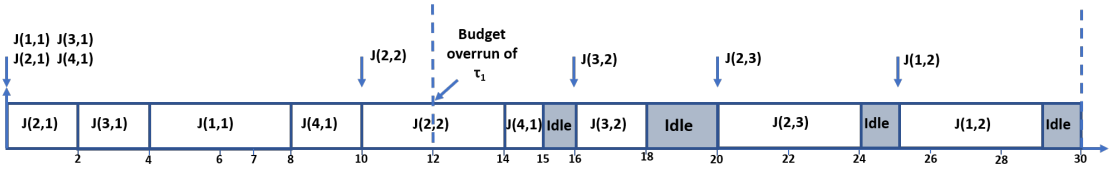

Figure 2.10: The EDF schedule with maximum periods of low criticality tasks to ensure their minimal service requirements when the system executes in a HI-crit phase.

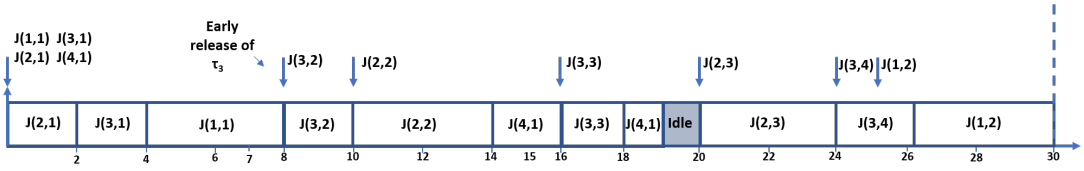

Figure 2.11: The ER-EDF schedule where the low criticality task τ_3 releases early at time 8, 16 and 24 when the system executes in a HI-crit phase.

A given set of E-MC tasks is said to be E-MC schedulable if the following conditions are satisfied:

1. $U(HI, LO)+U(LO, LO) \leq 1$ - If this condition is satisfied, it means that service requirements of both low and high criticality tasks in LO-crit phase can be guaranteed in the worst-case scenario.

2. $U(HI, HI) + U(LO, HI) \leq 1$ - If this condition is satisfied, it means that minimum service

requirements of low criticality tasks along with high-level requirements of high criticality tasks can be guaranteed in the worst-case scenario.

In the above conditions, $U(HI, LO) = \sum_{\tau_i \in \Gamma_H} C_i(LO)/T_i(LO)$ denotes the utilisation of high criticality tasks in the LO-crit phase. Γ_H and Γ_L denote a set of tasks with high (HI) and low (LO) criticality respectively. $U(HI, HI) = \sum_{\tau_i \in \Gamma_H} C_i(HI)/T_i(HI)$ denotes the utilisation of high criticality tasks in the HI-crit phase; $U(LO, LO) = \sum_{\tau_i \in \Gamma_L} C_i(LO)/T_i(LO)$ denotes the utilisation of low criticality tasks in the LO-crit phase; $U(LO, HI) = \sum_{\tau_i \in \Gamma_L} C_i(HI)/T_i(HI)$ denotes the utilisation of low criticality tasks in the HI-crit phase.

In the following, an example to show the effectiveness of modelling and scheduling of E-MC tasks is presented. Their timing parameters are given in the Table 2.9. Suppose that the second job of task τ_2 executes for more than its $C_2(HI)$, while other high criticality jobs take their corresponding low-level WCETs, then $U(HI, HI) + U(LO, LO) > 1$ for the example taskset. Therefore, it is not possible to guarantee the required service levels for the low criticality tasks τ_3 and τ_4 when system executes in the HI-crit phase. However, if the minimum service requirements of τ_3 and τ_4 can be specified by their maximum periods, which are 16 and 40 respectively, then the minimum requirements can be guaranteed under EDF (Earliest Deadline First) algorithm since the condition $U(HI, HI) + U(LO, HI) = 1$ is now satisfied. The corresponding EDF schedule for interval $[0, 30]$ is shown in Figure 2.10 . Here, we can see that there are several idle intervals (*i.e.* slack) in the EDF schedule since not all high criticality jobs take their high-level WCETs. Such slack can be exploited to improve the execution frequencies of low criticality tasks. For instance, task τ_3 can release its second job J(3,2) at time 8 (its early-release point) instead of waiting until time 16 (its maximum period) as shown in the Figure 2.11. It is important to ensure that the early releases of low criticality tasks do not affect the timeliness of high criticality tasks. For this, for each early release point, an algorithm presented in [1] is invoked to compute the available slack as well the slack demand. If the available slack is equal or higher than the slack demand, then the new jobs of low criticality tasks are released at this point.

2.2.4 Mode Change Model

In the following, a model proposed in [7] to capture functional mode change aspects (without the notion of mixed-criticality) of the real-time systems is discussed. This model can be considered as a simple extension of the classic sporadic task model [21] adopted to mode change setting. In a system that can exhibit r different modes, M_1 to M_r, any task τ_i can be represented as a set of tuples defined

as follows.

$$\tau_i = \{\tau_{i,1}, \tau_{i,2}, ..., \tau_{i,r}\}$$

In a specific mode M_p, $\tau_{i,p}$ can be given as the following,

$$\tau_{i,p} = \{C_i^{M_p}, T_i^{M_p}, D_i^{M_p}, P_i^{M_p}\}$$

where $C_i^{M_p}$ denotes the WCET of τ_i in M_p and $T_i^{M_p}, D_i^{M_p}, P_i^{M_p}$ denotes time period, deadline and priority of τ_i in M_p. When τ_i is not active in the mode M_p, then its WCET corresponding to the mode M_p can be considered as $C_i^{M_p} = 0$. Tasks have deadlines such that $D_i^{M_p} \leq T_i^{M_p}$. A mode change request (MCR) can be triggered when there is a need for the system to undergo a mode transition. An MCR is an event that can be triggered by any task to initiate the mode transition. Further, in the presence of an MCR, any task τ_i can be again classified into different types. If τ_i belongs to the old-mode, it may be:

- **old-mode completed -** $\tau_{i(O)}$**.** Such a task is active only in the old-mode but required to complete its execution even after the occurrence of MCR. Thus part of its execution may fall after the MCR.
- **Old-mode aborted -** $\tau_{i(A)}$**.** As the functionality of an aborted is no longer required in the new-mode, such a task gets aborted when the MCR occurs without loss of data consistency.

On the other side, if τ_i belongs to the new-mode, it may be:

- **New-mode, changed -** $\tau_{i(C)}$**.** Such a task is active in both modes, but the value of its parameters such as budgets, behaviours, period and deadline in the new-mode are different to those in the old-mode.
- **New-mode, unchanged -** $\tau_{i(U)}$**.** Such a task is active in both modes without any change in the values of its parameters (WCET, period and deadline).
- **Wholly new task -** $\tau_{i(W)}$. Such a task is active only in the new-mode.

2.2.4.1 Schedulability analysis for the mode change model

The study also presents a fixed priority based schedulability test for uniprocessor systems for the above task model. The worst-case response time equation for an old-mode task τ_i considers interference from other higher priority tasks belonging to the old-mode (old-mode completed tasks, aborted tasks

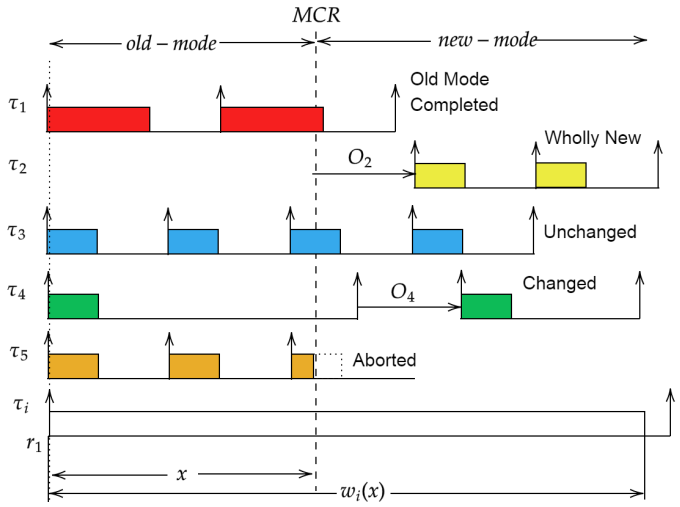

Figure 2.12: Worst-case response time for an old-mode task.

and changed tasks), from unchanged tasks and from new-mode tasks (changed and wholly new tasks). Figure 2.12 shows the temporal window $w_i(x)$ in which execution of τ_i falls along with the interference from other higher priority tasks. The total length of $w_i(x)$, as a function of x, where x denotes the time interval between the MCR arrival and the activation time of τ_i can be given by the solution to the following iterative equation.

$$\begin{aligned} w_i(x) = C_i + B_i + \sum_{j \in hp(i)_O} \left\lceil \frac{x}{T_j} \right\rceil C_j + \sum_{j \in hp(i)_N} \left\lceil \frac{w_i(x) - x - Y_j}{T_j} \right\rceil_0 C_j \\ + \sum_{j \in hp(i)_A} \left\{ \left\lfloor \frac{x}{T_j} \right\rfloor C_j + min(x - \left\lfloor \frac{x}{T_j} \right\rfloor T_j, C_j) \right\} \\ + \sum_{j \in hp(i)_U} \left\{ \left\lceil \frac{x}{T_j} \right\rceil C_j^s + \left\lceil \frac{w_i(x) - \left\lceil \frac{x}{T_j} \right\rceil T_j - Z_j}{T_j} \right\rceil_0 C_j \right\} \end{aligned} \tag{2.10}$$

The maximum blocking time for τ_i, denoted as B_i, can be calculated based on the equations proposed in [33]. $hp_O(i)$ denotes the set of old-mode completed tasks with priority higher than τ_i; $hp(i)_A$ denotes the set of old-mode aborted tasks with priority higher than τ_i; $hp(i)_N$ denotes the set new-mode tasks (changed and wholly-new tasks) with priority higher than τ_i. For new-mode tasks, two kinds of offsets are used. The offset Y denotes the time delay relative to x (MCR instant) applied for the wholly new and changed tasks whereas the offset Z is the time delay relative to the end of the period corresponding to the old-mode release of unchanged tasks.

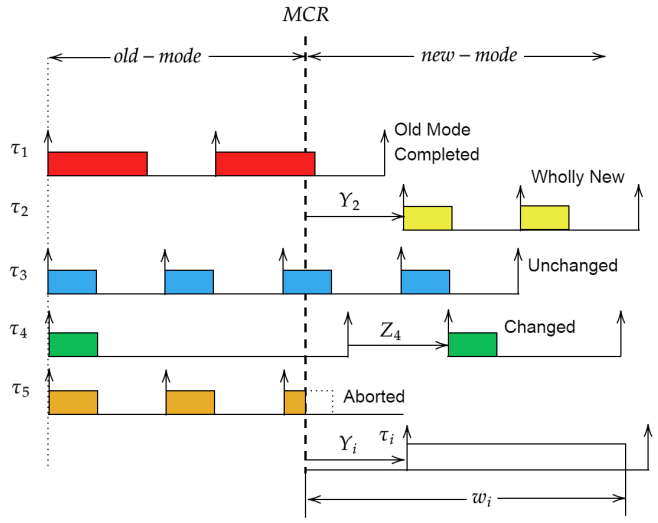

Figure 2.13: Worst-case response time for a new-mode task.

To compute the worst-case response time for the new-mode task, a temporal window w_i as shown in Figure 2.13 is considered. As the new-mode tasks may be released after a certain time delay (offset), the worst-case response time is given by the subtraction of this offset to the solution of the iterative equation 2.11.

$$w_i = C_i + B_i + \sum_{j \in hp(i)_O} C_j + \sum_{j \in hp(i)_N} \left\lceil \frac{w_i - Y_j}{T_j} \right\rceil_0 C_j + \sum_{j \in hp(i)_U} \left\{ C_j + \left\lceil \frac{w_i - T_j - Z_j}{T_j} \right\rceil_0 C_j \right\} \quad (2.11)$$

$$R_i = w_i - O_i \quad (2.12)$$

A task is considered to be schedulable if the computed value of its worst-case response time is less than or equal to its deadline. The value of the worst-case response time is also useful to decide the offset values for tasks to achieve the schedulability of the mode transition.

Chapter 3

Related Work

This chapter aims to compare task models that are categorized based on their ability to handle system overload caused due to service level violation of tasks or due to mode transitions. A good number of studies that are related to the design and analysis of mode change protocols use the terminology 'mode' to denote a specific operational situation. For example, depending on road type, the vehicle can be considered to be driven in either city road mode or highway mode. These studies focus on handling system overload due to a temporary spike in resource consumption pattern when tasks belonging to multiple modes execute together during mode transitions. Another category of studies can be related to the design of Mixed Criticality Systems (MCS) to handle system overload caused by timing faults due to service level violation of tasks. In the following, a detailed classification of mode change protocols and MCS task models is discussed.

3.1 Mixed Criticality System Models

Steve Vestal first proposed a model for MCS [15] in 2007 with a fixed priority scheduling algorithm based on Audsley's priority assignment strategy [37]. In this model, each task is assigned with a worst-case execution time (WCET) and additional execution estimates which are done with different levels of pessimism. The number of estimates is proportional to the criticality level of a task. This model does not guarantee the execution of lower critical tasks to meet their deadline under the occurrence of system overload due to the service level violation of high critical tasks. Vestal's model was further improved with certain assumptions which led to a standard MCS model [38], [19]. For simplicity of analysis, the standard MCS model considered a dual-criticality system where a task can be classified either as

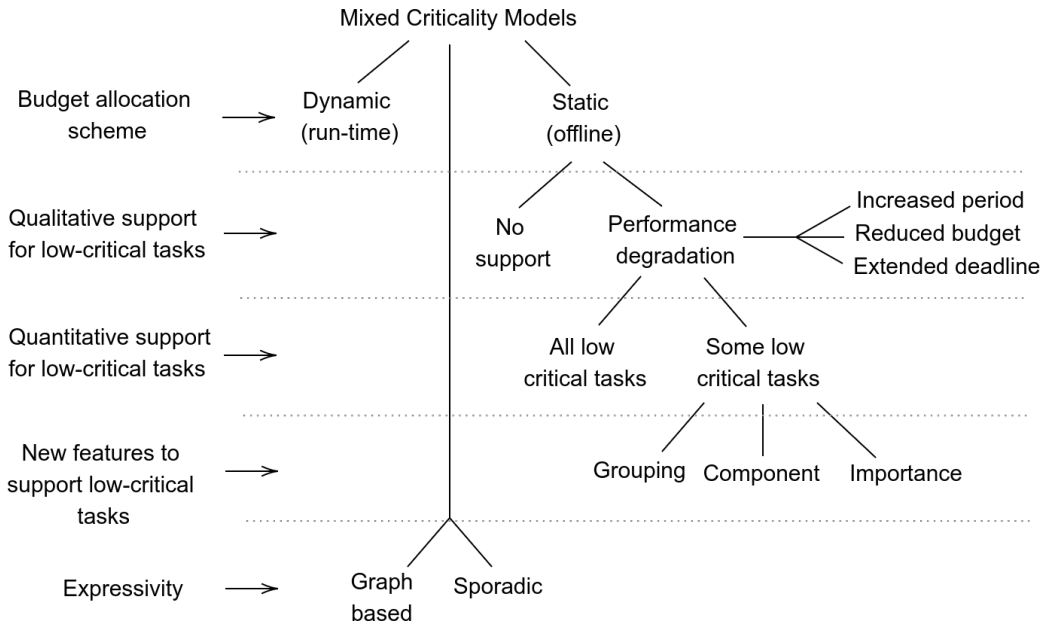

Figure 3.1: Classification of MCS task models.

low critical or high critical. The system can exist in two possible phases. These phases are denoted as LO-crit and HI-crit phase. A high critical task is provided with a WCET and an additional execution estimate which corresponds to its service levels in LO-crit phase of the system, whereas a low critical task is provided with only one execution estimate. Violation of service levels by high critical tasks caused the system transition from LO-crit to HI-crit phase. Studies that were based on the standard model such as [39], [38], [40], [19], [41], [20] share a pessimistic strategy that service levels for tasks whose criticality is relatively lower than that of the task violating its service level will be immediately degraded to zero (*i.e.*, the tasks are dropped) once the system switches to HI-crit phase. As a result, these works faced criticism from system engineers as they could not meet their expectations like guaranteed execution for low critical tasks in HI-crit phase, suitable conditions under which system can return to LO-crit phase etc., Some of these criticisms were later addressed, which led to an improved model as proposed in [26]. In this model, the low critical tasks are not abandoned when the system moves to HI-crit phase. The model assumes that the system switches back to LO-crit phase at the idle instant, and the budgets of all low critical tasks are restored to their original values. When the system moves to HI-crit phase, some guarantee is given to low critical tasks instead of dropping them. This is achieved with either assigning them the lowest priority among all high criticality tasks or with reduced budgets

or by extending their time periods.

In the following, existing MCS task models are classified based on how they handle low criticality tasks to handle system overload due to service level violation of any high critical tasks. A classification of these models is shown in Figure 3.1.

3.1.1 Classification Based on the Parameters Used for Selecting Tasks for Degradation

A subset of existing studies [26], [42], [1], [43], [44], that are extended from the improved standard sporadic task model, ensures a guaranteed but degraded support for all low critical tasks when the system executes in HI-crit phase. Some of the notable non-MCS works such as elastic task model [36] and weakly hard real-time system model [4] which are designed for the adaptive control system and hard real-time systems with predictable deadline misses have been extended to MCS. For example, works like elastic MCS model [1] and MCS with weakly hard constraints [45] are purely motivated from [36] and [4] respectively. The key idea of elastic MCS task model is to have multiple task periods (*i.e.* multiple service level intervals) for low critical tasks whose minimum service requirements is guaranteed by its largest period. However, at runtime, low critical tasks can be released early by exploiting the slack time generated from the over-provisioned budgets for high critical tasks to reduce their service intervals and thus improve their service levels. Degradation of low critical tasks by reducing their budgets is similar to existing works related to the imprecise computation technique [46], where workload models representing acceptable imprecise computations provide scheduling flexibility by trading off result quality to meet computation deadlines. The major drawback of the above studies is their pessimistic assumptions in handling service level violations of high criticality tasks. In HI-crit phase, all low critical tasks are degraded to recover the system from service level violation of high critical tasks. Also, all high criticality tasks are assumed to violate their service level simultaneously. These assumptions are sometimes criticised for its unrealistic pessimism. However, some studies which followed the improved model tried to address these issues by incorporating certain features which focused on providing execution guarantees to low critical tasks in HI-crit phase. These models can be further classified based on three criteria.

In some studies [47], [48], [49] belonging to the first category, only selected low critical tasks were degraded based on the service level violation of a specific high critical task by either decreasing budget or increasing period or extending the deadline. These models use additional features like task grouping, importance level, etc., to decide on specific low critical tasks to be degraded. In [47], the system consists of a set of tasks of different criticality grouped into separate software components. Each component is assigned a parameter called 'Tolerance Limit'. When any high critical task belonging to a component

violates its service level, all low critical tasks belonging only to this component are degraded. Tolerance Limit of a software component denotes the maximum number of simultaneous service level violation of high critical tasks that can be allowed without impacting the low critical tasks of other software components. At run-time, when the number of high critical tasks violating their service levels exceeds the Tolerance Limit, low critical tasks of other components are impacted. In [49] each group consists of one high critical task and a set of low critical tasks in contrast to the previous study where a component can have any number of high and low critical tasks. In [48], each low critical task is specified with an additional parameter called 'Importance'. When the system moves to a HI-crit phase, the importance value is used to decide which low critical tasks have to be impacted first.

The studies related to graph-based MCS models such as Mode-Switching Digraph Real-Time (MS-DRT) task model [31] and Interference Constraint Graph (ICG) [30] are highly generic and hence can model service level violation, task degradation, phase transitions and offsets for tasks. This gives a designer, flexibility to specify the behaviour of each task instance (or job) upon service level violation. Since not all low critical tasks have to be degraded, models like these have shown improvement in schedulability of low critical tasks in HI-crit phase. Depending on the task that has violated its service level, a designer can specify a set of tasks to be degraded independently of overrun task or system parameters. In [31], a designer is allowed to specify a fixed time interval after which tasks can be degraded once the overload occurs and give the flexibility to consider a change in all the task parameters (budget, period, deadline etc.). In Interference Constraint Graph (ICG) model, a designer can specify the amount of interference that can be allowed between tasks. In this model, each task is represented by a vertex. The directed edge between source and the destination vertices is represented by a value denoting the maximum possible interference that can be allowed by the source vertex to the destination vertex under service level violation of the destination vertex task. In MS-DRT task model, each vertex of the graph represents an instance of a task (job). Vertices are connected by the edges represented by straight or wiggly lines. The transition between the vertices via straight edge represents a possible task control flow (during functional mode changes) whereas transition via a wiggly line represents a change in system phase due to service level violation or degradation. The model differs from previous ones by not assuming any specific event (like mode change request, service level violation or any other faults) that will cause the transition between modes, but gives complete freedom to a designer to specify the event. The model can represent two or more number of modes and corresponding transitions.

3.1.2 Classification Based on the Budget Allocation Scheme Considered for a Task

Studies belonging to the previous classification allow a designer to specify multiple execution estimates for the high criticality tasks statically (off-line). Since the Certification Authorities (CA) may not be convinced with multiple execution estimates with different levels of pessimism as claimed in [50], some existing studies such as [51] and [52] consider the run-time allocation of budgets for high critical tasks in LO-crit phase of the system and have shown a significant improvement in schedulability and resource utilization. In dynamic mixed-criticality task and scheduling model [51], a combination of off-line and on-line techniques are used to improve the schedulability of tasks. Off-line, they use a schedulability analysis to determine total LO-crit phase budget for all high critical tasks combined and at run-time, they use a strategy to allocate budgets to individual jobs of high critical tasks based on this total budget. In [52] an adaptive reservation mechanism is used to cope up with uncertainties in task execution times owing to the difficulty of determining the exact worst-case Execution Time for the tasks. In this study, tasks' execution history is used to predict their required computational resource in near future, and the reserved budgets are adjusted according to their criticality levels. The model is targeted for soft real-time systems where some deadline misses are acceptable.

3.1.3 Classification Based on the Timeliness of Task Degradation

Timeliness of degradation refers to the graceful degradation of tasks at an appropriate time so that both the safety with reduced performance and an efficient processor utilization can be guaranteed. Existing MCS studies can be classified into three types.

1. The first type of studies such as [42, 53–55], consider a dynamic scheme where the tasks are degraded only if sufficient run-time slack to handle the overload is not available. These studies focus on improving processor utilization by postponing degradation as much as possible.

2. The second type comprises a majority of existing studies which consider a simple static strategy of degrading the selected tasks immediately at the time instant when the overload occurs.

3. In the third type studies such as [31], a designer is allowed to specify a fixed time interval after which tasks can be degraded once the service level violation of task occurs.

To restore the service guarantees for low critical tasks, a majority of the existing models consider that system returns to the LO-crit phase at the idle instant. One study, bailout protocol [56] reduces

the negative impact on low critical tasks via a timely return to LO-crit phase. The bailout protocol considered in this study considers that at any given time, the system can exist in one of three phases. They are normal phase, bailout phase and recovery phase. Here, a parameter called 'Bailout Fund' keeps track of the budget consumed by high critical task. When it violates the service level, the system switches from normal to bailout phase. When it becomes zero, the system first switches to the recovery phase. If no service level violation of high criticality task occurs and when all jobs of high criticality task finish, the system switches to normal phase.

3.1.4 Classification Based on the Number of Criticality Levels Considered for a Task

A vast majority of existing studies assume a fixed criticality value for a task whereas the model proposed in [57] assume that depending on the functional mode, criticality level of the safety functionality may change. This model motivates the need for considering a change in the criticality value of the task by providing an example from the automotive domain. The suspension control task can be considered as a critical part of the stability of the car at high speed, but it is only a comfort feature at low speed. Therefore, suspension task is assigned with higher criticality as it is crucial to ensure timing guarantees in high-speed mode. A criticality violation is said to occur when lower critical jobs prevent a job of a high criticality task from meeting its deadline. This study focused on avoiding the 'Criticality Violation' during mode transition. An important assumption made was that upon the occurrence of Mode Change Request (MCR), change in task parameters like criticality for new-mode changed tasks are only reflected for the jobs released in the new-mode.

3.2 Mode Change Protocols

Sometimes, tasks which are schedulable in a mode may not be schedulable during a mode transition due to additional load from tasks activated in the new-mode. Several mode change models consider a set of rules also known as mode change protocol to ensure mode transitions happen without violating properties such as periodicity, promptness and consistency. These protocols were designed for ensuring timing guarantees during mode transitions on both uniprocessor [58], [59], [35], [7] and multiprocessor [60], [61], [62] systems. These protocols focused on 5 categories of tasks. They are old-mode completed tasks, wholly new tasks, new-mode changed tasks and new-mode unchanged tasks. Details of each type is discussed in section 2.2.4. A classification of these protocols is shown in Figure 3.2. Some key observations of existing mode change protocols are as follows.

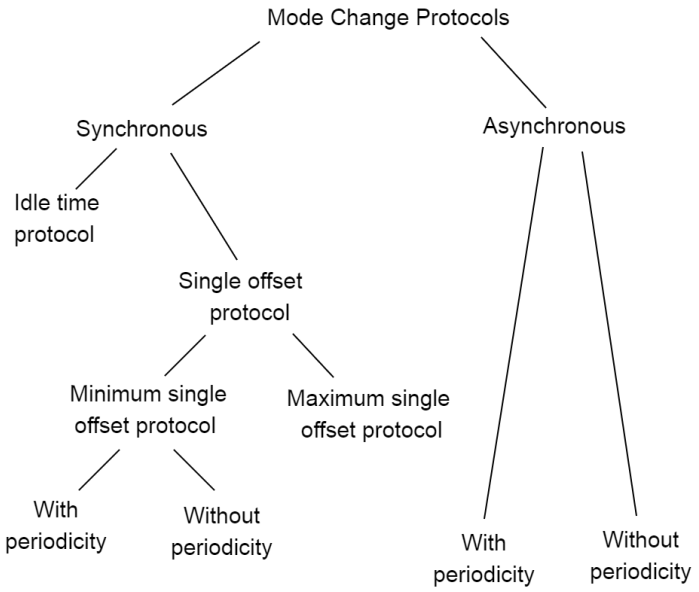

Figure 3.2: Classification of mode change protocols.

1. Depending on the activation patterns of wholly new tasks, protocols are classified as synchronous and asynchronous.

2. In synchronous protocols [6, 33, 35], both old-mode completed tasks and wholly new tasks are allowed to execute simultaneously.

3. In asynchronous protocols [8], the activation patterns for wholly new tasks are delayed by a certain offset (time delay). The calculation of offset varies depending on the protocol. The minimum value of the time period or the maximum WCET among all tasks influence the offset calculations.

4. Classification of protocols with periodicity [9] and without periodicity [63] is arrived based on the activation patterns of unchanged tasks during mode transition. Activation pattern where the periodicity of unchanged tasks is unaltered are known as protocols with periodicity. The protocols where the periodicity of the unchanged tasks are affected during mode transition are known as protocols without periodicity. In these protocols, the value of the time period of tasks remains the same.

5. Existing mode change task models consider only the changes in period and WCET of a changed task for schedulability analysis during mode transition.

3.2.1 Synchronous Mode Change Protocols

In the literature, a protocol is said to be synchronous if it never releases the wholly new tasks before all the instances of old-mode completed tasks active in new-mode have completed their execution. Various classifications of synchronous protocols are discussed in the following paragraphs.

In Idle time protocol proposed in [35], the arrival of a Mode Change Request (MCR) does not affect the normal activation patterns of old-mode completed tasks until an idle instance (no processor load) is detected. At idle instance, mode change routine deactivates old-mode completed tasks, performs a mode change specific actions and then, releases new-mode tasks. Although the protocol is simple, it lacks promptness where the release of the new-mode tasks is entirely dependent on the workload of old-mode completed tasks.

In another classification of mode change protocols, where the activation of wholly new tasks are delayed to certain time instance after MCR are termed as single offset protocols. The time delay is called an offset. Offset is the same for all wholly new tasks. Some of these protocols preserve the periodicity of unchanged tasks while others break it. Depending on how they treat unchanged tasks, they are further classified into three types. Type 1 is referred to as maximum period offset protocol [6]. In this protocol, wholly new tasks are delayed for a time equal to a period of less frequent tasks in both modes. Activation patterns of unchanged tasks are not affected, and hence it preserves the periodicity. The drawback of the protocol is that it lacks promptness. Since it is synchronous, schedulability analysis becomes easier. In type 2, known as a minimum single offset (without periodicity) [33], all old-mode completed tasks are allowed to finish their execution but are not released further. Hence offset value is set to sum of WCET of all old-mode completed tasks. This improves the promptness of the protocol significantly when compared to the minimum period offset protocol. Type 3 is known as minimum single offset protocol (with periodicity) [33] which is an extension of the type 2 protocol with consideration of the unchanged tasks. Activation patterns of the unchanged tasks remain the same in the new-mode.

3.2.2 Asynchronous Mode Change Protocols

In these protocols, both old-mode completed and wholly new-mode tasks are executed simultaneously to produce faster transitions when MCR occurs. These protocols need a specific schedulability analysis of the transition that takes into account the simultaneous execution of different task categories. Asynchronous protocols are further classified into three types as utilization based protocol, protocols with and without periodicity.

The utilization based protocol [8] requires a dynamic log of processor utilisation to calculate the available processor capacity at a give time instant. Based on the available capacity, time instance for release of wholly new tasks are calculated. But since the protocol does not consider the worst-case phasing of MCR, it may fail for certain task combinations as pointed out in [64]. In asynchronous protocol with periodicity [64], the last activation of the old-mode completed tasks is allowed to finish even after the MCR has occurred after which they are deactivated (as opposed to idle time protocol). The activation patterns of the unchanged tasks remain the same even with the occurrence of MCR. The new-mode changed tasks are released right after the end of their corresponding old-mode version. The wholly new tasks are released after sufficient offset after the MCR. The protocol supports the model where a task can have different WCET in different modes.

The asynchronous protocol without periodicity [63] differs from the previous ones by introducing an offset for all new-mode tasks including for those whose parameters are changed or unchanged. Hence they are called without periodicity. The advantage of asynchronous protocol is that mode transition can be made schedulable since all categories of tasks in new-mode can be delayed at will. Also, the schedulability analysis is simplified since exact instances at which tasks will be activated are known. Since asynchronous protocol without periodicity considerably improves promptness in reaction for the MCR, this can be considered to be practical enough to get adopted to the automotive domain.

3.3 Key Observations

Existing MCS models that adopted the improved standard sporadic task model [26] operate on the fundamental assumption that only low criticality tasks are allowed to be degraded when service level violation of any high criticality task occurs. None of these studies considers the possibility of degrading high criticality tasks to recover the system from service level violation of any lower criticality task. These studies do not consider the possibility of multiple ways of degrading a task when the system operates in a specific mode. These assumptions severely limit the flexibility to consider a specific way of degrading a task in a specific mode to handle system overload due to service level violation of tasks.

A subset of MCS task models such as [1] and [65] consider dual-criticality systems where degradation of low criticality tasks is achieved by increasing their time period when any task violates its service levels. They improve the quality of service for low critical tasks by considering the concept of early release (detailed in the section 2.2.3) to utilise the slack that is generated when any high criticality task does not execute till its pessimistic WCET. However, these models consider varying time periods

for tasks as their degradation strategy and hence their timing analysis used for computing slack do not consider multiple degraded budgets. The concept of early release of low criticality tasks is also considered in [65]. However, the framework proposed in [65] is designed for multi-rate, mixed criticality systems for synchronous languages on multi-processors. In this study, 3 criticality levels (life, mission and non-critical) for tasks are considered. The proposed models consider reduction in task's budget as its degradation strategy as opposed to the models discussed in [1] and [65] where change in the time period of tasks is considered as the degradation strategy.

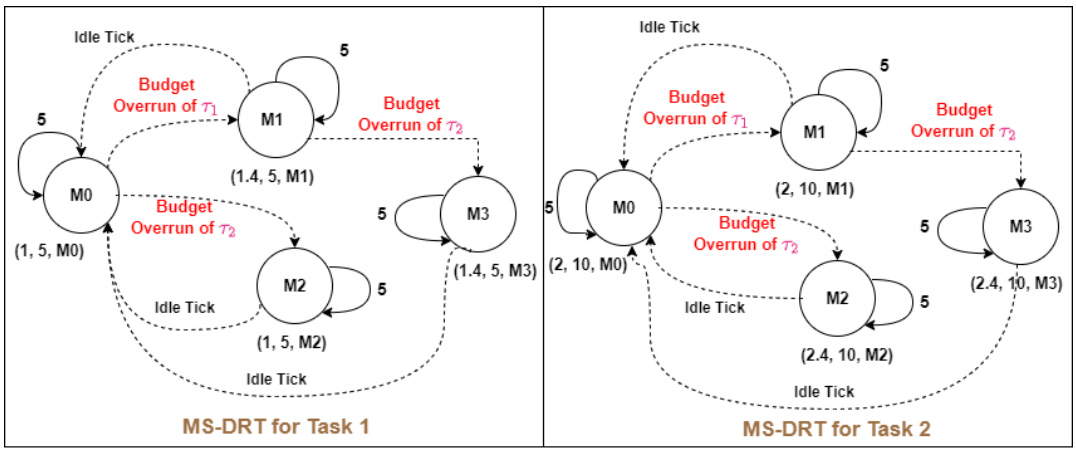

Figure 3.3: Specification of task parameters as per the MS-DRT model.

A major drawback of the graph based task models such as MS-DRT is that they suffer from state explosion problem leading to higher levels of complexity for task modelling and their timing analysis. In these studies, a mode represents an unique combination of timing parameters allocated for tasks. All tasks in such systems are considered to run in the same mode at any particular time point, *i.e.* the modes are system wide. To perform timing analysis, a designer is required to specify values of the timing parameters for tasks (budgets, time period etc.) in each mode. Therefore, at worst-case, for a system with n tasks, total number of budget combinations of tasks that need to be considered is of order $O(2^n)$. With such an approach, the specification of budgets that needs to be allocated for tasks in each mode require higher number of vertices and edges of order $O(2^n)$. Hence, timing analysis for such models tend to become harder when system has to be modelled for multiple modes and mode transitions. The above issues are explained using the following example. Consider a mixed-criticality system consisting of 2 tasks. Let these tasks be denoted as τ_1 and τ_2. Consider the following system behaviour to be achieved.

- Under normal conditions (when no budget overrun has occurred), all tasks should be allocated

with their budget estimated with low pessimism (normal budget).

- When any budget overrun is detected, only the task that has overrun its low pessimistic budget should be allocated with its budget estimated with high pessimism (safe budget) without affecting other tasks.

To specify the parameters to achieve the above system behaviour using the MS-DRT model, budget overrun events of tasks that can lead to change in budget allocated for tasks are captured as separate modes. Therefore, for a system with 2 tasks, one must consider 4 different modes as shown in Figure 3.3. Tasks τ_1 and τ_2 are modelled as separate MS-DRT graphs. Under normal conditions, system executes in mode $M0$. When τ_1 overruns its normal budget, system transition to the mode $M1$ happens instantaneously. In $M1$, τ_1 is allocated with its safe budget of 1.4 units while τ_2 continues to executes with its normal budget. When τ_2 overruns its normal budget, depending on the system mode, transition to the mode $M3$ or $M2$ happens instantaneously. Upon the detection of idle tick, system transition to $M0$ happens instantaneously.

The main idea behind the proposed task models in this thesis is to strike a balance between the higher complexity of the task specification and analysis of the graph based models and the conservatism of the more standard Response Time Analysis (RTA) approach. Hence, the proposed models are just an extension of the standard mixed-criticality sporadic task model to capture multiple ways of degradation and the ability to achieve higher level of flexibility in the choice of tasks to be degraded. Therefore, to achieve the system behaviour discussed above, tasks can be specified with the parameters of the CA-MCS as shown below.

$$\tau_1 = \{T_1, \vec{B_1}, C_1^n, C_1^s, \vec{C_1}, \mathcal{L}_1\} = \{5, \vec{B_1} = (B_1^1 = Safe, B_1^2 = Normal), 1.0, 1.4, \vec{C_1} = (), 2\}$$

Similarly, τ_2 can be specified as the following:

$$\tau_2 = \{T_2, \vec{B_2}, C_2^n, C_2^s, \vec{C_2}, \mathcal{L}_2\} = \{10, \vec{B_2} = (B_2^1 = Normal, B_2^2 = Safe), 2.0, 2.4, \vec{C_2} = (), 1\}$$

where C_1^n and C_2^n denote normal budgets of τ_1 and τ_2 respectively. C_1^n and C_2^n denote safe budgets of τ_1 and τ_2 respectively. B_1^1 denotes behaviour of the τ_1 when it overruns its normal budget. B_1^2 denotes behaviour of the τ_1 when τ_2 overruns its normal budget. $\vec{C_1}$ and $\vec{C_1}$ denote a vector of degraded budgets of τ_1 and τ_2. As no tasks are degraded in the example considered, they are empty. With these parameters, the semantics of the CA-MCS model is designed such that at run-time when a budget overrun of task is detected decision regarding the budget that needs to be allocated to tasks can be arrived based on the

behaviour parameter of tasks.

In [66], a semantic framework for the multi-mode real-time systems is proposed. The framework is highly expressive and can model both mixed-criticality and mode change aspects. With this framework, a designer can precisely specify behaviours of tasks during mode transitions such as a change in the time period, criticality, worst-case execution time, and ways to handle the pending events of the old-mode tasks and offset values for tasks released in the new-mode. However, the framework does not capture multiple degraded budgets for a task. These budgets will be required to precisely specify the budget that needs to be allocated for degraded tasks when service level violation of tasks occur during the mode transition.

Studies [57] and [55] considered both mode change and mixed-criticality aspects of the real-time system. The task model proposed in [55] supports different degraded budgets for a task in different modes. However, tasks cannot be specified with multiple degraded budgets for a given mode. Also, when any task violates its service level, these studies do not consider the possibility of degradation of tasks with criticality level higher than the overloading task.

Although the existing mode change protocols handle mode transition for different categories of tasks, they are not designed for mixed criticality systems where service level violation of tasks can threaten the system's safety. In reality, safety-critical systems such as automotive exhibit both multi-modal and mixed-criticality behaviours. For such systems, there is a need to have a well-defined mode change protocol to handle service level violations from both old- and new-mode tasks. The schedulability analysis has to consider both mixed-criticality as well as mode change behaviours to ensure that mode transitions are safe even if service level violation of old-mode and new-mode tasks occur. As the impact of service level violation of old-mode tasks during the mode transition might be carried over to the new-mode, existing mode change protocols do not provide clarity as to which tasks have to be degraded in the new-mode to handle service level violation of old-mode tasks.

Chapter 4

A Practical Degradation Model for Mixed-Criticality Systems

4.1 Introduction

In this chapter, a new task model termed as Context-Aware Mixed Criticality System (CA-MCS) [1] model is proposed. The CA-MCS model is designed to capture some key observations of MCS that are derived based on a case study related to the automotive domain and from functional safety standards such as ISO 26262. As discussed in Chapter 3, to handle system overload due to timing faults such as budget overrun of tasks, a majority of existing MCS models seem to give too much emphasis on the criticality to decide on the tasks to be degraded. Upon a system overload, studies [19], [39], [38], [40], [41] consider suspension of lower criticality tasks, whereas [26], [1] consider reduced budget or an increased inter-arrival time of lower criticality as the degradation strategy. But, an automotive case study discussed in [25] considers different ways of degrading a task performing safety-critical functionality related to environment perception. This motivates the need for the MCS model to decouple the task's criticality level and the decision involved in the choice of tasks to be degraded. Graph-based models such as [30] and [31] are highly generic but lead to a higher level of complexity both in modelling and timing analysis of the system. To overcome these issues, syntax and semantics of the CA-MCS model has been designed to give a higher level of flexibility not only in the choice of tasks to be degraded but also in the way in which tasks can be degraded, and use the criticality information to achieve desired performance degradation without impacting the system's safety.

[1] The work in this chapter has been published in [67]

In Section 4.2, the motivation for the CA-MCS model and some important observations of the MCS are discussed. Section 4.3 presents the syntax and the semantics of the CA-MCS model. In Section 4.4, a fixed priority response time analysis based sufficient schedulability test for the CA-MCS model is derived. In Section 4.5, experiments based on the synthetic tasksets and their results are discussed. Section 4.6 concludes this chapter.

4.2 Motivation for the CA-MCS model

4.2.1 Notion of Criticality

Existing dual criticality MCS task models (reviewed in Chapter 3) consider criticality to achieve graceful degradation of the system either by suspending or degrading tasks with relatively lower criticality level than the overloading task. The idea of criticality for tasks is borrowed from the concept of the design assurance level (DAL) or automotive safety integrity level (ASIL) discussed in functional safety standards such as ISO 26262 [14]. Studies such as [68] and [50] criticise MCS task models on using ASILs/DALs assigned to software tasks as their criticality value. They provide sufficient reasons as to why any MCS task model should not consider the decision on the tasks to be suspended/degraded based only on ASIL/DAL values. The main reasons are that the value of the ASILs/DALs that is actually been assigned to a software task may not be same as that of safety functionality performed by that task or may not reflect the true severity of the consequence of the failure of safety functionality performed by that task. These issues are elaborated with examples in the following paragraphs.

4.2.1.1 Factors and Criticality

In automotive, based on the ISO 26262's ASIL assignment process described in Chapter 2, ASIL for a safety functionality is derived based on severity, controllability and probability of exposure to the hazard that might occur when it malfunctions. A hazard with the lowest probability of exposure but with a higher severity class than others may be assigned with either equal or sometimes with lower ASIL value. Consequently, a software task can be assigned with a low ASIL due to the following reasons:

1. Probability of the occurrence of hazards due to the failure of the safety functionality is low.
2. Severity of the catastrophic effects due to the failure of the safety functionality may be low.
3. When a vehicle can be easily controlled even after the failure of the safety functionality.

Also, any safety integrity level that is derived based on probabilistic factors is a matter of concern for a designer. For example, consider a Hill Hold Application (HHA) which prevents roll-back of the vehicle when driven on hills [69]. Although HHA is vital for the vehicle, if the occurrence of a hazard due to the failure of HHA when riding the vehicle on a hill is considered as a rare event (example considered for exposure level in ISO 26262), HHA may be certified to lower ASIL [70]. Suspension or degradation of HHA owing to its lower ASIL may not be appropriate under some driving conditions.

4.2.1.2 Decomposition of safety requirements

Sometimes, a software or hardware element may not be capable to satisfy the stringent safety requirements required for the criticality level derived for the functionality. For example, consider a radar sensor for detecting the lead vehicle by a safety application such as Collision Avoidance (CA). Since CA is certified to ASIL D [71], hardware and software elements of radar should also meet ASIL D requirements. According to [72], FIT (1 FIT = 1 Failure In Time = 10^{-9} failures / hour) rate of radar is estimated to be around 30 FIT which corresponds only to ASIL C and not ASIL D (requires 10 FIT). To handle this, safety standards allow a designer to employ sufficient redundant elements so that higher safety requirements can be decomposed to a set of lower safety requirements which can then be allocated to redundant elements. ISO 26262 terms this as ASIL decomposition. ISO 26262 requires redundant elements to be completely independent, *i.e.* failure of one element should not affect others. Thus a designer may combine radar and an additional sensor (say V2V - Vehicle to Vehicle Communication) certified to ASIL B or lower to achieve the required ASIL D safety requirements. Since radar and V2V are based on completely different technologies, any failure in radar cannot affect V2V and vice-versa and can be considered as independent sensors. We can observe that even if V2V is certified to a lower ASIL, it can be used to achieve higher ASIL functionality.

From above discussions it is clear that the determination of ASILs to a software task is the result of a more complex assessment and design process. When decomposition of safety requirements is considered, ASIL value that is actually assigned to a software task may not be same as the ASIL of the safety functionality implemented by that task. Similarly a lower ASIL due to a lower probability of occurrence of a hazard can sometimes downplay the catastrophic effects due to the malfunction of the safety functionality performed by the software. Therefore when ASIL of a software task is used as its criticality value *i.e.* considering lower ASILs as less critical, it is not appropriate to use only the criticality value of a task to decide on the tasks to be degraded/suspended to handle system overload. Based on the above discussion, the following observation regarding MCS is presented.

Observation O1: *With any MCS task model, when the DALs/ASILs assigned to a task is also used as its criticality parameter then the decision to degrade or suspend a task based only on its criticality value can be inappropriate.*

4.2.2 Criticality and Graceful Degradation

An automotive application can be associated with both safety and performance functionalities. To handle timing faults, such as budget overruns, the possibility of compromising the performance of functionality without affecting its safety can still be considered. LIDAR (Light Detection and Ranging) is a commonly used sensor in an autonomous vehicle for perceiving the environment. Since any error in LIDAR processing may result in catastrophic consequences, it has to be certified to ASIL D. A case study in [25] demonstrates that multiple ways of degradation of a LIDAR task can be considered depending on the application and situation that demands the sensor data. The study considers a LIDAR processing task whose computing power depends on 4 parameters that determine the LIDAR configuration. They are acquisition angle, angular resolution, scanning frequency and the need for echo amplitudes. The required configuration (for *e.g.* acquisition angle) depends on the application. Depending on the speed range, ACC requires 16°-80° acquisition angle, whereas higher acquisition angles of about 180° are required for parking assist to prevent a collision with pedestrians and other vehicles. Also, depending on the dimensions of the objects to be detected, different angular resolutions can be used. For pedestrian detection, higher resolution is necessary in contrast to lead vehicle detection for CA. Under normal conditions (absence of timing faults), LIDAR can be configured with higher acquisition angle and angular resolution. When a timing fault occurs, one or more combinations of these parameters can be degraded to reduce the computing power to handle system overload. For example, in a highway, when a task of ACC overruns its budget, the angular resolution of LIDAR can be reduced without affecting CA application. Similarly, when any task of Steering Control (SC) causes the timing fault, degradation of LIDAR task can be achieved by reducing the acquisition angle without affecting ACC or CA application. Based on this case study, the following three observations regarding task degradation are presented. **Observation O2:** *Degradation of a task assigned with lower or higher ASIL can be achieved, in certain instances, without affecting the safety of the system.* **Observation O3:** *Multiple ways of degrading a task's budget is also possible.* **Observation O4:** *The flexibility to choose a specific degradation of a task depending on the task which has caused a timing fault can be useful in the design of MCS.*

Mode ↓	Tasks Functionality		
	Adaptive Cruise Control (ACC)	**Forward Collision Warning (FCW)**	**Steering Control (SC)**
Cruise Control - Constant Distance Mode	1. Apply throttle or brake computed to maintain constant distance between the lead vehicle and the follower vehicle. 2. Dynamic Speed Adaptation for a smooth maneuvering through curves.	Issue warnings when distance between vehicles is less than 15m.	Apply suitable steering commands to ensure vehicle is moving along the center of the track.

Table 4.1: Automotive applications and their functionalities.

4.2.3 Expressiveness of the Model

An important challenge that needs to be addressed by any MCS model is to clearly specify the budget that needs to be allocated for a task when one or more tasks overrun their budgets. To achieve this, graph-based models such as [30] and [31] can be considered as they are highly generic and can model budget overruns, task degradation, mode transitions and offsets for tasks. All possible combinations of tasks that can overrun their budgets at run-time can be modelled and hence leading to a higher level of complexity both in modelling and timing analysis of a system especially when tasks exhibit observations **O2**, **O3** and **O4**. To demonstrate the complexity involved in the task's budget specification, a steer control task is modelled as per the Mode-Switching Digraph Real-Time (MS-DRT) model proposed in [31]. The details of the system executing 3 automotive applications - Adaptive Cruise Control (ACC), Steer Control (SC) and Forward Collision Warning (FCW) are provided in Table 4.1.

The MS-DRT model of the SC task is shown in Figure 4.1. Each vertex represents the job type of SC. Each vertex is labelled with 3 parameters representing budget, relative deadline and mode of the corresponding job type. Four different estimations are considered for the SC task. They include one normal budget (estimated with low pessimism), one safe budget (estimated with high pessimism) and two different degraded budgets. The degraded budgets can be allocated to the SC task to handle budget overrun of ACC and FCW tasks respectively. It is assumed that degraded budgets are estimated with higher levels of pessimism. Hence, modes of SC tasks differs from one another with their budgets. The relative deadline of the SC task is assumed to be same for all modes. As per the MS-DRT model, each directed edge drawn with dotted lines represent possible mode switches.

A mode switch happens when budget overrun of any task occurs. Although there are three tasks in total only 4 possible budget overrun combinations of tasks are assumed to affect the budget allocated

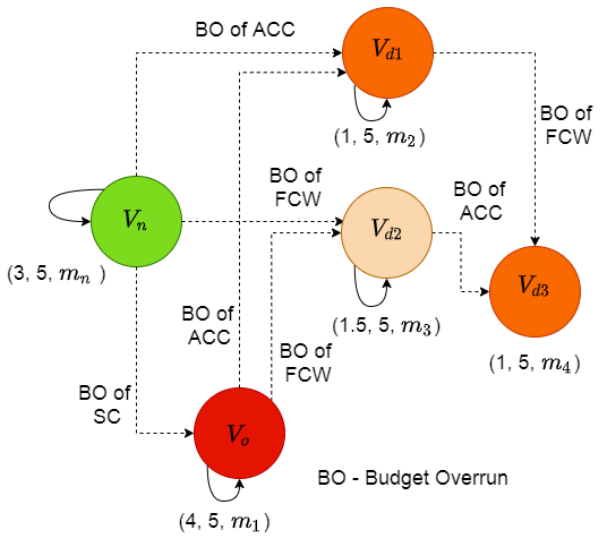

Figure 4.1: MS-DRT model for the steer control task considered in Table 4.1.

to SC task. They are budget overrun of SC only, budget overrun of ACC only, budget overrun of FCW only, budget overrun of both FCW and ACC. Therefore, 4 mode switches are considered. They are:

- When no task has overrun its normal budget, all tasks continue to execute with their normal budget and no system mode switch occurs. For the SC task, this rule is represented as the Vertex V_n with values (3, 5, m_n) where 3 denotes the normal budget (less pessimistic budget), 5 denotes the time period and m_n denotes the normal mode of the SC task.

- When any task overrun its normal budget, it is immediately allocated with its safe budget (pessimistic budget). In the example considered, when the SC task overruns its normal budget, the system switches from the mode m_n to m_1 . This is represented as the Vertex V_o with values (4, 5, m_1) where 4 denotes the safe budget of the SC task, 5 denotes its time period and m_1 denotes the safe mode of the SC task.

- While the system is in mode m_n or in m_1 and when ACC task overrun its normal budget, the system switches to the mode m_2. SC task is updated with its degraded budget corresponding to ACC task. This is represented as the Vertex V_{d1} with the parameters of the SC task with values

(1, 5, m_2) where 1 denotes the degraded budget allocated to the SC task to handle budget overrun of the ACC task, 5 denotes its time period and m_2 denotes the degraded mode 1 of the SC task.

- While the system is in mode m_n or in m_1 and when FCW task overrun its normal budget, the system switches to m_3. This is represented as the Vertex V_{d2} with values (1.5, 5, m_3) where 1.5 denotes the degraded budget allocated to the SC when budget overrun of FCW task occurs, 5 denotes its time period and m_3 denotes the degraded mode 2 of the SC task.

- While the system is in mode m_2 and when FCW task overrun its normal budget or while the system is in mode m_3 and when ACC task overrun its normal budget, SC task is updated with a degraded budget with minimum value among the degraded budgets corresponding to FCW and ACC tasks. This is represented as the Vertex V_{d3} with the parameters of the SC task with values (1, 5, m_4).

- No task is allowed to overrun its degraded budget or its safe budget. So there no need to consider the possibility that the SC task overrun its degraded or safe budgets. Although there can be more number of budget overrun combinations of tasks (typically, 2^n combination for a set of n tasks), only those budget overrun combinations that can lead to change/update in the budget allocated for a task is considered.

Therefore, in a system with n tasks, where each task has the potential to be degraded in n distinct ways, the MS-DRT would require exponential number of vertices and edges to model all possible combination of task overruns which will also have an impact on the complexity of the schedulability analysis.

Observation O5: *Any MCS model should be expressive enough to clearly specify the way to handle budget overrun of multiple tasks and the timing analysis of the MCS model should lead to an acceptable complexity while satisfying observations* **O1** *to* **O4**.

4.3 Context-Aware Mixed Criticality System Model

In this section, we provide some key intuition behind the design choices considered in the proposed model. Then, we discuss how observations **O1** to **O5**, derived in the previous section, are translated to the parameters for the proposed CA-MCS model. In later part of this section, we provide the syntax and semantics of the CA-MCS model.

4.3.1 Mapping Observations to the Model Parameters

4.3.1.1 Mapping overrun tasks and degraded budgets

The term 'criticality' for a task is predominantly used in the academic research. In reality, a designer of an automotive system is given only the ASIL value of the software task. However, an in-depth understanding of the ASIL determination process discussed in ISO 26262 (also discussed in Chapter 2) reveal that value of the ASIL that is actually been assigned to a software task may not be same as the ASIL value of the safety functionality performed by that task. Based on observations **O1**, **O2** and **O4**, we observe that the decision to choose whether a task must be degraded or not should not be based on its criticality when ASIL of a task is used as its criticality. As none of the existing studies related to MCS provide clarity on mapping ASIL of a task to its criticality value, issues can arise on the choice of tasks to be degraded/suspended when a designer wants to use existing MCS task models. Therefore, the need for decoupling task degradation and criticality arises.

In the proposed CA-MCS model, for each overrun task, a designer is given flexibility to specify tasks that needs to be degraded. With this approach, the criticality of a task does not play any role in the choice of tasks to be degraded/suspended. As the choice of tasks to be degraded is independent of their ASIL values, no issues arise when ASIL of a task is considered as its criticality.

From the observation **O3**, it is clear that each task can have multiple degraded budgets. In the CA-MCS model, when two or more tasks overrun their budgets, the choice of the specific degraded budget that needs to be allocated to a degrading task is done based on its criticality level. From this, we can observe that the degradation of tasks and their criticality levels are not completely decoupled. To achieve this, a specific index for each degraded budget can be considered in such a way that the index of each degraded budget of a task corresponds to the index of an overrun task. For example, consider a task τ_i whose budgets can be denoted by a vector $\vec{C_i} = (C_i^1, C_i^2, ..., C_i^k)$. When any task τ_j overruns its budget then C_i^j can be allocated to τ_i to handle system overload. In this way, the higher level of flexibility in the choice of degradation of tasks provided by the CA-MCS model can solve issues linked to the observations derived from the ASIL assurance process and to handle issues related to lack of clarity in mapping ASIL of the task to its criticality level.

4.3.1.2 Multiple budgets

In general, the implementation of a task can be considered to have safety parts and performance parts as shown in the Figure 4.2. In the CA-MCS model, two estimates of a task's execution that includes all

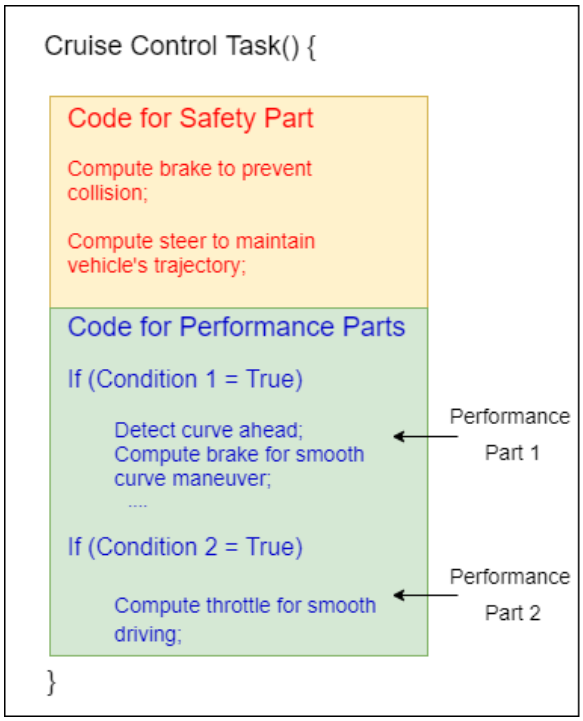

Figure 4.2: Implementation structure of a cruise control task containing a safety part and two different performance parts.

safety and performance parts is considered. They are termed as normal and safe budgets. The level of pessimism involved in measuring task's normal budget is lower than its safe budget. These assumptions regarding budget estimations of a task are similar to the assumptions made in [73].

Task degradation is achieved when one or more performance parts of a task are disabled/excluded from its execution based on some conditions such as detection of budget overrun of specific tasks. Therefore, multiple ways of degrading a task can be achieved by disabling one or more performance parts. For each degraded budget of a task, budget measurements has to be done after identifying the relevant performance parts involved in the task's execution. The difference in the values of the degraded budgets is due to the exclusion of one or more performance parts of the task considered during measurements. For example, the task structure of the cruise control shown in the Figure 4.2 consists of a safety part and 2 performance parts. Hence, degraded budgets are the measurements done for the task execution that includes all safety parts but only some selected performance parts.

It is important to note that the safety goals such as computation of correct brake, steer and throttle values are achieved by tasks even with their degraded budgets. Here, the notion of task degradation

represents only the reduction in the quality (accuracy or resolution) of the output and not on the reduction of the core safety functionality itself. Hence, the possibility of multiple degraded budgets enhances the flexibility in achieving the desired quality of the output without compromising on the safety goals needed to be achieved by tasks. Therefore, the CA-MCS task model presented in this thesis require three types of budget to be measured for each task. They are the task's normal budget, safe budget and degraded budgets. At worst-case, for a system with n tasks, for any task τ_i, we can assume at most $n-1$ degraded budgets corresponding to the budget overrun of the remaining $n-1$ tasks (excluding τ_i).

4.3.1.3 Measurement of budgets

The system designer of MCS can consider two different methods for estimating these budgets. They are the following:

1. Measurement based analysis [74].
2. Static methods [75].

Measurement based analysis involves measuring execution times of code segments on a real hardware or on a simulator for some set of inputs. Measurements are taken using devices like logic analyzer or by using a processor clock. Even if this method seems to be the more practical approach, it is tough to capture all possible paths that a task can take when executing through the code. Due to the complicated behavior of hardware and software, some of the actual long execution time for the task could be missed. Therefore, a "safety margin" is added with the observed longest execution time, for example, adding 20% to the observed longest execution time is considered.

With static analysis methods it is possible to identify the worst theoretically possible longest execution time for the task without executing the code on the hardware. This value can be considered as an upper bound Worst Case Execution Time (WCET). Therefore, this method requires a precisely accurate model of the timing characteristics of the processor such as the behavior of pipelines, caches, memory, buses, and any other feature of the hardware under examination that may affect execution time of machine instructions. The accuracy of the estimation of execution time depends on the quality of the model. In this thesis, the following assumptions are made to determine the values of different budgets:

1. All budgets (normal, safe and degraded) for tasks are estimated using measurement based analysis.

2. The longest execution time estimated using the measurement based analysis for the task execution that includes all safety and performance parts is considered as the value of the task's normal budget.

3. We assume that a safe budget of a task is its known upper bound execution time. Hence, an addition of a "safety margin" of 20% to the estimated longest execution time (normal budget) for the task execution that includes all safety and performance parts is considered as the value of the task's safe budget. The safety margin can be decided by a designer based on the ASIL value of the task *i.e.* higher safety margin can be considered for tasks with higher ASIL value.

4. As the amount of pessimism involved in the estimation of a task's safe budget is higher than its normal budget, we assume that the normal budget of a task is lower than or equal to its safe budget.

5. An addition of a "safety margin" of 20% to the estimated longest execution time for the task execution that includes all safety but selected performance parts is considered as the value of the task's degraded budget. Since a task can be degraded in multiple ways, for each degraded budget, budget estimation is done after identifying the relevant performance parts involved in the task's execution.

6. For simplicity of the analysis, we assume that the degraded budget of a task is always lower or equal to its normal budget.

7. It is assumed that the values of the estimated safe and degraded budgets with higher levels of pessimism are always sufficient for the task to finish executing its safety and performance parts. Therefore, in the proposed model, the task is allowed to overrun only its normal budget and not its safe or degraded budgets.

If static methods are employed then the worst theoretically possible longest execution time identified for the task execution that includes all safety and performance parts can be considered as the value of safe budget. Similarly, the worst theoretically possible longest execution time identified for each type of task degradation can be considered as the value of the degraded budget.

4.3.1.4 Task behaviour

With respect to MCS task models, guaranteeing ASIL requirements means that occurrence of any timing faults of lower ASIL task such as its budget overrun should not prevent higher ASIL tasks from achieving their intended safety goals. It is possible that the reduced budget that is allocated to a task instantaneously upon the detection of budget overrun of tasks can become insufficient to achieve its safety goal. Therefore, to guarantee ASIL requirements, the semantics of the model can be designed in such a way that upon the detection of budget overrun of tasks, among those tasks that are chosen for degradation, only tasks with relatively lower or equal ASIL (criticality) value than the overrun task are degraded instantaneously. Further, all active jobs of higher ASIL tasks are allowed to finish their current execution and only their subsequent releases are updated with their degraded budgets. With this approach, it can be ensured that the timing faults of lower ASIL tasks do not affect higher ASIL tasks.

In order to determine the budget allocation of tasks at run-time to guarantee ASIL requirements, it is necessary to know two important information regarding task's budget. They are the following:

1. To know whether the budget of a task is intended to achieve its safety with good performance or with degraded performance.

2. To know whether a task can be allowed to overrun its allocated budget or not as it is necessary to prevent task overrunning its budgets that are estimated at higher level of pessimism.

Therefore, to capture above two information, models proposed in this thesis require a designer to specify 'behaviour' of a task's budget as an additional parameter. This information regarding task's budgets must be used at run-time to decide whether to allow the task to overrun its allocated budget. For example, consider a task τ_i allocated with a budget of 3 units. If this budget is estimated at lower levels of pessimism, then it can be allowed to execute beyond 3 units. When this happens, it can be immediately allocated with the budget estimated at higher levels of pessimism usually with values greater than 3 units.

4.3.1.5 Handling budget overrun of multiple tasks

In the proposed CA-MCS model, degradation of tasks is considered to handle system overload due to budget overrun of tasks. Based on the observation **O4**, the choice of the degraded budget for a degrading task should depend on overrun tasks. With a single overrun task, straight forward mapping of the specific budget of the degrading task to the overrun task is useful. However, when two or more tasks overrun their budgets, the choice of the budget for degrading tasks should be based on a specific rule. With CA-MCS, this rule is motivated based on the following intuition:

1. When multiple tasks are degraded or suspended simultaneously, it is possible that one such combination may impact safety of the system. For example, degradation/suspension of either radar or V2V (Vehicle to Vehicle) sensor task will not impact safety of the system. However, degradation/suspension of both radar and V2V sensor tasks simultaneously may lead to non-availability of sufficient redundancy of the system and therefore can possibly lead to catastrophic consequences. In the proposed CA-MCS model, to reduce the burden on a designer to verify the safety of all possible task degradation combinations, when two or more overrun tasks are of different criticality, only the degraded budget corresponding to overrun tasks with highest criticality is allocated as budget for a task. This will greatly reduce the burden for a designer because it is enough to ensure that all possible combinations of budget overrun and degradation of tasks at a specific criticality level will not impact safety of the system. As the safety requirements are more stringent for higher criticality tasks [14], it is also sensible to ensure that degradation corresponding to higher criticality overrun tasks are always honoured.

2. When two are more overrun tasks have the same highest criticality, it is sensible to consider the degraded budget with minimum value among degraded budgets corresponding to overrun tasks with the highest criticality. This will greatly help in handling increased system overload due to the budget overrun of multiple tasks. Again, this will not impact safety of the system as it is assumed that designer has already ensured that all possible combinations of budget overrun and degradation of tasks at a specific criticality level will not impact the safety of the system.

In the proposed CA-MCS model, the concept of system's state is used to keep track of the the highest criticality among all overrun tasks. At any given time, the system is considered to execute in a specific state. For example, when a system executes in a safe state S_l which is the system state at criticality level l, it means that the highest criticality among all overrun tasks is l. When any task with criticality greater than $l + 1$ overruns its normal budget, then system transition to S_{l+1} is done. With this information, it

is possible to update budget for tasks when any task is found to overrun its normal budget.

4.3.2 Basic Definitions

Definition 8 ***Budget Overrun (BO)*** *of a task τ_i is considered to have occurred when τ_i, assigned with a budget C_i, does not signal its completion even after executing for C_i time units.*

Definition 9 ***Behaviour*** *of a task depends on the amount of execution time consumed by a task to achieve its intended functionality. A task's behaviour is considered to be* ***Normal*** *when it can guarantee both safety and expected performance with its normal budget. A task's behaviour is considered to be* ***Safe*** *when it can guarantee both safety and expected performance only with its safe budget. A task's behaviour is considered to be* ***Degraded*** *when it can guarantee only safety with lower than expected performance using its degraded budget.*

4.3.3 Syntax of the CA-MCS model

A Context-Aware Mixed Criticality System, represented as $S \stackrel{\text{def}}{=} (\Gamma, \mathcal{L})$, where Γ represents a set of n tasks denoted as $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$. $\mathcal{L} = \{1, 2, .., L\}$ denotes an ordered set of criticality levels with a total order supported by S. L and 1 denote the highest and the lowest criticality levels in S respectively. Any task $\tau_i \in \Gamma$ generates an infinite sequence of jobs and can be specified by a tuple $\tau_i = \{T_i, \vec{B}_i, C_i^n, C_i^s, \vec{C}_i, \mathcal{L}_i\}$. T_i (minimum arrival interval) denotes the time period of τ_i. C_i^n denotes the normal budget of τ_i. C_i^s denotes the safe budget of τ_i. $\vec{C}_i = (C_i^1, C_i^2, ..., C_i^k)$ denotes a vector of degraded budgets of τ_i. $\vec{B}_i = (B_i^1, B_i^2, ..., B_i^k)$ denotes a vector of behaviours of τ_i. Each $B_i^j \in \{Normal, Safe, Degraded\}$ denotes the behaviour of τ_i when $\tau_j \in \Gamma$ overruns its normal budget C_j^n. Each B_i^j is associated with a specific budget C_i^j such that τ_i is assigned with C_i^j when $\tau_j \in \Gamma$ overruns its normal budget. $\mathcal{L}_i \in \mathcal{L}$ denotes the criticality level of τ_i. Following assumptions are made regarding budgets of τ_i.

1. $C_i^j \leq C_i^n$, if $B_i^j = Degraded$.
2. τ_i is allowed to overrun only its normal budget (C_i^n).
3. $C_i^n \leq C_i^s$.

4.3.4 Semantics of the CA-MCS model

At any time $t \geq 0$, depending on the run-time behaviour of tasks, S can exist in any state in $\mathcal{ST} = \{S_0, S_1, S_2, ..., S_L\}$ where L denotes the total number of criticality levels in S. S_0 denotes normal state and $S_1, S_2, ..., S_L$ denote safe states. System starts (at t = 0) in S_0.

Conditions to determine system's state:

Let t_x be the current time instant and let $t_{idle} < t_x$ be the latest time instant at which system experienced an idle instant. Let e_i^{max} denote the largest execution time of any job of τ_i at t_x since t_{idle}. S is said to be in the normal state S_0 at t_x, if the condition $\forall \tau_i \in \Gamma$, its $e_i^{max} \leq C_i^n$ is satisfied. In other words, S is said to be in the normal state S_0 at t_x if no tasks have overrun their normal budget in the interval $t_{idle} < t \leq t_x$.

Let $l = max(\mathcal{L}_i), \forall \tau_i \in \Gamma_l$ where Γ_l denotes a set of tasks that satisfy the condition $C_i^n < e_i^{max} \leq C_i^s, \forall \tau_i \in \Gamma$ at t_x. S is said to be in the safe state $S_l \in \mathcal{ST} \setminus S_0$ at t_x, if $\Gamma_l \neq \emptyset$ at t_x. A state transition in S can happen from S_m to S_n only if $n > m$ in the interval $t_{idle} < t \leq t_x$.

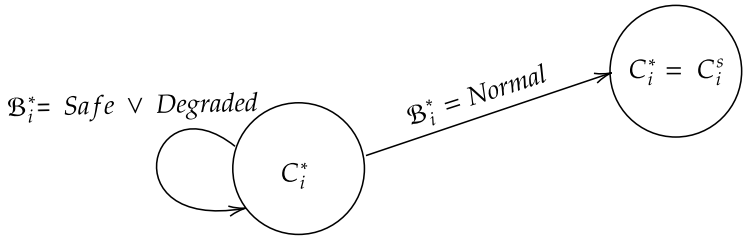

Figure 4.3: Change in the value of C_i^* for different values of B_i^* when a job of τ_i has executed for C_i^* without signalling its completion at t_x.

Transition from one state to another state:

Let the current time instant be t_x and the current system state be $S_l \in \mathcal{ST}$. $\forall \tau_i \in \Gamma$, let C_i^* denote the current budget allocated to τ_i and let B_i^* denote the current behaviour of τ_i. Let e_i^* denote the time for which an active job of τ_i has executed till t_x. Let J_i denote a job of τ_i that has executed for C_i^* without signalling its completion (budget overrun) at t_x. The current budget (C_j^*) allocated for τ_i at t_x depends on the value of B_i^* as shown in Figure 4.3. Depending on on the value of B_i^* at t_x, C_i^* is updated as the following:

1. If B_i^* = $Normal$, the current budget of τ_i is immediately updated as C_i^s *i.e.* C_i^* is immediately updated as C_i^* = C_i^s. B_i^* is immediately updated as $B_i^* = Safe$.

2. If $B_i^* = Safe \vee Degraded$, J_i is terminated immediately without affecting budgets of other tasks.

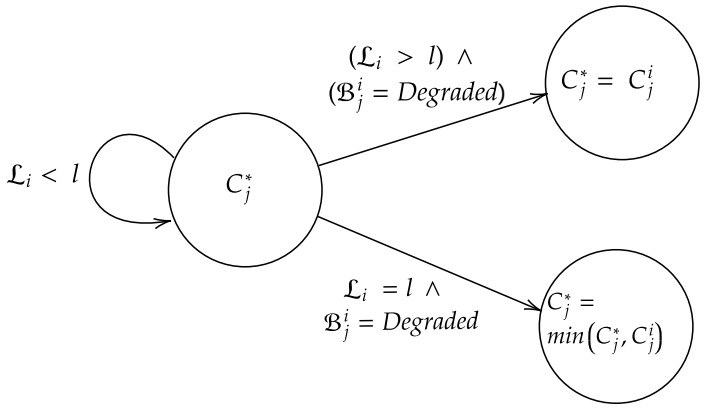

Figure 4.4: Change in the value of C_j^* for different values of $\mathcal{L}_i$ and B_j^i when a job of τ_i has executed for C_i^* without signalling its completion at t_x.

$\forall \tau_j \in \Gamma \setminus \tau_i$, the budget allocated for τ_j depends on the value of $\mathcal{L}_i$ as shown in Figure 4.4. Hence, the following three cases are considered.

Case 1: $\mathcal{L}_i = l$

$\forall \tau_j \in \Gamma \setminus \tau_i$, if $B_j^i = Degraded$ then three cases are possible depending on the value of e_j^* :

1a. If $\mathcal{L}_j \leq \mathcal{L}_i$ and $e_j^* \geq C_j^i$, τ_j is assumed to be finished. The subsequent releases of τ_j are executed with $\min(C_j^*, C_j^i)$.

1b. If $\mathcal{L}_j \leq \mathcal{L}_i$ and $e_j^* < C_j^i$, then C_j^* is immediately updated as $C_j^* = \min(C_j^*, C_j^i)$.

1c. If $\mathcal{L}_j > \mathcal{L}_i$, only the subsequent releases of τ_j are executed with $\min(C_j^*, C_i^j)$.

Case 2: $\mathcal{L}_i > l$

A state transition from $\mathcal{S}_l$ to $\mathcal{S}_{\mathcal{L}_i}$ is immediately done. If $B_j^i = Degraded$ then three cases are possible depending on the value of e_j^* :

Symbols	Meaning
S	a context-aware mixed criticality system.
Γ	a set of tasks in S.
$\mathcal{ST}$	a set of system states.
t_{idle}	time instant at which S is idle.
$\mathcal{L}$	a set of criticality levels supported by S.
τ_i	a task in S.
$\mathcal{L}_i$	criticality of τ_i.
T_i	time period of τ_i.
$\vec{B_i}$	a vector of behaviours of τ_i.
C_i^n	normal budget of τ_i .
C_i^s	safe budget of τ_i.
$\vec{C_i}$	a vector of degraded budgets of τ_i.
C_i^j	a degraded budget assigned to τ_i when $\tau_j \in \Gamma \setminus \tau_i$ overruns its normal budget.
t_{idle}	time instant at which S is idle.

Table 4.2: Symbols and their meanings used in the CA-MCS model.

2a. If $\mathcal{L}_j \leq \mathcal{L}_i$ and $e_j^* \geq C_j^i$, τ_j is assumed to be finished.

2b. If $\mathcal{L}_j \leq \mathcal{L}_i$ and $e_j^* < C_j^i$, C_j^* is immediately updated as $C_j^* = C_j^i$.

2c. If $\mathcal{L}_j > \mathcal{L}_i$, only the subsequent releases of τ_j are executed with C_j^j.

Case 3: If $\mathcal{L}_i < l$, no other tasks are affected.

The system returns to the normal state at the idle instant.

As the proposed CA-MCS model considers many parameters, in Table 4.2, list of symbols and their meanings used in syntax and semantics of the CA-MCS model are tabulated for the reference purpose.

4.3.5 An Example to Illustrate the CA-MCS Model

To bring more clarity, an example taskset and its schedule is discussed. Let us consider a system S with 4 tasks (τ_1 to τ_4). The details of parameters of tasks are given in Table 4.3 and their schedule is shown in Figure 4.5. $\mathcal{L}_i$ denotes the criticality of τ_i. P_i denotes the priority of τ_i. The Fixed Priority Pre-emptive Scheduling (FPPS) scheme is considered for scheduling tasks. The system S starts at $t = 0$. When τ_2, with criticality level equal to 1 overruns its normal budget at $t = 1.0s$, system undergoes a

Task τ_i	**Criticality** $\mathcal{L}_i$	**Priority** P_i	**Time Period** T_i	**Budgets**	**Behaviour**
τ_1	2	2	5	$C_1^n = 0.4, C_1^s = 0.8$	$B_1^1 = Safe$ $B_1^2 = Normal$ $B_1^3 = Normal$ $B_1^4 = Normal$
τ_2	1	1	10	$C_2^n = 1.0, C_2^s = 1.4$ $C_2^1 = 0.15, C_2^3 = 0.9$	$B_2^1 = Degraded$ $B_2^2 = Safe$ $B_2^3 = Degraded$ $B_2^4 = Normal$
τ_3	2	3	25	$C_3^n = 0.6, C_3^s = 7.4$	$B_3^1 = Normal$ $B_3^2 = Normal$ $B_3^3 = Safe$ $B_3^4 = Normal$
τ_4	1	4	49	$C_4^n = 3.4, C_4^s = 9.6$ $C_4^1 = 2.7, C_4^2 = 2.1$ $C_4^3 = 2.8$	$B_4^1 = Degraded$ $B_4^2 = Degraded$ $B_4^3 = Degraded$ $B_4^4 = Safe$

Table 4.3: Taskset example.

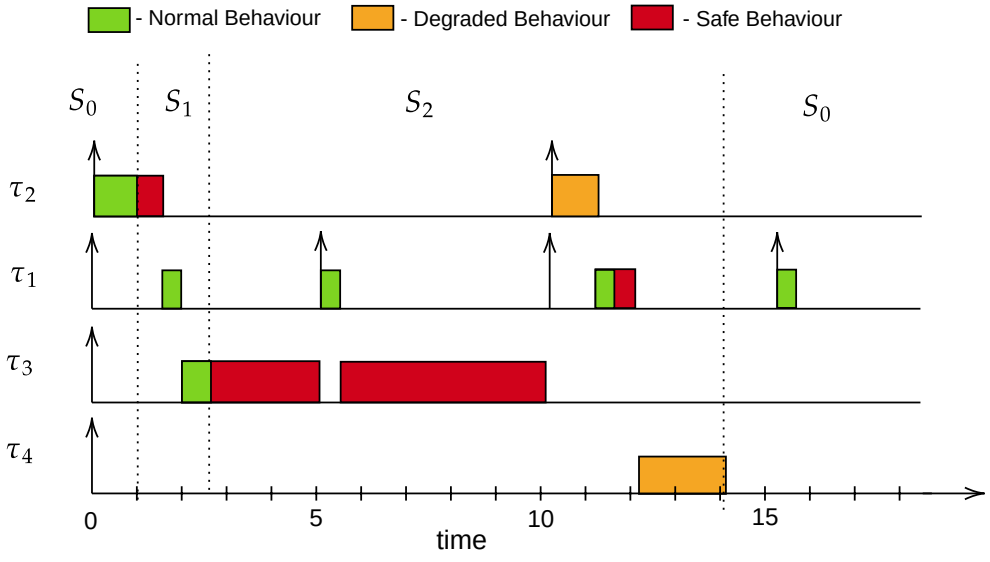

Figure 4.5: An example schedule.

transition from the normal state S_0 to the safe state S_1. As a result, task τ_4 is assigned with its degraded budget corresponding to τ_1 *i.e.* τ_4 is assigned with C_4^1 (= 2.1). By choosing the degraded budget corresponding to the overrun task τ_2, the observation **O4** is satisfied.

When the task τ_3, with criticality level equal to 2, overruns its normal budget at $t = 2.4s$, system undergoes a transition from S_1 to S_2. As a result, tasks τ_2 and τ_4 are assigned with their degraded budgets corresponding to τ_3 *i.e.* τ_4 is assigned with C_4^3 (= 2.8) and τ_2 is assigned with C_2^3 (= 0.9). An important assumption made in this step is that when multiple tasks have overrun their normal budgets, degraded budget corresponding to the highest criticality task is considered. In this case, both τ_1 and τ_3 has overrun their normal budgets. But, τ_4 is allocated with degraded budget corresponding to τ_3 since criticality of τ_3 is higher than τ_1. By doing this, the model satisfies both **O4** and **O5** and advantages of this assumption are already discussed in subsection 4.3.1.5.

When τ_1, with criticality level also equal to 2 overruns its normal budget at $t = 11.3s$, system continues to execute in the state S_2 but τ_4 undergoes further degradation and is allocated with the budget of 2.1 which is the minimum budget among its degraded budgets corresponding to τ_1 and τ_3 *i.e.* min(C_4^1, C_4^3) = 2.1. The degraded budget allocated to τ_2 (currently 0.15) remains unchanged since min(C_2^1, C_2^3) = 0.15. Again, as discussed in subsection 4.3.1.5, with this approach, the model can efficiently handle budget overrun of multiple tasks to satisfy **O5** by avoiding higher level of complexity to model all possible combination of tasks overruns within a system state.

System transition to the normal state S_0 happens when the system experiences the idle instant at $t = 13.8$.

4.4 Schedulability Analysis

In this section, we focus on the Fixed Priority Pre-emptive Scheduling scheme for the CA-MCS model comprising a static set of n tasks with implicit deadline (*i.e.* deadline equal to period) which execute on a single processor. The only resource that is shared among the tasks is the processor. It is assumed that the scheduler overheads during task preemption such as the computation time required to activate a task, schedule the tasks, and terminate a task are included in the value of budget for tasks. Existing studies such as [76] discuss techniques to reduce the number of preemptions of tasks. These are not considered in the proposed framework and will be considered as the potential future work. The measurements of execution time for tasks on the experimental setup were arrived after performing simulations for large number of times for different driving scenarios (different vehicle speeds, straight

and curved roads etc.). Hence the measurement-based estimation for budget for tasks includes the cost due to scheduler, preemption delays of tasks that occurred during the simulation. It is assumed each task can generate a sequence of infinite number of jobs such that jobs generated by a task are separated by a specific minimum arrival interval. The system is defined for a normal state and L safe states of operation. The taskset is assumed to be fully defined and for each task, their ids, priorities and indices of their budgets are assigned apriori. The worst-case response time of a task τ_i, denoted by R_i, gives the maximum possible time that any job of τ_i can take to complete. A task is schedulable if it's R_i does not exceed its deadline.

Since the system might experience one or more state transitions when a job of τ_i is active, all possible combinations of state transitions have to be considered to calculate R_i. R_i for τ_i is the maximum response time among response times computed for all possible valid state transition pattern.

Definition 10 *A **valid transition pattern** is a monotonic increasing sequence of time instants denoted as $P = [t_1, t_2, ..., t_L]$ where t_k denotes the instance at which the system transition to the state S_k happens.*

Let $\mathcal{P} = \{P^1, P^2, ..P^m\}$ denote a set of predefined valid state transition patterns that can occur in the interval $[0, T_i]$. $P^k = [t_1, t_2, ..., t_L]$ denotes an array of state transition time instances such that $0 \leq t_1 \leq t_2 \leq ... \leq t_L$. Let t_0 represent the system start time and hence its value can be considered to be zero *i.e.* $t_0 = 0$. In our analysis, we assume that for any $k > 0$, for given predefined valid state transition pattern, when t_l and t_{l+k} occur at same time instance, only t_{l+k} is considered to have occurred. This assumption is valid for a uni-processor system since at any given time instant there can be only one active task execution. In the following, various steps considered to compute worst-case response time of an task τ_i are discussed.

4.4.1 Steps to Compute the Worst-Case Response Time of a Task

Step 1: As a first step, release patterns of tasks that maximizes the interference for τ_i for any given state transition pattern $P^l \in \mathcal{P}$ are found. This is required to compute the maximum number of jobs from each interfering task that can interfere τ_i. Theorems 1 and 2 presented in subsection 4.4.2 discuss such release patterns under normal and budget overrun conditions.

Step 2: Due to the complexity of the CA-MCS model, the analysis is performed in a structured approach in which for each interfering task τ_j, time instances of a specific valid transition pattern $P^l \in \mathcal{P}$ are separated into 3 sets of time intervals based on the τ_j's behaviour (normal, safe and degraded). Interference

is analyzed by abstracting the influences from other time intervals of the different states. Depending on the safe state considered, jobs of an interfering task can interfere with their normal, safe or degraded budgets. For the release patterns discussed in Theorems 3 and 4, the maximum number of jobs that can interfere with normal, safe and degraded budgets need to be found to compute the interference from interfering tasks. For this purpose, equations 4.1, 4.2 and 4.3 are presented in subsection 4.4.2.

Step 3: From the semantics of the CA-MCS model, it is clear that a task can be allocated with any one of its degraded budgets depending on the overrun tasks. Hence, for the WCRT analysis for τ_i, it is required to determine the maximum possible degraded budget that can be allocated to an interfering task τ_j in each safe state. In the following, we present definitions for terms used to find the maximum value of the degraded budget that can be allocated to an interfering task τ_j in each safe state.

Definition 11 *For a task $\tau_j \in \Gamma$,*

$$DH_j = \begin{cases} 1, & \forall \tau_k \in \Gamma \setminus \tau_j, (\mathcal{L}_k > \mathcal{L}_j) \wedge B_j^k = Degraded \\ 0, & Otherwise \end{cases}$$

In the above expression, the value of DH_j is set to 1 if the behaviour of τ_j is such that it is always degraded when any task with criticality level greater than $\mathcal{L}_j$ overruns its normal budget. Otherwise, DH_j is considered to be 0.

Definition 12 *For a task $\tau_j \in \Gamma$,*

$$DL_j = \begin{cases} 1, & \forall \tau_k \in \Gamma \setminus \tau_j, (\mathcal{L}_k < \mathcal{L}_j) \wedge B_j^k = Degraded \\ 0, & Otherwise \end{cases}$$

Similarly, the value of DL_j is set to 1 if the behaviour of τ_j is such that it is always degraded when any task with criticality level lower than $\mathcal{L}_j$ overruns its normal budget. Otherwise, DL_j is considered to be 0.

The values of DH_j and DL_j are derived from B_j and can be used to determine the maximum budget with which τ_j can interfere in any given safe state (except $S_{\mathcal{L}_j}$). In the safe state $S_{\mathcal{L}_j}$, τ_j can interfere with its safe budget (C_j^s). This is possible when the system undergoes state transition to $S_{\mathcal{L}_j}$ due to the budget overrun of τ_j which can lead to degradation of other tasks with criticality level same as that of τ_j. As the semantics of the CA-MCS model does not allow tasks to overrun their degraded budget, τ_j can continue to execute with its safe budget in $S_{\mathcal{L}_j}$ without getting degraded. Hence, expressions for

computing DL_j and DH_j do not consider tasks with criticality level same as that of τ_j.

Definition 13 *$maxB(\tau_j, l)$ returns the maximum budget that can possibly be assigned to τ_j in the state S_l due to the budget overrun of any task $\tau_k \in \Gamma \setminus \tau_j$ with $\mathcal{L}_k = l$.*

$$maxB(\tau_j, l) = \begin{cases} C_j^n, & (l = 0) \vee ((l < \mathcal{L}_j) \wedge (DL_j = 0)) \\ C_j^s, & (l = \mathcal{L}_j) \vee ((l > \mathcal{L}_j) \wedge (DH_j = 0)) \\ \max\limits_{\substack{\forall \tau_k \in \Gamma \setminus \tau_j, \\ \mathcal{L}_k = l}} C_j^k, & Otherwise \end{cases}$$

In Definition 13, if $DH_j = 0$, τ_j is always assigned with its safe budget in any safe state S_l for any l in the range $\mathcal{L}_j \leq l \leq L$. Otherwise, it is assigned with safe budget only in $S_{\mathcal{L}_j}$ and with degraded budgets in any safe state S_l for any l in the range $\mathcal{L}_j < l \leq L$. Similarly, if $DL_j = 1$, τ_j is always assigned with its degraded budget in any safe state S_l for l in the range $0 < l < \mathcal{L}_j$. Otherwise if $DL_j = 0$, τ_j is always assigned with its normal budget in any safe state S_l for any l in the range $0 \leq l < \mathcal{L}_j$ including the normal state S_0.

Step 4: In the next step, the worst-case response time for τ_i considering interference from jobs of τ_j for a given $P^l \in \mathcal{P}$ is computed using Equation 5.7. This value is denoted as R^l. Subsequently, the maximum value among R^l computed for each $P^l \in \mathcal{P}$ is denoted as R_i can be found using Equation 4.5.

In the following, we provide corollaries and theorems that are used to derive the equations for fixed-priority based sufficient schedulability test for tasks on pre-emptive uniprocessors for the CA-MCS model.

4.4.2 Worst-Case Scenarios

In the following theorems, it is shown that the release pattern of any higher priority task τ_j that can maximize its interference to τ_i depends on the valid state transition pattern considered, values of DH_j, DL_j and on the length of the time interval considered. The following theorem considers the conditions that can maximize the interference from τ_j when both τ_i and τ_j are requested simultaneously.

Theorem 1 *For any given valid state transition pattern, for any $t > 0$, interference in interval [0, t] for a job of τ_i from any higher priority task τ_j is maximized when τ_i is requested simultaneously with the request for τ_j under any of the following conditions,*

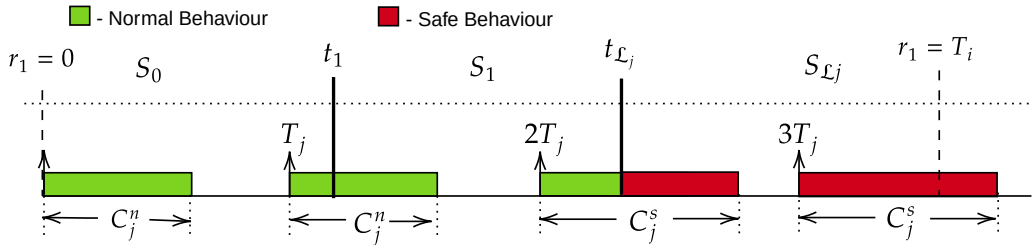

Figure 4.6: Release pattern for τ_j with $DL_j = 0$.

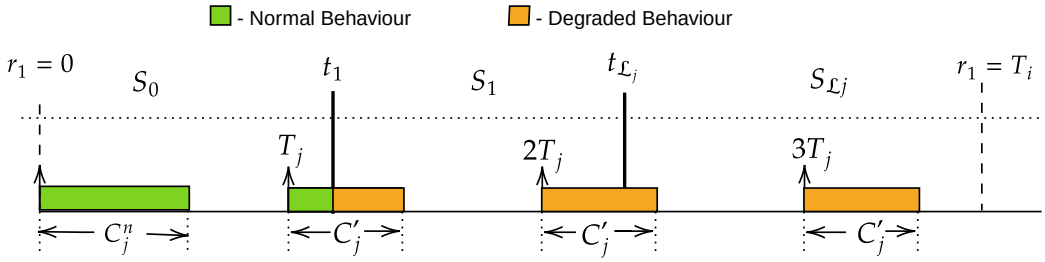

Figure 4.7: Release pattern for τ_j with $DL_j = 1$.

- *Condition A1: for any* $0 < t \leq t_{\mathcal{L}_j}$
- *Condition A2: for any* $t > t_{\mathcal{L}_j}$*, and if* $DH_j = DL_j = 1$ *and if* $0 \leq t_k < t_{\mathcal{L}_j}$ *for any* k *in the range* $1 \leq k < \mathcal{L}_j$.

Proof: Let $\tau_1, \tau_2, ..., \tau_i$ denote a set of priority-ordered tasks with τ_i being the task with the lowest priority. Consider a particular request for τ_i that occurs at $r_1 = 0$. Suppose that between r_1 and $r_1 + T_i$, requests for the task τ_j with priority higher than τ_i, occur at 0, T_j, $2T_j$,..., kT_j, as illustrated in Figure 4.6 and Figure 4.7.

Condition A1: $0 < t \leq t_{\mathcal{L}_j}$

For $0 < t \leq t_{\mathcal{L}_j}$, as per the semantics of the CA-MCS model, no release of τ_j can execute beyond its normal budget in the interval [0, t]. Further, in this interval, depending on the value of DL_j, τ_j can execute either with its normal budget or with one of its degraded budgets. From Definition 12, it is clear that if $DL_j = 0$, then releases of τ_j can execute with their normal budget in the interval [0, t]. By advancing (shifting left) the release pattern of τ_j by ϵ, interference from τ_j can decrease utmost by ϵ as execution of τ_j gets excluded in the interval [0, t]. It is also possible that the interference from τ_j will remain unchanged in the interval [0, t] when outbound interference due to advancement of the release

pattern is replaced by the inbound interference which can occur for ϵ value in the range $0 < \epsilon \leq C_j^n$. Similarly, by delaying (shifting right) the release pattern by ϵ interference from τ_j either gets decreased by ϵ or will remain unchanged for $0 < \epsilon < (T_j - C_j^n)$.

From Definition 12, if $DL_j = 1$, then releases of τ_j can execute with their normal budget only in the interval $[0, t_1]$ and with degraded budgets in the interval $(t_1, t]$. From the syntax of the CA-MCS model, it is clear that degraded budgets of τ_j can be equal or lower than its normal budget. At worst-case, degraded budgets of τ_j can be assumed to be equal to its normal budget. Then, by advancing the release pattern of τ_j by ϵ, interference from τ_j can decrease utmost by ϵ as the execution of τ_j gets excluded in the interval $[0, t]$. It is also possible that the interference from τ_j will remain unchanged in the interval $[0, t]$ when outbound interference due to advancement of the release pattern is replaced by the inbound interference which can occur for ϵ value in the range $0 < \epsilon \leq C_j^n$. Similarly, by delaying (shifting right) the release pattern by ϵ, interference from τ_j either gets decreased by ϵ or will remain unchanged for $0 < \epsilon < (T_j - C_j^n)$.

Condition A2: for any $t > t_{\mathcal{L}_j}$, if $DH_j = DL_j = 1$ and for any k in the range $1 \leq k < \mathcal{L}_j$, $0 \leq t_k < t_{\mathcal{L}_j}$

If it is the case that there is a time instant t_k such that $0 \leq t_k < t_{\mathcal{L}_j}$ for any k in the range $1 \leq k < \mathcal{L}_j$, and if $DL_j = 1$, then τ_j will be degraded before $t_{\mathcal{L}_j}$. As the semantics of CA-MCS does not allow τ_j to overrun its degraded budget, it is not possible for τ_j to execute with its safe or normal budget beyond t_k. If $t_k > 0$, then it is possible that for $k = 1$, τ_j can execute with its normal budget in the interval $[0, t_1]$ and with degraded budget in the interval $(t_1, t]$. Otherwise, if $t_k = 0$, then τ_j can interfere only with any of its degraded budget in the interval $[0, t]$. In either case, by delaying the release pattern of τ_j by ϵ, interference from τ_j can either remain unchanged or decreased. Interference can decrease utmost by ϵ as some execution of τ_j gets excluded in the interval $[0, t]$ or the interference from τ_j will remain unchanged for $\epsilon \leq (T_j - C_j^s)$ beyond which definitely some execution of τ_j can be excluded in the interval $[0, t]$. Advancing the release pattern of τ_j by ϵ, interference from τ_j can decrease utmost by ϵ as execution of τ_j gets excluded in the interval $[0, t]$. It is also possible that the interference from τ_j will remain unchanged in the interval $[0, t]$ when outbound interference due to advancement of the release pattern is replaced by the inbound interference which can occur for ϵ value in the range $0 < \epsilon \leq C_j'$. The above arguments are valid when the degraded budgets of τ_j are less than or equal to its normal budget which also an important assumption made in the syntax of the CA-MCS model. ■

The following theorem considers the release pattern of τ_j that can maximize the interference from τ_j to τ_i when τ_j overruns its normal budget.

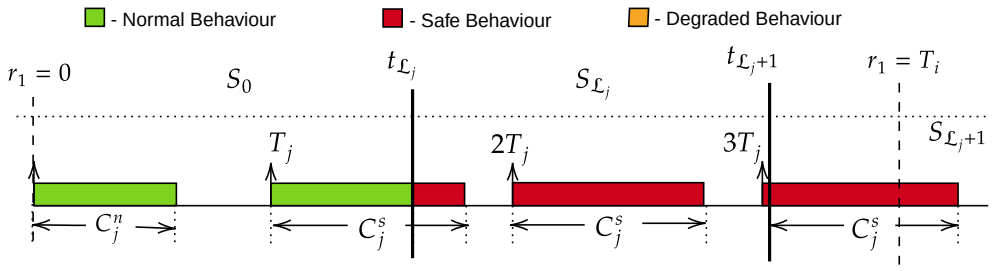

Figure 4.8: Release pattern of τ_j with $DH_j = 0$.

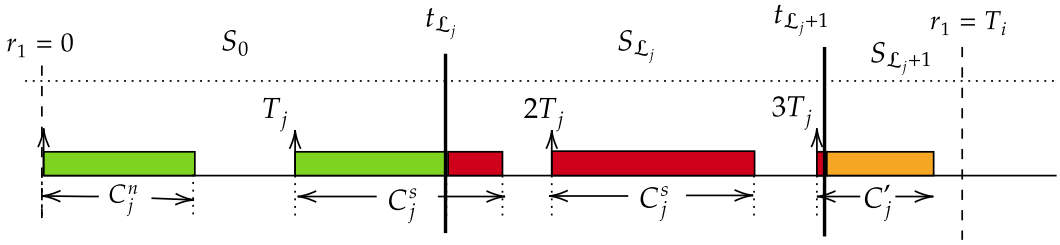

Figure 4.9: Release pattern of τ_j with $DH_j = 1$.

Theorem 2 *For any given valid state transition pattern, for any $t > t_{\mathcal{L}_j}$, the interference in interval [0, t] for a job of τ_i from any higher priority task τ_j is maximized when the budget overrun of τ_j occurs at $t_{\mathcal{L}_j}$ under any of the following conditions,*

- *Condition A3: $t > t_{\mathcal{L}_j}$ and $DH_j = 0$*
- *Condition A4: $t > t_{\mathcal{L}_j}$ and $DH_j = 1$ and $DL_j = 0$*
- *Condition A5: $t > t_{\mathcal{L}_j}$ and if $DH_j = DL_j = 1$ and if $t_k = t_{\mathcal{L}_j}$, $\forall k$ in the range $1 \leq k < \mathcal{L}_j$*

Proof:

Let $\tau_1, \tau_2, ..., \tau_i$ denote a set of priority-ordered tasks with τ_i being the task with the lowest priority. Consider a particular request for τ_i that occurs at $r_1 = 0$. Suppose that between r_1 and $r_1 + T_i$, requests for the task τ_j with priority higher than τ_i, occur at 0, T_j, $2T_j$,..., kT_j, as illustrated in Figure 4.8 and Figure 4.9. Clearly, the pre-emption of τ_i by τ_j, will cause a certain amount of delay in the completion of the request for τ_i that occurred at r_1.

Condition A3: $t > t_{\mathcal{L}_j}$ and $DH_j = 0$

From Definition 11, it is clear that if $DH_j = 0$, all releases of τ_j that are active at or after $t_{\mathcal{L}_j}$ can execute with their safe budget. By advancing the release pattern of τ_j by ϵ, interference from τ_j in the interval [0, t] is decreased as the number of jobs interfering with safe budgets is now reduced since the release of τ_j that was previously assumed to overrun its normal budget at $t_{\mathcal{L}_j}$ will finish its execution before $t_{\mathcal{L}_j}$ with normal budget. By delaying the release pattern by ϵ, interference either remains unchanged or decreased. Interference from τ_j will remain unchanged for $\epsilon < T_j$ beyond which number of releases of τ_j interfering with safe budget decreases leading to decrease in the interference from τ_j in the interval [0, t].

Condition A4: $t > t_{\mathcal{L}_j}$ and $DH_j = 1$ and $DL_j = 0$

From Definition 11, we note that if $DH_j = 1$, releases of τ_j can execute with their safe budget only in the interval $[t_{\mathcal{L}_j}, t_{\mathcal{L}_{j+1}}]$ and with their degraded budget in the interval ($t_{\mathcal{L}_{j+1}}$, t]. This scenario is shown in Figure 4.9.

By delaying the release pattern by ϵ, some instances of τ_j that were previously interfering with safe budgets in the interval $[t_{\mathcal{L}_j}, t_{\mathcal{L}_{j+1}}]$ can be pushed beyond $t_{\mathcal{L}_{j+1}}$. These releases of τ_j can now interfere only with degraded budget leading to decrease in interference in the interval [0, t]. However, the number of releases of τ_j interfering with safe budgets will remain unchanged in the interval [0, t] for $0 < \epsilon < T_j$ since the release of τ_j that was considered to overrun its normal budget at $t_{\mathcal{L}_j}$ can now overrun its normal budget at any time greater than $t_{\mathcal{L}_j}$. Therefore, interference from τ_j will remain unchanged in the interval [0, t] for $\epsilon < T_j$.

By advancing the release pattern by ϵ, release of τ_j that was previously considered to overrun its normal budget at $t_{\mathcal{L}_j}$, can now finish its execution with its normal budget before $t_{\mathcal{L}_j}$. Therefore, number of releases of τ_j interfering with safe budget decreases leading to decrease in interference in the interval [0, t]. Note that for τ_j to overrun its normal budget at $t_{\mathcal{L}_j}$, no release of τ_j in the interval [0, $t_{\mathcal{L}_j}$] are degraded. This can happen when $DL_j = 0$ or when $t_1 = t_2 = ... = t_{\mathcal{L}_j}$.

Condition A5: $t > t_{\mathcal{L}_j}$ and if $DH_j = DL_j = 1$ and if $t_k = t_{\mathcal{L}_j}$, $\forall k$ in the range $1 \leq k < \mathcal{L}_j$

If for all k in the range $1 \leq k < \mathcal{L}_j$, $t_k = t_{\mathcal{L}_j}$, and when τ_j overrun its normal budget at $t_{\mathcal{L}_j}$ then system transition happen directly from S_0 to $S_{\mathcal{L}_j}$. With such a state transition, no releases of τ_j can execute with degraded budget even if $DL_j = 1$. From Definition 11, we note that if $DH_j = 1$, releases of τ_j can execute with their safe budget only in the interval $[t_{\mathcal{L}_j}, t_{\mathcal{L}_{j+1}}]$ while those that are active beyond $t_{\mathcal{L}_{j+1}}$ can execute only with their degraded budgets.

By delaying the release pattern by ϵ, some instances of τ_j that were previously interfering with safe budgets in the interval $[t_{\mathcal{L}_j}, t_{\mathcal{L}_{j+1}}]$ can be pushed beyond $t_{\mathcal{L}_{j+1}}$. These releases of τ_j can now interfere only with degraded budget leading to decrease in interference in the interval $[0, t]$. However, the number of releases of τ_j interfering with safe budgets will remain unchanged in the interval $[0, t]$ for $0 < \epsilon < T_j$ since the release of τ_j that was considered to overrun its normal budget at $t_{\mathcal{L}_j}$ can now overrun its normal budget at any time greater than $t_{\mathcal{L}_j}$. Therefore, interference from τ_j with will remain unchanged in the interval $[0, t]$ for $\epsilon < T_j$.

By advancing the release pattern by ϵ, the instance of τ_j released that was previously considered to overrun its normal budget at $t_{\mathcal{L}_j}$, can now finish its execution with its normal budget before $t_{\mathcal{L}_j}$. Therefore, the number of releases of τ_j interfering with safe budget decreases leading to a decrease in interference in the interval $[0, t]$. ■

In the following, based on the release patterns and conditions discussed in the above two theorems, expressions to find the upper bound number of releases of τ_j interfering in the interval $[0, t]$ for any given valid state transition pattern $P^l \in \mathcal{P}$ with safe budget (denoted as $N_s(j, t, P^l)$), normal budget (denoted as $N_n(j, t, P^l)$) and degraded budgets (denoted as $N_d(j, t, P^l)$) are presented.

Expression for $N_s(j, t, P^l)$:

$$N_s(j,t,P^l) = \begin{cases} 0, & A1 \vee A2 \\ \min\left\{1 + \left\lceil \frac{t - t_{\mathcal{L}_j}}{T_j} \right\rceil_0, \left\lceil \frac{t}{T_j} \right\rceil \right\}, & A3 \\ \min\left\{1 + \left\lceil \frac{min(t, t_{\mathcal{L}_j+1}) - t_{\mathcal{L}_j}}{T_j} \right\rceil_0, \left\lceil \frac{min(t, t_{\mathcal{L}_{j+1})}}{T_j} \right\rceil \right\}, & A4 \vee A5 \end{cases} \tag{4.1}$$

where $P^l \in \mathcal{P}$ is a predefined valid state transition pattern that can occur in the interval $[0, T_i]$. P^l consist an array of state transition time instances such that $0 \leq t_1 \leq t_2 \leq t_3 ... \leq t_L$ where L denotes the maximum criticality level supported by the system.

For any time t in the range $0 < t \leq t_{\mathcal{L}_j}$ considered by the conditions $A1$ and $A2$, no job of τ_j can execute beyond its normal budget. Hence, the value of $N_s(j, t, P^l)$ is 0. Condition $A3$ considers the value of DH_j to be 0, and therefore the maximum number of releases of τ_j that can interfere with safe budget and can fit into an interval of length $t - t_{\mathcal{L}_j}$ is

$$1 + \left\lceil \frac{t - t_{\mathcal{L}_j}}{T_j} \right\rceil_0$$

For smaller values of $t_{\mathcal{L}_j}$, the above expression can be pessimistic when considering the number of jobs that can fit in an interval of length t. Therefore, we define the value of $N_s(j,t,P^l)$ as

$$\min\left\{1+\left\lceil\frac{t-t_{\mathcal{L}_j}}{T_j}\right\rceil_0, \left\lceil\frac{t}{T_j}\right\rceil\right\}$$

For conditions $A4$ and $A5$, the value of DH_j is considered to be 1 and therefore the expression corresponding to these conditions upper bounds the number of jobs of τ_j that can interfere with safe budget in the interval of length $min(t, t_{\mathcal{L}_{j+1}})$ - $t_{\mathcal{L}_j}$. For smaller values of $t_{\mathcal{L}_j}$, only the interval of length $min(t, t_{\mathcal{L}_{j+1}})$ is considered to reduce the pessimism.

Expression for $N_n(j,t,P^l)$:

$$N_n(j,t,P^l) = \begin{cases} \left\lceil\frac{t}{T_j}\right\rceil, & A1 \wedge DL_j = 0 \\ \left\lceil\frac{min(t,t_1)}{T_j}\right\rceil, & A2 \vee (A1 \wedge DL_j = 1) \\ \left\lceil\frac{t}{T_j}\right\rceil - N_s(j,t,P^l), & A3 \wedge DL_j = 0 \\ \left\lceil\frac{min(t,t_{\mathcal{L}_j+1})}{T_j}\right\rceil - N_s(j,t,P^l), & A4 \vee A5 \end{cases} \tag{4.2}$$

As discussed in [19], the upper bound number of releases of τ_j that can interfere τ_i in an interval of length t is given as $\left\lceil\frac{t}{T_j}\right\rceil$. For condition $A1$, if the value of DL_j is equal to 0 then all jobs of τ_j released in the interval [0, t] interfere with normal budget and the value of $N_n(j,t,P^l)$ is given as $\left\lceil\frac{t}{T_j}\right\rceil$. Otherwise, if the value of DL_j is equal to 1 then jobs of τ_j can interfere with normal budget only in the interval [0, t_1] and the value of $N_n(j,t,P^l)$ is given as $\left\lceil\frac{min(t,t_1)}{T_j}\right\rceil$ where t_1 denotes the time instant at which system transition to the safe state S_1 happens. For the condition $A3$, if the value of DL_j is equal to 0 then jobs of τ_j can interfere with normal budget only in the interval [0, $t_{\mathcal{L}_j}$]. Hence, the value of $N_n(j,t,P^l)$ is found by subtracting $N_s(j,t,P^l)$ which is the upper bound number of jobs that can interfere with safe budget in the interval [0, t] from the total number of jobs that can be released in the interval of length t. Conditions $A4$ and $A5$ consider the value of DH_j to be 1. Therefore, the value of $N_n(j,t,P^l)$ is found by subtracting $N_s(j,t,P^l)$ from the total number of jobs that can be released in the interval of length $min(t, t_{\mathcal{L}_{j+1}})$.

Expression for $N_d(j,t,P^l)$:

Once N_s and N_n are found, remaining releases of τ_j can be considered to interfere with the degraded

budgets and is given by the following expression,

$$N_d(j,t,P^l) = \left\lceil \frac{t}{T_j} \right\rceil - (N_s(j,t,P^l) + N_n(j,t,P^l)) \tag{4.3}$$

In the following corollary, the equation for computing the worst-case response time for a task τ_i for a specific valid transition pattern is considered.

Corollary 1 *For a task τ_i, its worst-case response time for a specific valid transition pattern $P^l \in \mathcal{P}$, is given by the solution to the following iterative equation,*

$$R_i^l = C_i^x(R_i^l) + \sum_{j \in hp(i)} \left\{ N_s(j, R_i^l, P^l) * C_j^s + N_n(j, R_i^l, P^l) * C_j^n + N_d(j, R_i^l, P^l) * C_j^{\mathcal{D}} \right\} \tag{4.4}$$

where $hp(i)$ denotes the set of tasks with priority higher than τ_i. For any $t > 0$, the value of $C_i^x(t)$ is given based on the following expression.

$$C_i^x(t) = \begin{cases} C_i^n, & t < t_{\mathcal{L}_i} \\ C_i^s, & t \geq t_{\mathcal{L}_i} \end{cases}$$

For an interfering task τ_j, $C_j^{\mathcal{D}}$ denotes its maximum degraded budget that can be allocated to it and is given by the following expression,

$$C_j^{\mathcal{D}} = max(DH_j * C_j^{\mathcal{H}}, DL_j * C_j^{\mathcal{L}})$$

*where $C_j^{\mathcal{H}}$ denotes the maximum degraded budgets that can be assigned to τ_j when one or more tasks with criticality value **higher** than $\mathcal{L}_j$ overrun their normal budget. $C_j^{\mathcal{L}}$ denotes the maximum degraded budget that can be assigned to τ_j when one or more tasks with criticality value **lower** than $\mathcal{L}_j$, overrun their normal budget.*

Proof: Since the releases of a higher priority task τ_j can interfere either with normal or safe or degraded budgets, the worst-case response time for τ_i is the sum of its safe budget and the upper bound interference from releases of higher priority tasks interfering with normal, safe and degraded budgets. ■

In the following corollary, the equation for computing the worst-case response time for a task τ_i for all possible state transition patterns that can occur in the interval $[0, T_i]$ is considered.

Symbols	Meaning
$hp(i)$	a set of tasks with priority higher than τ_i.
$\mathcal{P}$	a set of valid state transition patterns that can occur in the interval $[0, T_i]$.
t_l	the time instant at which state transition to the safe state S_l.
R_i^l	the worst-case response time of τ_i for a specific valid transition pattern $P^l \in \mathcal{P}$.
R_i^o	the worst-case response time for an old-mode task τ_i.
N_n	the upper bound number of releases of any task interfering with normal budget.
N_d	the upper bound number of releases of any task interfering with degraded budget.
N_s	the upper bound number of releases of any task interfering with safe budget.
$C_j^{\mathcal{D}}$	the maximum degraded budget that can be allocated to τ_i.

Table 4.4: Symbols and their meanings used in the schedulability analysis of the CA-MCS model.

Corollary 2 *The worst-case response time for a task τ_i is given by the following equation,*

$$R_i = \max_{\forall P^l \in \mathcal{P}} R_i^l \tag{4.5}$$

Proof: For a task τ_i, its worst-case response time is the sum total of the maximum possible interference from higher priority tasks that can interfere τ_i among all possible valid state transition patterns. ■

A task is considered to be schedulable if the value of its worst-case response time obtained by equation 4.5 is less than or equal to its deadline.

As the proposed model provides a higher level of flexibility in the choice of degradation of tasks, there is a need to consider many symbols to capture values of many variables to decide on the tasks to be degraded at run-time. Hence, the analysis of the CA-MCS model requires many parameters to represent these variables. In Table 4.4, list of symbols and their meanings used in the schedulability analysis of the CA-MCS model are tabulated for the reference purpose.

4.4.3 Run-Time Complexity for Computing the Worst-Case Response Time

The computational complexity for computing R_i is $O(N * T_i * (T_i)^L)$ where N denotes the total number of tasks, T_i denotes the time period of τ_i (the task whose worst-case response time is to be found) and T_i^L denotes the maximum possible number of transition instances that need to be considered in the analysis. L can be regarded as a constant, and hence the resulting complexity is still pseudo-polynomial with respect to the number of tasks.

4.4.4 Priority Assignment

The three conditions that need to be adhered by a schedulability test (referred to as S below) so that the Audsley's algorithm can be employed for the priority assignment discussed in [77] are the following:

- "The schedulability of a task τ_k may, according to test S, depend on any independent properties of tasks with priorities **higher than** τ_k, but not on any properties of those tasks that depend on their relative priority ordering".
- "The schedulability of a task τ_k may, according to test S, depend on any independent properties of tasks with priorities **lower than** τ_k, but not on any properties of those tasks that depend on their relative priority ordering".
- "When the priorities of any two tasks of adjacent priority are swapped, the task being assigned the higher priority cannot become unschedulable according to test S, if it was previously schedulable at the lower priority. (As a corollary, the task being assigned the lower priority cannot become schedulable according to test S, if it was previously unschedulable at the higher priority)".

An inspection of the worst-case response time equation presented in equation 4.4 is observed to adhere to the three conditions stated in [77]. Therefore, the Audsley's algorithm can be employed for the priority assignment. It is clear from equation 4.4 that to compute worst-case response time of τ_i, only tasks with priority higher than τ_i are considered. The parameters of the CA-MCS task model such as safe budget, normal budget, degraded budgets, criticality and time period of each higher priority task are considered to compute the worst-case response time of a task. These parameters are independent and are not affected by the priority ordering of tasks. In equation 4.4, it can be observed that the budget with which any higher priority task can interfere in a given time interval depends on the time instances considered in the state transition pattern and not on the priority ordering of higher priority tasks. The maximum value of the degraded budget from any higher priority task in a given time interval is dependent only on the criticality level of other tasks that can overrun their normal budget in that interval and independent of their priority ordering thereby satisfying conditions 1 and 2. Since equation 4.4 considers only the interference from higher priority tasks, swapping tasks of adjacent priority cannot make the task being assigned the higher priority unschedulable as the number of higher priority tasks that can now interfere at the new priority level decreases. Therefore, the worst-case response time for the task being assigned the higher priority is reduced. Similarly, the worst-case response time of the task that is assigned with a lower priority value increases due to increase in the number of higher priority tasks and therefore it cannot become schedulable, if it was previously unschedulable at the higher priority.

4.4.5 Feasibility Analysis for the CA-MCS model

Let Γ denote a set of tasks specified as per the CA-MCS model presented in the subsection 4.3.3. Γ is said to be feasible if the following conditions are satisfied,

$$\max_{0 \leq l \leq L} \left(\sum_{\forall \tau_j \in \Gamma} maxB(\tau_j, l)/T_j \right) \leq 1 \tag{4.6}$$

The function $maxB(\tau_j, l)$ returns the maximum budget with which τ_j can execute in the safe state S_l. Therefore, equation 4.6 checks whether the sum of the utilisation of all tasks of the system when it executes only in the safe state S_l without undergoing any state transitions is less than or equal to 1. When the taskset Γ satisfies equation 4.6, it means that Γ is feasible. It is to be noted that the feasibility condition in equation 4.6 is only a necessary condition.

The feasibility condition presented above does not consider degradation information and thus leads to a run-time complexity of $O(L * N^2)$ where L denotes the total number of criticality levels supported by the system and N denotes the total number of tasks in the system S.

4.5 Evaluation

In the following, we present results of the experiments that were primarily designed to show the impact of considering parameters that are unique to the CA-MCS model such as degraded budgets and behaviour information of tasks on the taskset schedulability. In the worst-case response time analysis that was presented in equation 4.5, any task τ_j can have 4 different combinations of the values of DH_j and DL_j. Each combination can be considered as a degradation scheme to handle budget overrun of a task. To evaluate the differences between these schemes, we generate 4 categories of tasksets depending on the values of DH_j and DL_j as follows.

- In category 1, behaviour values of any task τ_j are such that τ_j is always degraded whenever a task with criticality higher than $\mathcal{L}_j$ overruns its normal budget and remains unaffected when any other tasks overrun their normal budget. This is considered as **scheme 1 (degradation for HC overrun)** where $DH_j = 1$ and $DL_j = 0$. This is similar to the degradation scheme adopted in the majority of MCS studies that consider dual criticality system where all lower criticality tasks are either suspended or degraded when any higher criticality task overruns its normal budget.
- In category 2, any task τ_j is always degraded whenever any other task overrun its normal budget.

This is considered as **scheme 2 (degradation for HC and LC overrun)** where $DH_j = 1$ and $DL_j = 1$.

- In category 3, no tasks are degraded. This is considered as **scheme 3 (no degradation)** and hence $DH_j = DL_j = 0$.
- In category 4, τ_j is degraded whenever a task with criticality lower than $\mathcal{L}_j$ overruns its normal budget and hence $DH_j = 0$ and $DL_j = 1$. This is considered as **scheme 4 (degradation for LC overrun)**.

Further, to evaluate the effect of different degrees of task degradation on the schedulability of the generated tasksets, the parameter termed as 'Degradation Factor' is introduced. The value of 'DF' determines the maximum value of the degraded budget for a task in any given mode.

4.5.1 Taskset Generation

For the experiments, the parameters of the taskset were generated as follows:

- Utilisations of tasks were generated using the UUnifast algorithm [78]. This algorithm ensures that the distribution of the generated utilisation values are unbiased.
- Time periods of tasks were generated in the range 100 to 1000ms and follow a log-uniform distribution [79].
- Deadlines for tasks were set equal to their periods.
- The safe budget of each task was determined based on the period and utilisation selected: $C_i^s = U_i/T_i$
- The normal budget of each task was a determined based on the fixed multiplier of the safe budget, $C_i^n = C_i^s$/CF (e.g., CF = 2.0).
- Each task belongs to one of 4 criticality levels (1 to 4) and the probability that a generated task τ_i is assigned with a criticality is $1/4$.
- The degraded budgets for τ_j are generated randomly in the range [0, DF*C_j^n] where DF denotes 'Degradation Factor' whose range is $0 \leq$ DF ≤ 1 (*e.g.*, C_i^k = DF*C_i^n where DF = 0.5). Therefore, degraded budgets are always lower than or equal to the normal budget.

In our experiments, utilisation of the taskset was varied from 0.1 to 1.0[2]. For each utilisation value in each category, 1000 tasksets were generated. The schedulability of the generated tasksets were determined using the schedulabitlity test based on equation 4.5.

4.5.2 Schedulability Tests Investigated

To the best of our knowledge, there is no existing study that provides a sufficient schedulability test under fixed-priority scheduling scheme for an uniprocessor system for a MCS task model that considers multiple degraded budgets for tasks. Among the state-of-the art MCS task models, only the Interference Constraint Graph (ICG) task model considered in [30] was observed to be comparable against the proposed CA-MCS model due to two reasons:

- The ICG task model is a specification of mixed-criticality tasksets where a designer can specify the allowed interferences between a pair of tasks during run-time. The ICG model considered in [30] does not capture the possibility of multiple degraded budgets of a task. However, it is possible to achieve a higher level of flexibility in the choice of tasks to be suspended depending on the overrun tasks. When the interference constraint graph is fully connected from high criticality tasks to low criticality tasks, it means that any budget overrun of high criticality tasks, all low criticality tasks are dropped immediately to handle system overload due to budget overrun of tasks.

- A sufficient schedulability test based on worst-case response time analysis under preemptive fixed priority scheduling has already been derived for the ICG model in [30]. However, it is important to note that schedulability test for ICG has been derived by considering suspension of tasks with relatively lower criticality than the overrun tasks. Hence, only normal and safe budgets of the generated tasks were considered in the evaluation of ICG.

Other existing MCS models such as MC-ADAPT [80] and MS-DRT [31] consider Earliest Deadline First (EDF) scheduling technique and hence it is not meaningful to compare their schedulability against the proposed models using synthetic tasksets.

A detailed description of the ICG task model along with its fixed priority response time analysis based sufficient schedulability test are discussed in section 2.2.2. Equation 2.9 corresponds to the schedulability test for ICG that was originally presented in [30] and can be directly used to compute the

[2]Utilisation here is computed from the normal budget C_i^n values only *i.e.* $U_i = C_i^n/T_i$

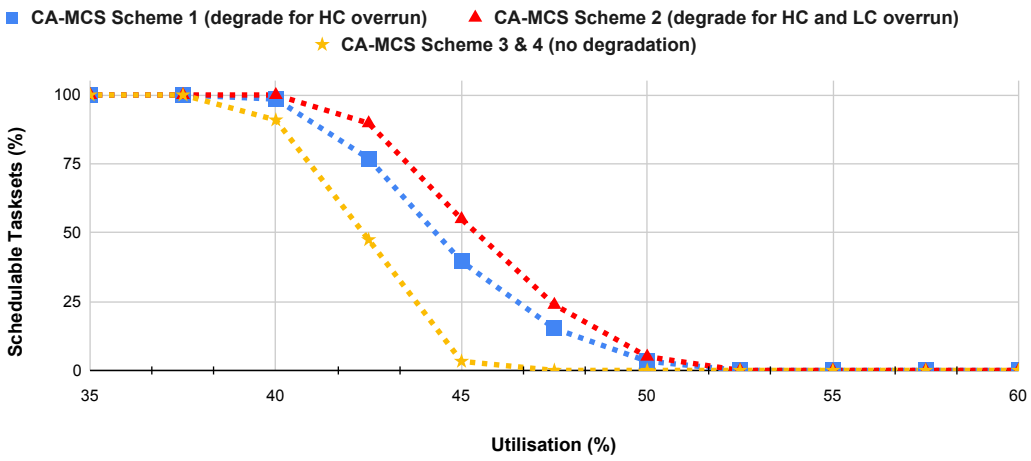

Figure 4.10: Schedulability for different degradation schemes considered for the CA-MCS model.

worst-case response time for the generated tasks. A taskset Γ is ICG-Schedulable if $\forall \tau_i \in \Gamma$, we have that $R_i \leq D_i$, where R_i is a fixed-point solution of the following recurrence relation:

$$R_i = \sigma(\tau_j, \tau_j) + \sum_{j \in \mathbf{hp}(i)} \left\lceil \frac{R_i}{T_j} \right\rceil \min\{\sigma(\tau_j, \tau_j), \sigma(\tau_j, \tau_i)\} \tag{4.7}$$

where $\mathbf{hp(i)}$ denotes a set of tasks with priority higher than τ_i, $\sigma(\tau_j, \tau_j) = C_j^s$, and $\sigma(\tau_j, \tau_i)$ is given based on the following expression:

$$\sigma(\tau_j, \tau_i) = \begin{cases} C_j^n, & \mathcal{L}_j > \mathcal{L}_i \\ C_j^s, & \mathcal{L}_j \leq \mathcal{L}_i \end{cases}$$

In the above expression, when criticality of the interfering task τ_j is higher than τ_i, τ_i is suspended when τ_j overruns its normal budget (C_j^n). Hence, τ_j can interfere τ_i only upto C_j^n units. Otherwise, when criticality of the interfering task τ_j is lower than or equal to τ_i, τ_i is not affected when τ_j overruns its normal budget (C_j^n). Hence, τ_j can interfere τ_i upto C_j^s units.

4.5.3 Evaluation of the CA-MCS Model

It is not always meaningful to compare different task models in terms of response time analysis because the objective of these different task models is not to improve schedulability, but rather to provide better degradation support. Therefore, an automotive testbed was built and experiments were performed

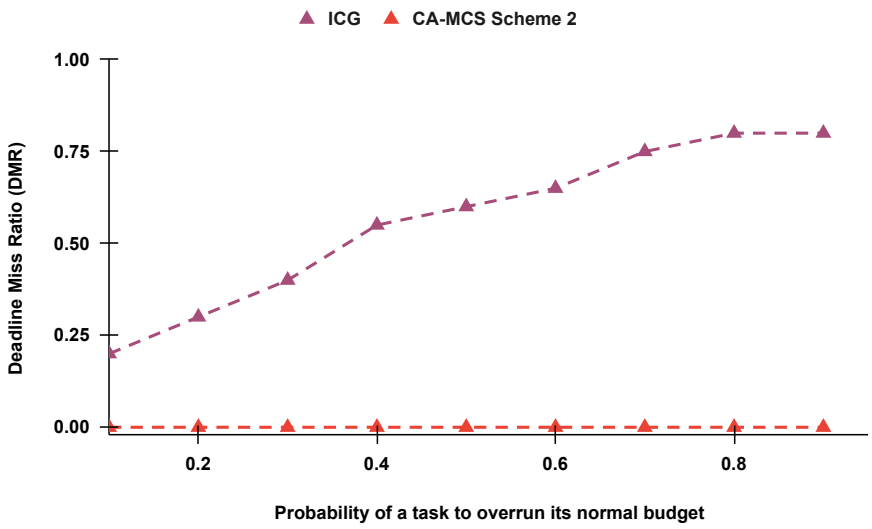

Figure 4.11: The Deadline Miss Ratio for different values of the probability of a task to overrun its normal budget.

to compare different degradation strategies supported by the existing models and the CA-MCS model. The details of the testbed and experiment results are discussed in Chapter 6.

We compare CA-MCS with ICG in terms of deadline miss ratio (DMR) of tasks. DMR is the ratio of the number of the unfinished jobs over the total number of jobs released in a given time interval. As the ICG considers dropping of lower criticality tasks to handle budget overrun, we assume that a task in the dropped state releases its job but does not execute. For a given randomly-generated task set schedulable by both CA-MCS and ICG model, we simulate the behavior of tasks with a given probability of a task to overrun its normal budget denoted as P^O, for 1000 time units. According to CA-MCS, on idle tick, the system is switched back to the normal state (all tasks are assigned with normal budget and all lower criticality tasks are active).

Figure 4.11 shows DMR varying P^O from 0.05 to 0.95 in increments of 0.5 (utilisation of each taskset was 40%). ICG shows a higher DMR for a higher P^O (a large number of budget overruns at runtime) while with CA-MCS no jobs miss their deadline as jobs are allowed to execute with their degraded budgets to finish their execution.

Figure 4.12 shows DMR varying simulation durations (utilisation of each taskset was 40%). It shows that the simulation duration does not affect the simulation results.

Figure 4.10 plots the percentage of tasksets generated that were schedulable for a system of 20 tasks,

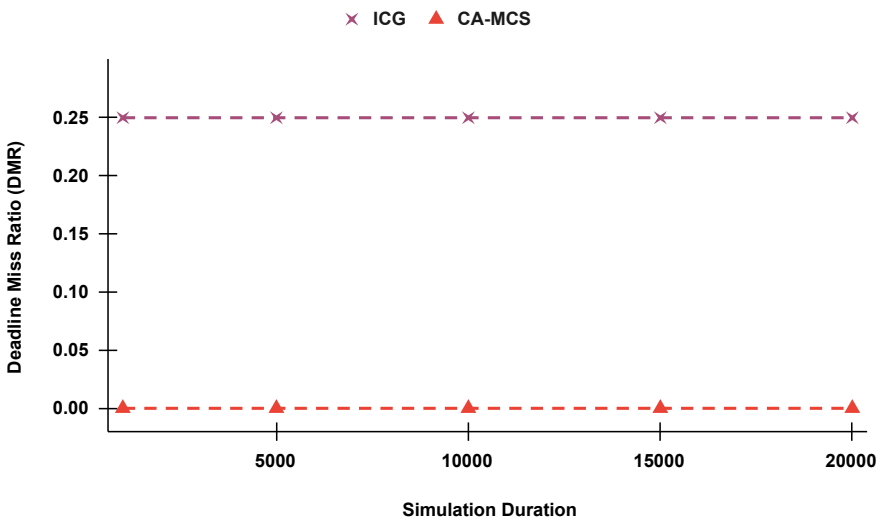

Figure 4.12: The Deadline Miss Ratio for different simulation duration.

with a degradation factor of 0.2, the task having a safe budget 2.0 times its normal budget (CF = 2.0) and for 4 degradation schemes, for the CA-MCS model. It is observed that with higher utilisations, scheme 2 seem to perform better than scheme 1. It is also observed that there is no difference in performance for schemes 3 and 4. Similar performances among these schemes are due to the same values of worst-case interference when computed using the proposed worst-case response time equation for the CA-MCS model.

In the following figures, we consider the weighted schedulability measure $W_x(q)$ [81] as a function of parameter q for schedulability test x. For each value of q, this measure aggregates results for all of the generated tasksets for utilisation levels considered in the range 0.1 to 1.0 with interval steps of 0.1. Let $S_x(\Gamma, q)$ denote the binary result (0 or 1) of schedulability test x for a taskset Γ with parameter value q:

$$W_x(q) = (\sum_{\forall \Gamma} u(\Gamma).S_x(\Gamma, q))/\sum_{\forall \Gamma} \tag{4.8}$$

where $u(\Gamma)$ is the utilisation of taskset Γ. The main benefit of using the weighted schedulability measure is to reduce what would normally require a 3-dimensional plot to 2 dimensions [81]. By multiplying the taskset utilisation with the individual schedulability results indicates that higher weights placed on being able to schedule higher utilisation tasksets. Figure 4.13 depicts the schedulability when the number of tasks in a taskset is varied. Figure 4.15 varies the criticality factor. Since the CA-MCS model supports multiple degraded budgets (maximum of $n - 1$ degraded budgets possible) for a task, to quantify the

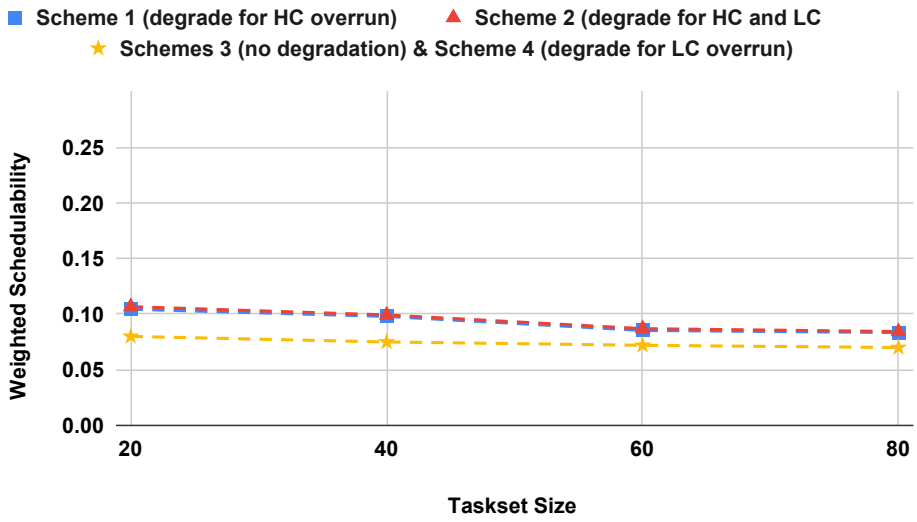

Figure 4.13: Effect of varying the number of tasks in the schedulability for the CA-MCS model.

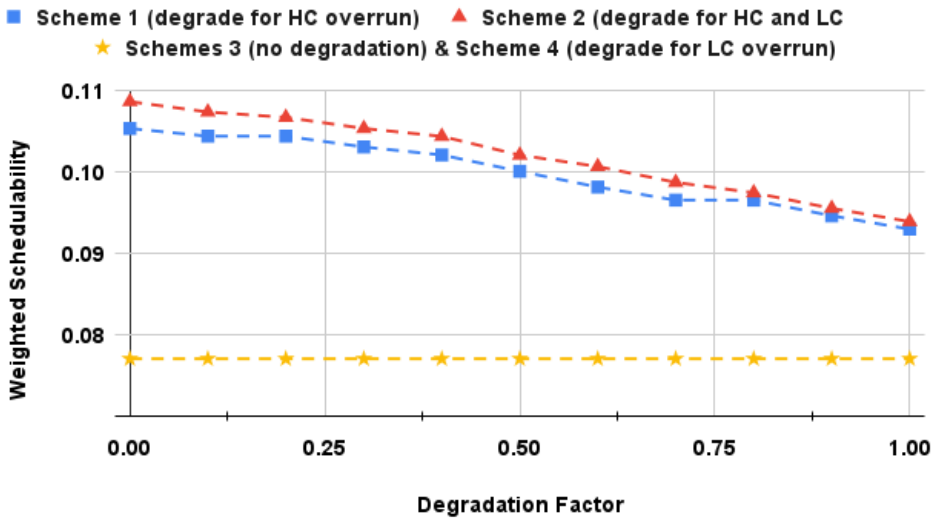

Figure 4.14: Effect of varying the degradation factor (DF) in the schedulability for the CA-MCS model.

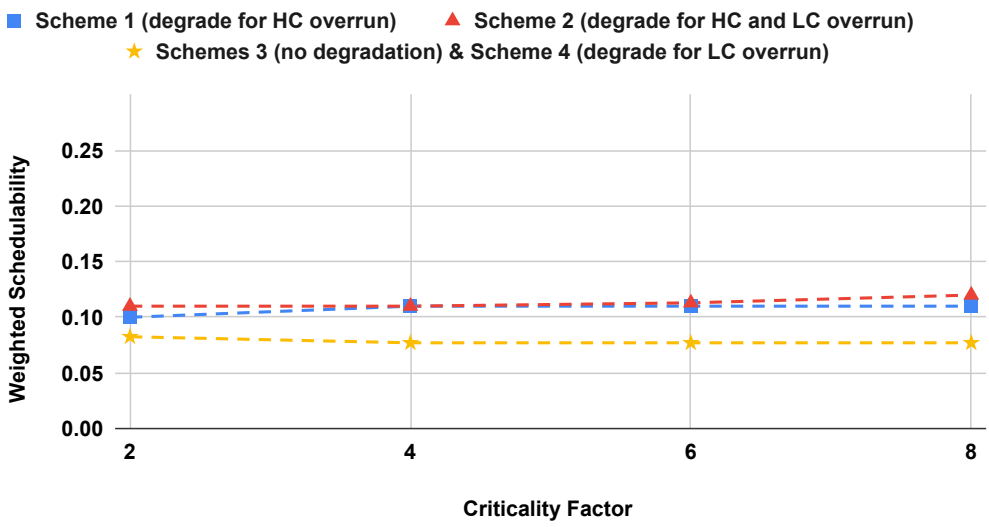

Figure 4.15: Effect of varying the criticality factor (CF) in the schedulability for the CA-MCS model.

effects of degradation in the schedulability performance for schemes 1, 2 and 4, we consider degradation factor (DF) applicable to all tasks in the system. Figure 4.14 varies the degradation factor (DF) from 0 to 1.0. Note that DF = 0 means that the task is suspended to handle the budget overrun and DF = 1 denotes that the task's degraded budget is equal to its normal budget. From Figure 4.14, it is clear that when DF = 1, schemes 1 and 2 perform better than the scheme 3 even though in both cases task's degraded budget are equal. This difference can be attributed to the worst-case interference which differs for schemes 1, 2 with that of schemes 3, 4 even if their budgets are equal in their magnitude where a task cannot overrun its degraded budget, but can overrun its normal budget. Figure 4.13 and Figure 4.14 show that there is a decline in the schedulability performance for higher values of taskset size and DF. They also show that schemes 1 and 2 perform better than schemes 3 and 4 at higher DF values.

The key observation from these figures is that the relationship between different degradation schemes considered for the CA-MCS model on the graphs remains stable and that the schedulability test proposed for the CA-MCS model is an effective way of scheduling mixed criticality systems when task degradation to handle the system overload is considered.

4.6 Conclusion

This chapter presented the Context-Aware Mixed Criticality System (CA-MCS) model to handle the system overload caused due to budget overrun of tasks within a specific mode. The model achieved its main objective of providing a higher level of flexibility to capture multiple ways of degrading a task's functionality and in the choice of selection of tasks for graceful degradation. The model can be considered expressive enough to use criticality information of tasks to achieve desired performance degradation. The run-time complexity of the proposed schedulability test for the CA-MCS model is pseudo-polynomial with respect to the number of tasks. The main objectives of the experiments based on synthetic tasksets were to show the usefulness of capturing behaviour parameter and degraded budgets of tasks. From the experiment results, it can be observed that the number of tasksets that were deemed schedulable for a given utilisation value depends on the values of DH and DL and the degree of degradation of tasks. The effect of varying the value of degradation factor (DF) for each scheme gave some insights on how the schedulability of tasksets depend on the degraded budgets of tasks. Therefore CA-MCS is generic enough to support any degradation scheme and is suitable to be adopted for any safety-critical domain.

Chapter 5

Design and Analyses of Multi Mode Mixed-Criticality System

5.1 Introduction

The software performing safety-critical functionalities in domains such as automotive are capable of executing in different modes [64]. A mode here refers to the set of tasks configured to perform specific safety functionalities. Each mode can differ by the set of functionalities with distinct safety and performance aspects. For example, a vehicle can be considered to run safety functionalities such as Adaptive Cruise Control (ACC) or Parking Assist (PA) depending on the environment in which it is driven (highway road or parking lot). A system undergoes a mode transition upon a mode change request (MCR) triggered by external or internal events. When system transition happens from one mode to another, required functionality may change resulting in a change in the number of tasks and values of their parameters like budget, period, deadline etc.

A system overload for a safety-critical system can occur in two possible ways. First, an unexpected increase in the execution time of a task, also termed as a budget overrun, is a common theme in the existing mixed-criticality system literature. The Context-Aware Mixed Criticality System (CA-MCS) model presented in the Chapter 4 is designed to handle system overload due to budget overrun of tasks that can occur within a specific mode. Second, a spike in the resource consumption pattern leading to a temporary system overload when tasks belonging to different modes execute together for a certain time interval when the system switches from one mode to another. This can lead to a scenario where tasks which are schedulable in a mode may not be schedulable during a mode transition due to additional load

from tasks of the new-mode.

A vast majority of existing models focus on handling only one aspect of the system overload *i.e.* they handle either budget overrun ([19, 38–40, 82]) or mode transition ([6, 33, 35, 64]). However, the model proposed in [57], considers both mixed-criticality and mode change aspects but does not consider the possibility of multiple degraded budgets while providing execution support for low criticality tasks when any higher criticality task overrun its normal budget. Also, the model does not consider the possibility of multiple degraded budgets for a task. As discussed in Section 4.2.3, graph-based MCS models [30, 31] are highly generic to handle system overload due to both budget overruns and mode transitions but these models do use criticality information of tasks effectively leading to a higher level of complexity to analyse their timing behaviour.

Existing MCS models that are based on the sporadic task model use criticality to decide on the tasks to be degraded to handle system overload. However, with graph-based models, all the execution scenarios of tasks required to decide tasks that needs to be suspended or degraded can be modelled with higher number of vertices. Therefore, criticality information of tasks is not required if a designer chooses to use graph based MCS model.

In this chapter, the Multi-Mode Mixed Criticality System (MM-MCS) model[1] is introduced to address an important challenge in handling system overload when a system is capable of experiencing budget overruns and functional mode changes. To handle system overload, the MM-MCS model extends the idea of CA-MCS to decide on the way in which tasks need to be degraded within a specific mode and considers a set of rules (mode change protocol) to determine which task's budget overrun can be allowed to influence the system's degradation when budget overrun and mode transition occur simultaneously. Additionally, the notion of offset (a time delay) for tasks is also considered to postpone releases of new-mode tasks to handle system overload during mode transitions.

The structure of this chapter is as follows. Section 5.2 presents the motivation for the MM-MCS model. The syntax and semantics of the MM-MCS model is presented in Section 5.3. Section 5.4 is devoted to the derivation of the worst-case response time and its correctness. An algorithm to compute offsets is presented in Section 5.5. The experiments and results are discussed in Section 5.6. Finally, Section 5.7 concludes this research work.

[1]The work in this chapter will be submitted as Vijaya Kumar Sundar, Saravanan Ramanathan and Arvind Easwaran. Design and analyses of functional mode changes for mixed criticality systems, *Real-Time Systems Journal*.

5.2 Motivation

In Chapter 4, key observations of MCS that are required to be satisfied while operating in a specific mode are discussed. In the following, based on the recommendations of ISO 26262 and observations from safety-critical automotive applications, additional observations of the system that can exhibit both mixed-criticality and functional mode change behaviours are discussed. Therefore, MM-MCS model can be considered as an extension of the CA-MCS model with additional parameters and semantics to capture both mixed-criticality and mode change aspects of the safety-critical real-time systems. Therefore, the MM-MCS model is also required to satisfy observations **O1 to O5** that are discussed in Section 4.2.

5.2.1 Functionality Change

The functionality of a safety-critical real-time task can vary depending on the mode. For example, Adaptive Cruise Control (ACC) [28] application can operate in two modes viz 'constant distance' or 'constant velocity'. In constant distance mode, ACC implements an algorithm to maintain a specific distance between the vehicles (say 50m) such that under heavy braking of the lead vehicle, there is sufficient time for the follower vehicle to apply brakes to maintain a safe distance. In constant velocity mode, ACC implements an algorithm to ensure that vehicle's velocity is always constant. Similarly, when the vehicle switches to the 'collision avoidance' mode, functionality of ACC is no longer required and Collision Avoidance Application (CAA) [83] is activated to perform the emergency braking to ensure there is no head-on collision with the lead vehicle. The functionality of the steer control application (SC) in the constant-distance mode is to apply steering to ensure the vehicle is moving along the centre of the track. But in collision avoidance mode, SC can perform an emergency manoeuvre (left or right steer) to avoid a head-on collision with the lead vehicle. As the functionality of a task depends on the mode, parameters of a task such as its budget, period and deadline can also vary for each mode [7]. Table 5.1 shows the functionalities of applications in the constant distance mode and in the collision avoidance mode.

As per the functional safety standards such as ISO26262, the factors (*e.g.,* probability of exposure) used to determine ASIL for a software element are arrived by considering the maximum safety assurance level demanded by the software to mitigate all possible safety hazards that can occur due to its malfunction among all possible modes of its operation. This is evident from the following three clauses stated in ISO 26262.

Modes ↓	**Tasks Functionality**			
	Adaptive Cruise Control (ACC)	**Forward Collision Warning (FCW)**	**Collision Avoidance (CA)**	**Steering Control (SC)**
Cruise Control - Constant Distance Mode	1. Apply throttle or brake to maintain constant distance between the lead vehicle and the follower vehicle. 2. Dynamic Speed Adaptation for a smooth maneuvering through curves.	Issue warnings when distance between vehicles is less than 10m.	Not active in this mode.	Apply steering to ensure vehicle is moving along the center of the track.
Collision Avoidance Mode	Not active in this mode.	Not active in this mode.	Apply emergency braking if the distance between the lead and the follower vehicle is less than 5m.	Apply left steer or right steer to avoid head-on collision with the lead vehicle.

Table 5.1: Example tasks performing automotive functionalities in different modes.

- Clause 7.4.3.4 in part 3 of ISO 26262 states that "The probability of exposure of each operational situation shall be estimated based on a defined rationale for each hazardous event. The probability of exposure shall be assigned to one of the probability classes, E0, E1, E2, E3 and E4".
- Clause 7.4.4.3 in part 3 of ISO 26262 states that "A safety goal shall be determined for each hazardous event with an ASIL evaluated in the hazard analysis. If similar safety goals are determined, these may be combined into one safety goal".
- A note under Clause 7.4.4.4 states that "If combined safety goals refer to the same hazard in different situations, then the resulting ASIL of the safety goal is the highest one of the considered safety goals of every situation."

Therefore, from the above clauses, we can observe that the ASIL of a task is derived by considering the maximum ASIL of the safety goal among ASILs of all the safety goals that it is designed to achieve. Therefore, with ASIL of a task same as its criticality, the criticality of a task can be considered as a constant value in all modes. Based on the above discussions, the key observations that need to be satisfied by any system model consisting of tasks that can exist in different modes can be given as follows.

- **Observation O6 (Resource Consumption):** *The model should capture the properties of tasks that can exhibit functional mode changes upon the occurrence of Mode Change Request (MCR). For example, when an MCR occurs, the resource consumption pattern for such tasks can either increase or decrease depending on the required functionality in a specific mode.*

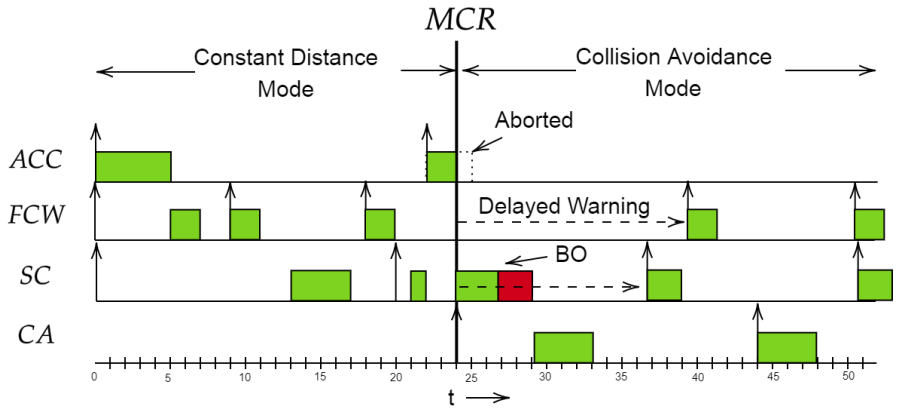

Figure 5.1: An example schedule for tasks described in the Table 5.1.

- **Observation O7 (Periodicity and Criticality) :** *Some tasks (such as sampling tasks in control systems) may need to execute periodically even during the mode transition. The activation pattern of these tasks must be carefully handled during the transition. Similarly, it is reasonable to assume that the criticality of a task remains unchanged in all those modes in which it is active.*

5.2.2 Timeliness of the Mode Transition

In ISO 26262 [14], it is recommended that when the system's safety is threatened, a designer can consider safety mechanisms that can carry out system transition to a mode where the emergency operation can be performed to provide safety. The ISO 26262 stresses the need that emergency operation cannot be long and can be considered safe only if it is performed within a short time interval. Further, warning and degradation concept of ISO 26262 recommends that the conditions to switch to any safe state after the emergency operation have to be clearly specified as part of the system design. For this, the system design can include degradation (reduced functionality, performance, or both) as part of the emergency operation with clear specifications on the way in which the driver can be alerted of potentially reduced functionality and the way to provide this reduced functionality to reach a safe state.

Let us consider a system containing automotive applications such as Adaptive Cruise Control (ACC), Forward Collision Warning (FCW), Steering Control (SC) and Collision Avoidance Application (CAA) as an example to illustrate the way in which the above recommendations can be applied. The details of these applications are considered in Table 5.1. In the schedule shown in Figure 5.1, the system initially starts in constant distance mode. In this mode, the functionality of ACC is to maintain a constant distance

between the leader vehicle and the follower vehicle. The functionality of the Forward Collision Warning (FCW) is to provide warnings when the distance between the vehicles is less than 10 meters. Similarly, when the distance between the lead vehicle and the follower vehicle is less than or equal to 5m, there is a high possibility of a head-on collision between these vehicles thereby threatening the system's safety. This scenario can trigger a mode change request for the system to execute in collision avoidance mode. In Figure 5.1, the system switches to collision avoidance mode at t = 24 units. The first release of CAA task is activated immediately at the instant when the mode change request is triggered (t = 24 units) so that the vehicle's velocity can be reduced with emergency braking. Here, the emergency operation of applying brakes is performed by Collision Avoidance Application (CAA). At the end of emergency braking, one option is to give back the control to the driver. For this purpose, visual or audio warnings related to the status of the vehicle control are given by the Forward Collision Warning (FCW) task at the end of emergency braking. As the functionality of ACC is no longer required in the emergency interval, it gets aborted immediately at t = 24 units.

In the example considered, the functionality of the steer control (SC) task in the constant-distance mode is to maintain the vehicle at the centre of the lane. However, the functionality of the steer control (SC) task in the collision avoidance mode is to decide on the emergency manoeuvre (with left or right steer) to avoid head-on collision. It is possible that an instance of the steer control task released in the constant distance mode and active in the collision avoidance mode can experience budget overrun (BO) in the collision avoidance mode. Under such scenarios, it is important not to suspend the steer control task abruptly, but rather allow it to complete its safety functionality corresponding to the constant distance mode. Further, new releases of steer control task in the collision avoidance mode can be activated after checking for the possibility of avoiding the collision with left or right steer after a specific delay relative to the MCR instant. These releases can execute with the functionality corresponding to the collision avoidance mode *i.e.* deciding the emergency manoeuvre (with left or right steer). Based on the above discussions, the following requirements for the design of mode change systems can be recognized.

- **Observation O8 (Promptness):** *There can be new-mode tasks whose execution are required to be completed before a determinate time after the mode change was requested. Similarly, there can be new-mode tasks whose activation can be delayed till the emergency operation is complete. These two requirements model the need for a prompt response, especially useful when the system switches to an emergency mode.*

- **Observation O9 (Safety Violation):** *To ensure that safety violation does not occur, for any task*

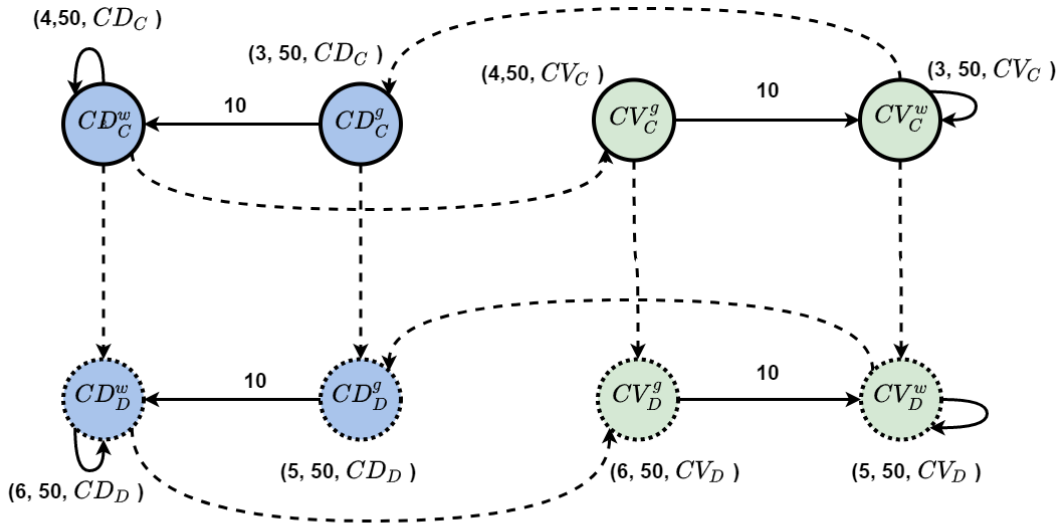

Figure 5.2: MS-DRT model for Adaptive Cruise Control.

that is active in both old-mode and in the new-mode, it is important to ensure that all of its releases are allowed to complete for its allocated budget.

5.2.3 Expressiveness and Analysis Efficiency of the MCS Model

Existing MCS models that are based on the sporadic task model do not capture some of the key observations of MCS such as multiple degraded budgets of a task, or higher level of flexibility in the choice of tasks to be degraded. Graph-based MCS models such as [30] and [31] are highly generic and hence can model budget overruns, task degradation, mode transitions and offsets for tasks. As stated in Chapter 4, the drawback of graph-based models is that they can easily lead to a higher level of complexity to model and analyse its timing behaviour with an increase in the number of tasks. To illustrate the above challenges, the Adaptive Cruise Control (ACC) application that exhibits both mixed-criticality and functional mode change behaviour is modelled as per the syntax and semantics of the Mode-Switching Digraph Real-Time (MS-DRT) task model proposed in [31].

The ACC task consists of two functional modes, 'constant distance mode (CD)' and 'constant velocity mode (CV)'. The software (task) of ACC is assumed to have two execution estimates in each mode, one is estimated at ASIL C and the other is estimated at ASIL D. If the ACC task overruns its estimate at ASIL C, the system should enter any degraded mode where ACC task is given additional time to

execute with WCET valid at ASIL D at the expense of some other tasks. Further, it is assumed ACC task is never degraded. Therefore, we get four modes in total, the modes CD and CV using ASIL C execution time estimates, and degraded versions of the same modes valid at ASIL D. We call the modes CD_C, CV_C in ASIL C, and CD_D and CV_D in ASIL D. With these names, the mode structure of the ACC task is shown in Figure 5.2.

The edge represented as dotted lines denotes switching from any mode at ASIL C to the corresponding one at ASIL D (*e.g.*, from CD_C to CD_D). The logic for this mode switching means that control is shifted to a newer version of the same vertex in the higher criticality level. The gate vertex, superscripted by "g", models logic for the mode transition at either ASIL C or ASIL D. The sporadic behaviour of the task in a mode is captured by the work vertex, superscripted by "w". The edges that are drawn as straight arrows represent possible control flow without any change in the mode of the corresponding job type.

From the example, it clear that to model a system consisting of n tasks where each task is capable of getting degraded in $n-1$ distinct ways, and when the system is capable of operating in m possible modes, a designer may have to consider a maximum of 2^{n+m} number vertices and edges to specify all possible degraded budgets in a specific mode for all possible mode transitions. This also has an impact on the timing analysis of the graph-based MCS models.

Based on the above discussions, the following observations can be made - **Observation O10 (Expressiveness):** *The syntax and semantics of the model must lead to analysis with an acceptable complexity to perform its timing verification for mode transitions.* **Observation O11 (Efficient mode transition):** *Any decision regarding budget allocation or offsets for tasks, that is required to be taken online during the mode transition, should be simple and must lead to quick transition.*

5.3 The Multi Mode Mixed Criticality System Model

In this section, at first, definitions for some of the terms relevant to mode change in MCS are provided. This is followed by the discussion on how the observations of a system that can exhibit both mixed-criticality and functional mode changes are translated to the syntax and semantics for the proposed MM-MCS model. In the later part of this section, we present an example to illustrate the proposed model.

Definition 14 (Old-mode task) *For a given mode transition from an old-mode M_p to the new-mode M_s, a task is said to be an **old-mode task** if it has its release time in M_p.*

Definition 15 (New-mode task) *For a given mode transition from an old-mode M_p to the new-mode M_s, a task is said to be a **new-mode task** if it has both its release time and deadline in M_s.*

5.3.1 Mapping Requirements to the Model Parameters

To address **O1 - O5**, approach that is used in the Context-Aware Mixed Criticality System (CA-MCS) model proposed in Chapter 4 is adopted. Therefore, all assumptions regarding task parameters such as budgets, behaviour etc., are also applicable for the MM-MCS model. In the following, a detailed discussion on the intuition behind design choices considered for semantics of the MM-MCS is provided.

5.3.1.1 Multiple budgets

For each task, a normal budget, a safe budget and set of all degraded budgets that can be allocated to it in a specific mode is considered. For each task belonging to a specific mode, a behaviour parameter is considered to convey whether the allocated budget is normal, safe or degraded and to decide whether a task be allowed to overrun when a specific budget is allocated to it. At worst-case, for a system with n tasks, for any task τ_i, at most $n-1$ degraded budgets corresponding to the budget overrun of the remaining $n-1$ tasks (excluding τ_i) is assumed. We assume that the normal budget of a task is lower than or equal to its safe budget and the degraded budget of a task is always lower or equal to its normal budget. We assume that the estimation of the task's normal budget is done with lower pessimism than its safe and degraded budgets. Hence, a task is allowed to overrun only its normal budget and not allowed to overrun its degraded budgets or its safe budget. It is assumed that degraded budget for a task is always sufficient to achieve its required safety functionality.

As the system can be active in multiple modes performing different functionalities, each task is assigned with a unique set of budgets, behaviours and time period based on the mode to satisfy **O6**. Based on the observation **O7**, the criticality of a task is considered to be same in all those modes in which it is active. To recognize the change in resource consumption pattern upon the arrival of MCR, for a given pair of modes, a task can belong to one of 5 task types (old-mode completed, aborted, unchanged, changed, wholly new). Details of each task type is provided in Section 5.3. This approach is similar to the one discussed in [33].

5.3.1.2 Offsets for new-mode tasks

As the functionality of tasks in the new-mode can be entirely different from that of old-mode, there is no use in considering degradation of new-mode tasks to handle system overload due to budget overrun of the old-mode tasks. Therefore, to handle budget overrun of any old-mode task after the occurrence of mode change request (MCR), instead of degradation of new-mode tasks, offsets for new-mode tasks is considered. An offset here refers to a time delay that is applied to the first release of new-mode tasks after the MCR. To improve the promptness and periodicity of higher criticality tasks and to satisfy **O7** and **O8**, algorithm is presented in section 5.5 to compute the offset value based on the criticality of tasks. At run-time, for a new-mode task, offset is considered relative from MCR instance. In case offsets are not desirable for some tasks (like emergency braking), the proposed model allows a designer to provide null offsets.

5.3.1.3 Determination of the system's state and task degradation in the new-mode

As no tasks (old- and new-mode tasks) are considered for degradation to handle budget overrun of old-mode tasks in the new-mode, the semantics of the MM-MCS model is designed in such a way that the system transition to the normal state ($S_{0,s}$) of the new-mode happens immediately upon the occurrence of Mode Change Request (MCR). However, all instance of tasks released in the old-mode but active in the new-mode and executing with normal behaviour are allowed overrun their normal budgets in the new-mode. In other words, there is no impact on the degradation of new-mode tasks due to the budget overrun of old-mode tasks active in new-mode. This approach has two advantages as the following:

- As no new-mode tasks are degraded due to old-mode tasks, it is possible to abstract the influence of old-mode tasks on the new-mode. Hence, the complexity of the timing analysis for an old-mode task can be drastically reduced as it is not required to consider all possible state transition instances that can occur in the new-mode.
- This approach can also greatly reduce the complexity involved in deciding the state of the new-mode during mode transitions.

Consequently, in the schedulability analysis, the value of offsets for new-mode tasks can lead to higher values. In the proposed model, to handle a mode transition, a synchronous protocol without periodicity is considered *i.e.* both old-mode completed tasks and wholly new tasks are allowed to execute simul-

taneously and the periodicity of the unchanged tasks are affected during mode transition. The main advantage of considering mode change protocol without periodicity is that the schedulability of tasksets during mode transition can be improved with appropriate offsets for unchanged tasks. With this approach, it is possible to analyse each mode independently and abstract the influences of tasks belonging to one mode onto the other. Also, there is no need to consider all possible state transition combinations that can occur in both modes simultaneously and therefore there is a significant reduction in the analysis complexity.

5.3.1.4 Handling budget overrun of multiple tasks

To efficiently handle the budget overrun of multiple tasks (to satisfy **O10**), the proposed model considers an approach similar to the CA-MCS model *i.e.* when multiple tasks of the same criticality l overrun their budgets, for a task that is chosen to be degraded is assigned with the minimum degraded budget among all of the possible degraded budgets that can be allocated to it due to the budget overrun of tasks with criticality l. But, when multiple tasks with different criticality overrun their budgets, only the degraded budget corresponding to the task with the highest criticality among all tasks that have overrun their budgets are allocated.

5.3.1.5 Return to normal state

A vast majority of the MCS task models makes an assumption that that once the system transition to the next criticality state, it cannot move back until the idle instant. This assumption reduces the analysis complexity. Therefore, in the MM-MCS model, system is reset to normal state at the idle instant. The way in which the system's criticality level can be reduced safely without waiting for the idle instant can be considered as the potential future work for the proposed CA-MCS and MM-MCS tasks models.

5.3.2 Syntax of the MM-MCS model

A Multi-Mode Mixed Criticality System (MM-MCS) is defined as $S \stackrel{\text{def}}{=} (\Gamma, \mathcal{L}, \mathcal{M}, \Pi)$. $\mathcal{L} = \{1, 2, .., L\}$ denotes an ordered set of criticality levels with a total order supported by S. L and 1 denote the highest and the lowest criticality levels in S respectively. $\mathcal{M}$ is the set of modes in which S can operate and is given as $\mathcal{M} = \{M_1, M_2, ..., M_r\}$. Γ is the set of tasks, and $\Pi \subseteq \mathcal{M}X\mathcal{M}$ is the set of mode transitions of the system. In a system with r modes, M_1 to M_r, a task $\tau_i \in \Gamma$ can be specified

as a set of tuples defined as follows.

$$\tau_i = \{\mathcal{L}_i, \vec{O_i}, \tau_{i,1}, \tau_{i,2}, ..., \tau_{i,r}\}$$

where $\mathcal{L}_i$ denotes criticality of τ_i. $\vec{O_i} = (O_{i,1}, O_{i,2}, ..., O_{i,r})$ denotes a r dimensional vector of offset values where $O_{i,p}$ denotes the maximum permissible delay in the release of τ_i relative to the instant at which a Mode Change Request (MCR) triggering a system transition to M_p happens. Each mode $M_p \in \mathcal{M}$ is associated with a set of tasks $\Gamma_p \subseteq \Gamma$ that are active in mode M_p. In a specific mode M_p, $\tau_{i,p}$ can be given as the following,

$$\tau_{i,p} = \{T_{i,p}, \vec{B_{i,p}}, C^n_{i,p}, C^s_{i,p}, \vec{C_{i,p}}, \vec{\mathcal{T}_{i,p}}\}$$

$T_{i,p}$ (minimum arrival interval) denotes time period of τ_i in M_p. $C^n_{i,p}$ denotes the normal budget of τ_i in M_p. $C^s_{i,p}$ denotes the safe budget of τ_i in M_p. $\vec{C_{i,p}} = (C^1_{i,p}, C^2_{i,p}, ..., C^k_{i,p})$ denotes a vector of degraded budgets of τ_i in M_p. $\vec{B_{i,p}} = (B^1_{i,p}, B^2_{i,p}, ..., B^k_{i,p})$ denotes a vector of behaviours of τ_i in M_p. Each $B^j_{i,p} \in \{Normal, Safe, Degraded\}$ denotes the behaviour of τ_i when $\tau_j \in \Gamma_p$ overruns its normal budget $C^n_{j,p}$. Each $B^j_{i,p}$ is associated with a specific budget $C^j_{i,p}$ such that τ_i is assigned with $C^j_{i,p}$ when $\tau_j \in \Gamma_p$ overruns its normal budget.

Following assumptions are made regarding budgets of τ_i:

1. $0 \leq C^j_{i,p} \leq C^n_{i,p}$, if $B^j_{i,p} = Degraded$.
2. In a mode M_p, τ_i is allowed to overrun only its normal budget ($C^n_{i,p}$).
3. $C^n_{i,p} \leq C^s_{i,p}$.
4. If $\tau_i \notin \Gamma_p$, $C^n_{i,p} = C^s_{i,p} = 0$ and $\forall \tau_j \in \Gamma \setminus \tau_i, C^j_{i,p} = 0$

$\vec{\mathcal{T}_{i,p}} = (t^1_{i,p}, t^2_{i,p}, ..., t^r_{i,p})$ denotes a vector of task types of τ_i. For any $s \in \mathbb{N}$, $t^s_{i,p}$ denotes the task type of $\tau_i \in \Gamma$ when the system undergoes a transition from the old-mode M_p to the new-mode M_s. Therefore, $t^s_{i,p} \in \{\tau_{i(O)}, \tau_{i(A)}, \tau_{i(C)}, \tau_{i(U)}, \tau_{i(W)}\}$. For a given mode transition $(p, s) \in \Pi$, the task type of τ_i can be one of the following:

- **Old mode completed ($\tau_{i(O)}$):** A task τ_i is called an old-mode completed ($\tau_{i(O)}$) of a mode transition $(p, s) \in \Pi$ iff $\tau_i \in \Gamma_p \setminus \Gamma_s$ and when such a task is required to complete its execution even after the occurrence of MCR. Hence, part of its execution may fall after the MCR.

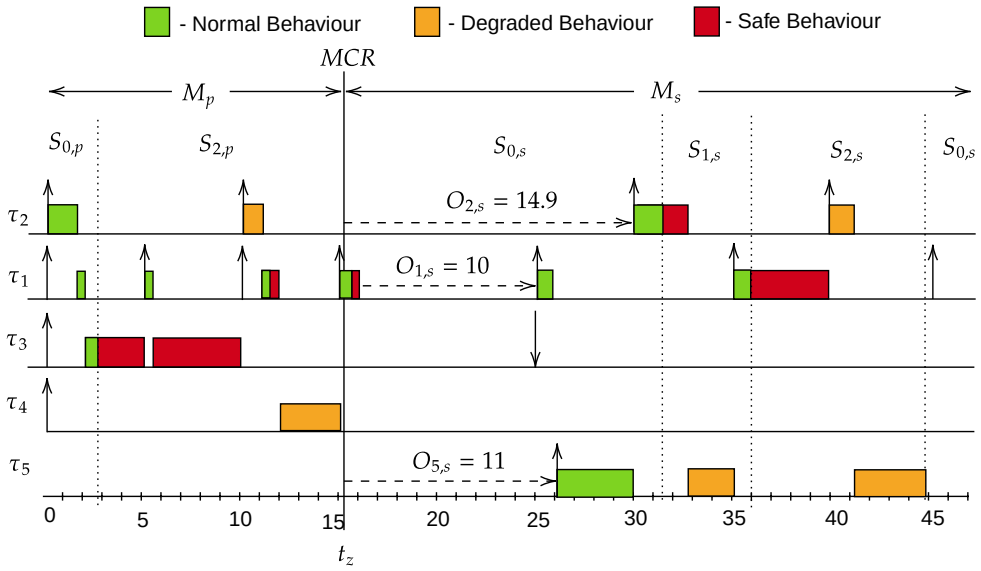

Figure 5.3: Taskset example.

- **Old mode aborted** ($\tau_{i(A)}$): A task τ_i is called an old-mode aborted of a mode transition $(p, s) \in \Pi$ iff $(\tau_i \in \Gamma_p \setminus \Gamma_s)$ and such a task is aborted when the MCR occurs.

On the other side, if τ_i belongs to the new-mode, it may be:

- **New-mode, changed** ($\tau_{i(C)}$): A task τ_i is called a new-mode, changed ($\tau_{i(C)}$) of a mode transition $(p, s) \in \Pi$ if the condition $(\tau_i \in \Gamma_p \cap \Gamma_s) \wedge ((C_{i,p} \neq C_{i,s}) \vee (T_{i,p} \neq T_{i,s}))$ is True.
- **New-mode, unchanged** ($\tau_{i(U)}$): A task τ_i is called a new-mode, unchanged ($\tau_{i(U)}$) of a mode transition $(p, s) \in \Pi$ if the condition $(\tau_i \in \Gamma_p \cap \Gamma_s) \wedge (C_{i,p} = C_{i,s}) \wedge (T_{i,p} = T_{i,s})$ is True.
- **Wholly new task** ($\tau_{i(W)}$): A task τ_i is called wholly new task ($\tau_{i(W)}$) of a mode transition $(p, s) \in \Pi$ if the condition $\tau_i \in \Gamma_s \setminus \Gamma_p$ is True.

We assume that $\mathcal{L}_i$ is same for all modes and its value can be in the range 1 to L where L is the total number of criticality levels in S. To produce a transition from one mode to another, a mode change request (MCR) is assumed to be triggered either by a task or by any external event at any time instant.

Task τ_i	$\mathcal{L}_i$	**Type**	$T_{i,p}$	$P_{i,p}$	**Mode** M_1	
					Budgets	**Behaviour**
τ_1	2	Changed	5	2	$C^n_{1,p}$ = 0.4, $C^s_{1,p}$ = 0.8	$B^1_{1,p} = Safe$ $B^2_{1,p} = Normal$ $B^3_{1,p} = Normal$ $B^4_{1,p} = Normal$
τ_2	1	Unchanged	10	1	$C^n_{2,p} = 1.4$, $C^s_{2,p} = 2.8$, $C^1_{2,p}$ = 1.1 , $C^3_{2,p}$ = 1.0	$B^1_{2,p} = Degraded$, $B^2_{2,p} = Safe$, $B^3_{2,p} = Degraded$, $B^4_{2,p} = Normal$
τ_3	2	Old-mode completed	25	3	$C^n_{3,p} = 0.6$, $C^s_{3,p} = 8.4$	$B^1_{3,p} = Normal$ $B^2_{3,p} = Normal$ $B^4_{3,p} = Safe$ $B^4_{3,p} = Normal$
τ_4	1	Aborted	49	7	$C^n_{4,p} = 3.4$, $C^s_{4,p} = 9.6$, $C^1_{4,p}$ = 2.9, $C^3_{4,p}$ = 2.7	$B^1_{4,p} = Degraded$, $B^2_{4,p} = Normal$, $B^3_{4,p} = Degraded$ $B^4_{4,p} = Safe$
τ_5	2	Not active in this mode				
Task τ_i	$\mathcal{L}_i$	**Type**	$T_{i,s}$	$P_{i,s}$	**Mode** M_2	
					Budgets	**Behaviour**
τ_1	2	Changed	10	5	$C^n_{1,s}$ = 0.8, $C^s_{1,s}$ = 2.0,	$B^1_{1,s} = Safe$ $B^2_{1,s} = Normal$ $B^5_{1,s} = Normal$
τ_2	1	Unchanged	10	4	$C^n_{2,s}$ = 1.4, $C^s_{2,s}$ = 2.8, $C^1_{2,s}$ = 1.2, $C^5_{2,s}$ = 1.0	$B^1_{2,s} = Degraded$, $B^2_{2,s} = Safe$, $B^5_{2,s} = Degraded$,
τ_3	2	Not active in this mode				
τ_4	1					
τ_5	2	Wholly new	70	6	$C^n_{5,s}$ = 10.0, $C^s_{5,s}$ = 25.0, $C^1_{5,s}$ = 10.0 , $C^2_{5,s}$ = 10.0	$B^1_{5,s} = Degraded$, $B^2_{5,s} = Normal$, $B^5_{5,s} = Safe$

Table 5.2: Taskset Details.

5.3.3 An Example to Illustrate the MM-MCS Model

Let us consider a system S that can operate in two different modes denoted as M_p and M_s. Figure 5.3 shows the schedule of execution of tasks when a MCR occurs at t_z leading to the mode transition from M_p to M_s. The system executes 4 tasks (τ_1 to τ_4) in M_p and 3 tasks (τ_1, τ_2 and τ_5) in M_s. The details of parameters of tasks are given in the Table 5.2. $\mathcal{L}_i$ denotes the criticality of τ_i, $P_{i,p}$ and $P_{i,s}$ denote the priorities of τ_i in M_p and M_s respectively with lower numerical value denoting higher priority level. The Fixed Priority Pre-emptive Scheduling (FPPS) scheme is considered for scheduling tasks.

State transitions within the old-mode M_p: The system S starts at $t = 0s$. First releases of τ_1 and τ_2 finish their execution with their normal budgets at $t = 1.4s$ and $t = 1.8s$ respectively. When τ_3, with criticality level equal to 2 overruns its normal budget at $t = 2.4s$, system undergoes a transition from $S_{0,p}$ to $S_{2,p}$. As a result, tasks τ_2 and τ_4 are assigned with their degraded budgets corresponding to τ_3 *i.e.* τ_4 is assigned with $C^3_{4,p}$ (= 2.7) and τ_2 is assigned with $C^3_{2,p}$ (= 1.0). By choosing the degraded budget corresponding to the overrun task τ_3, the observation **O4** (stated in section 4.2) is satisfied.

When τ_1, with criticality level also equal to 2 overruns its normal budget at $t = 11.4s$, the system continues to execute in the state $S_{2,p}$. τ_4 is not updated with the degraded budget corresponding to τ_1 (which is 2.9) since min($C^1_{4,p}$, $C^3_{4,p}$) = 2.7. In this step, when multiple tasks of same criticality have overrun their budget, minimum degraded budget corresponding to the overrun tasks is chosen for other degrading tasks. Therefore, an increased system overload due to budget overrun of both τ_1 and τ_2 can be compensated by choosing the degraded budget which is the minimum value of the degraded budget of τ_4 corresponding to degraded budgets of τ_1 and τ_2. By doing this, the model need not consider all possible combinations of task overruns and the choice of degraded budget chosen for each overrun combination thereby satisfying observations **O5** and **O10**.

Transition from the old-mode M_p to the new-mode M_s: The task type of τ_4 is aborted. Therefore, when a MCR triggering the mode transition from the old-mode M_p to the new-mode M_s occurs at the time instance $t_z = 15.1s$, τ_4 gets aborted immediately at $t = 15.1s$ and not allowed to complete its remaining execution of 0.2 units.

When τ_1 that was released in M_p at $t = 15.0s$ overruns its normal budget at $t = 15.4s$, the system's state in the new-mode M_s is not affected and τ_1 is allowed to finish executing its safe budget. This step is done to satisfy **O9**. By doing this, budget overrun of τ_1 has no degrading effect on the tasks released in the new-mode M_s. The system becomes idle at $t = 15.8s$. In Figure 5.3, new-mode releases of an unchanged task τ_2, changed task τ_1 and wholly new task τ_5 happen after an offset relative to t_z.

Therefore, the first release of τ_1 and τ_5 occur at $t = 25.1s$ and $t = 26.1s$ respectively and that of τ_2 occur at $t = 30s$ as its offset value was considered to be 14.9. This step satisfies **O8**.

The budget overrun of τ_2 at $t = 31.4s$ result in the change in its behaviour from 'Normal' to 'Safe' leading to a system transition from $S_{0,s}$ to $S_{1,s}$. τ_5 despite being higher critical than τ_2, is assigned with its degraded budget to handle the budget overrun. Although the value of this degraded budget of τ_5 is same as that of its normal budget, the only difference is that τ_5 is allowed to overrun its normal budget but not its degraded budget.

When τ_1 overruns its budget at $t = 35.9s$, the state transition from $S_{1,s}$ to $S_{2,s}$ is triggered. At this instance, although the behaviour of τ_2 is 'Safe', it is immediately updated with the degraded budget equal to 1.2 (= $C^1_{2,s}$) as the behaviour $B^1_{2,s}$ is 'Degraded'. As τ_5 finishes its execution, the system becomes idle and return to the normal state $S_{0,s}$ at $t = 45.0s$.

5.3.4 Semantics of the MM-MCS model

At any time $t \geq 0$, S can exist in any mode in $\mathcal{M} = \{M_1, M_2, ..., M_r\}$. In any given mode $M_p \in \mathcal{M}$, depending on the run-time behaviour of tasks, S can exist in any state belonging to the set $\mathcal{ST}_p = \{S_{0,p}, S_{1,p}, S_{2,p}, ..., S_{L,p}\}$ where L denotes the maximum criticality level supported by the system S. $S_{0,p}$ denotes normal state and $S_{1,p}, S_{2,p}, ..., S_{L,p}$ denote safe states in M_p. Let the system start (at t = 0) in the normal state $S_{0,p}$ in M_p.

Conditions to determine the system's state:

Let t_x be the current time instant and let $t_{idle} < t_x$ be the latest time instant at which system experienced an idle instant in M_p. Let e_i^{max} denote the largest execution time of any job of $\tau_i \in \Gamma_p$ at t_x since t_{idle}. S is said to be in the normal state $S_{0,p}$ at t_x, if the condition $\forall \tau_i \in \Gamma_p$, its $e_i^{max} \leq C^n_{i,p}$ is satisfied. In other words, S is said to be in the normal state $S_{0,p}$ at t_x if no tasks have overrun their normal budget in the interval $t_{idle} < t \leq t_x$.

Let $l = max(\mathcal{L}_i), \forall \tau_i \in \Gamma^l_p$ where $\Gamma^l_p \subseteq \Gamma_p$ denotes a set of tasks belonging to M_p that satisfy the condition $C^n_{i,p} < e_i^{max} \leq C^s_{i,p}, \forall \tau_i \in \Gamma_p$ at t_x. S is said to be in the safe state $S_{l,p} \in \mathcal{ST}_p \setminus S_{0,p}$ at t_x if $\Gamma^l_p \neq \emptyset$.

In a given mode M_p, a state transition in S can happen from $S_{m,p}$ to $S_{n,p}$ only if $n > m$ in the interval $t_{idle} < t \leq t_x$.

Transition from one state to another state (within a mode):

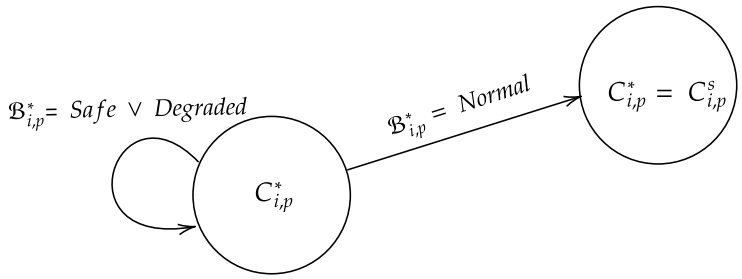

Figure 5.4: Change in the value of $C^*_{i,p}$ for different values of $B^*_{i,p}$ when a job of τ_i has executed for C^*_i without signalling its completion at t_x.

Let the current time instant be t_x and the current system state be $S_{l,p} \in \mathcal{ST}_p$. $\forall \tau_i \in \Gamma$, let $C^*_{i,p}$ denote the current budget allocated to τ_i and let $B^*_{i,p}$ denote the current behaviour of τ_i. Let e^*_i denote the time for which an active job of τ_i has executed till t_x. Let $J_{i,p}$ denote a job of τ_i that has executed for $C^*_{i,p}$ without signalling its completion (budget overrun) at t_x. The current budget allocated for τ_i (*i.e.* $C^*_{j,p}$) at t_x depends on the value of $B^*_{i,p}$ as shown in Figure 5.4. Depending on on the value of $B^*_{i,p}$ at t_x, $C^*_{i,p}$ is updated as the following:

1. If $B^*_{i,p}$ = $Normal$, the current budget of τ_i is immediately updated as $C^*_{i,p} = C^s_{i,p}$. $B^*_{i,p}$ is immediately updated as $B^*_{i,p} = Safe$.
2. If $B^*_{i,p} = Safe \vee Degraded$, $J_{i,p}$ is terminated immediately without affecting budgets of other tasks.

$\forall \tau_j \in \Gamma_p \setminus \tau_i$, the budget allocated for τ_j at t_x depends on the value of $\mathcal{L}_i$ as shown in Figure 5.5. Hence, the following three cases are considered.

Case 1: $\mathcal{L}_i = l$

$\forall \tau_j \in \Gamma_p \setminus \tau_i$, if $B^i_{j,p} = Degraded$, three cases are possible depending on the value of e^*_j :

1a. If $\mathcal{L}_j \leq \mathcal{L}_i$ and $e^*_j \geq C^i_{j,p}$, τ_j is assumed to be finished. The subsequent releases of τ_j are executed with $\min(C^*_{j,p}, C^i_{j,p})$.

1b. If $\mathcal{L}_j \leq \mathcal{L}_i$ and $e^*_j < C^i_{j,p}$, then $C^*_{j,p}$ is immediately updated as $C^*_{j,p} = \min(C^*_{j,p}, C^i_{j,p})$.

1c. If $\mathcal{L}_j > \mathcal{L}_i$, only the subsequent releases of τ_j are executed with $\min(C^*_{j,p}, C^i_{j,p})$.

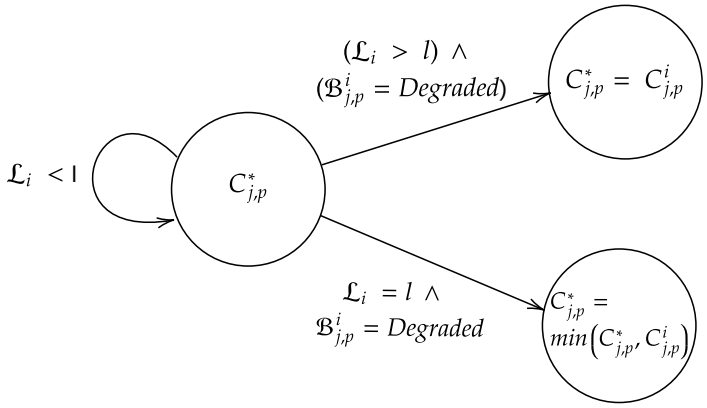

Figure 5.5: Change in the value of $C^*_{j,p}$ for different values of $\mathcal{L}_i$ and $B^i_{j,p}$ when a job of τ_i has executed for $C^*_{i,p}$ without signalling its completion at t_x.

Case 2: $\mathcal{L}_i > l$

A state transition from $\mathcal{S}_{l,p}$ to $\mathcal{S}_{\mathcal{L}_i,p}$ is immediately done. If $B^i_{j,p} = Degraded$, three cases are possible depending on the value of e^*_j :

2a. If $\mathcal{L}_j \leq \mathcal{L}_i$ and $e^*_j \geq C^i_{j,p}$, τ_j is assumed to be finished. The subsequent releases of τ_j are executed with $C^i_{j,p}$.

2b. If $\mathcal{L}_j \leq \mathcal{L}_i$ and $e^*_j < C^i_{j,p}$, $C^*_{j,p}$ is immediately updated as $C^*_{j,p} = C^i_{j,p}$.

2c. If $\mathcal{L}_j > \mathcal{L}_i$, only the subsequent releases of τ_j are executed with $C^j_{j,p}$.

Case 3: If $\mathcal{L}_i < l$, no other tasks are affected.

Transition from one mode to another (Mode Change Protocol):

Let the MCR requesting a mode transition from M_p to M_s arrive at time t_z. Let $t_k < t_z$ denote the latest time instant at which system has undergone a state transition to any state $S_{k,p} \in \mathcal{ST}_p$ in M_p. Consider a job $J_{i,p}$ of τ_i released at $t_r < t_z$ still active at t_z. Let $C^*_{i,p}$ denote the current budget of τ_i that was allocated to it at t_k in M_p. Let $B^*_{i,p}$ denote the current behaviour of τ_i that was allocated to it at t_k in M_p.

Upon the occurrence of MCR, the protocol is as follows: The system is reset to $S_{0,s}$ *i.e.* to the normal state of the new-mode mode M_s, immediately at t_z. $\forall \tau_i \in \Gamma_s$, the first release of τ_i in M_s happens at

$t_z + O_{i,s}$. $\forall \tau_i \in \Gamma_p$, if $t^s_{i,p} = \tau_{i(A)}$ *i.e.* if the task type of τ_i is aborted then τ_i is terminated at t_z. Otherwise, $J_{i,p}$ is allowed to execute to finish $C^*_{i,p}$.

Handling budget overrun of old-mode tasks in the new-mode:

Let the current time instant be $t_x > t_z$. Let $J_{i,p}$ execute for $C^*_{i,p}$ without signalling its completion at t_x *i.e.* budget overrun of an old-mode task τ_i detected at t_z. Depending upon the value of $B^*_{i,p}$, two cases are possible.

1. If $B^*_{i,p}$ = $Safe \vee Degraded$, $J_{i,p}$ is terminated and not allowed to continue. No other tasks are affected.
2. If $B^*_{i,p}$ = $Normal$, $J_{i,p}$ is immediately updated with $C^s_{i,p}$ which is the safe budget corresponding to mode M_p. No other tasks are affected. $B^*_{i,p}$ is immediately updated as $B^*_{i,p} = Safe$.

In a given mode, the system is reset to normal state at the idle instant.

Handling multiple mode and state transitions:

It is assumed that MCRs may arrive at any time; however, to ensure that jobs of tasks belonging to not more than two different modes can interfere, we assume that consecutive MCRs are separated by at least twice the maximum period among all tasks in all modes. It is assumed that mode transitions are non-deterministic *i.e.* if multiple transitions are enabled at the same time, the system selects any one in a non-deterministic manner.

It is assumed that multiple state transitions cannot occur at the same time. This assumption is valid for the uniprocessor system as the scenario where the trigger to multiple state transitions due to the simultaneous budget overrun multiple tasks with same or different criticality cannot happen as only one task is allowed to execute at a time. It is assumed that when trigger to both state transition and mode transition occurs at the same time, the state transition is assumed to have higher priority than the mode transition *i.e.* budget for tasks are first determined based on the state transition semantics and then immediately followed by the decisions corresponding to mode transition semantics. By giving higher priority to state transition semantics, the decisions corresponding to tasks (only old-mode tasks are active at this time) such as updating their budgets are performed first and only then decisions corresponding to mode transition such as aborting a task, determining the release time for new-mode tasks are performed.

As the proposed MM-MCS model considers many parameters, in Table 5.3, list of symbols and their meanings used in syntax and semantics of the MM-MCS model are tabulated for the reference purpose.

Symbols	Meaning
S	a multi-mode mixed criticality system.
Γ	a set of tasks in S.
$\mathcal{ST}_p$	a set of system states in M_p.
t_{idle}	time instant at which S is idle.
Γ_p	set of tasks belonging to the mode M_p in S.
$\mathcal{L}$	a set of criticality levels supported by S.
$\mathcal{M}$	a set of modes in which S can operate.
Π	a set of mode transitions in S.
τ_i	a task in S.
M_p	a mode in which S can operate.
$\mathcal{L}_i$	criticality of τ_i.
$\vec{O_i}$	vector of offset values of τ_i.
$O_{i,p}$	the maximum permissible delay in the release of τ_i relative to the instant at which a Mode Change Request (MCR) triggering a system transition to M_p happens
t_z	a mode transition time instant.
$O_{i,p}$	the maximum permissible delay in the release of τ_i relative to t_z in M_p.
$T_{i,p}$	time period of τ_i in M_p.
$\vec{B_{i,p}}$	a vector of behaviours of τ_i in M_p.
$C^n_{i,p}$	normal budget of τ_i in M_p.
$C^s_{i,p}$	safe budget of τ_i in M_p.
$\vec{C_{i,p}}$	a vector of degraded budgets of τ_i in M_p.
$\vec{\mathcal{T}_{i,p}}$	a vector of task types of τ_i when the system undergoes a transition from the old-mode M_p to the new-mode M_s .
$C^j_{i,p}$	a degraded budget assigned to τ_i when $\tau_j \in \Gamma_p \setminus \tau_i$ overruns its normal budget.
t_{idle}	time instant at which S is idle.
$\mathcal{P}_{p,z}$	a set of valid state transition patterns that can occur in the interval $[0, t_z]$ in M_p.
$t_{l,p}$	time instant at which state transition to the safe state $S_{l,p}$.

Table 5.3: Symbols and their meanings used in the proposed MM-MCS model.

5.4 Schedulability Analysis

In this section, the Fixed Priority Pre-emptive Scheduling scheme for the MM-MCS model comprising a set of tasks with an implicit deadline (*i.e.* deadline equal to period) which execute on a single processor is presented. The only resource that is shared among the tasks is the processor. Overheads due to context switches and the operation of the scheduler are assumed to have been included as a part of their budgets. In a given mode, it is assumed each task can generate a sequence of infinite number of jobs that are separated by a specific minimum arrival interval. In a given mode, the system is defined for a normal state and L safe states of operation. It is also assumed that the taskset is fully defined and for each task, their ids, priorities, type and indices of their budgets are assigned apriori. The worst-case response time (WCRT) of a task τ_i gives the maximum possible time that any job of τ_i can take to complete. A task is schedulable if it's WCRT does not exceed its deadline. Therefore, for τ_i, all possible mode and state transition combinations that can occur in an interval of length equal to its time period need to be considered to determine its WCRT.

Those tasks released in the old-mode but still active in the new-mode are referred as old-mode tasks while those tasks released in the new-mode with deadline also in the new-mode are referred as new-mode tasks. As the steps involved in finding WCRT for an old-mode task is different from that for a new-mode task, their WCRTs are computed separately.

5.4.1 Steps To Compute the Worst-Case Response Time of an Old-Mode Task

Let t_z denote a time instance at which a mode transition from an old-mode M_p to a new-mode M_s is triggered. The amount of interference from higher priority tasks in a given mode depends on the length of time intervals of each safe states corresponding to that mode. Hence, the worst-case response time of an old-mode task τ_i is the maximum value of its response time which is computed by considering interference from both old- and new-mode tasks for all possible t_z and state transition combinations that can occur in M_p and M_s. For each t_z, the maximum response time for τ_i in the old-mode is found by considering all possible valid patterns of state transition instances that can occur in the old-mode. Therefore, for each t_z, in the range $[0, T_{i,p}]$, $\mathcal{P}_{p,z} = \{P^1_{p,z}, P^2_{p,z}, ..P^m_{p,z}\}$ denotes a set of valid state transition patterns that can occur in the interval $[0, t_z]$ in M_p. $P^k_{p,z} = [t_{1,p}, t_{2,p}, ..., t_{L,p}, t_z]$ denotes an array of state transition time instances such that $0 \leq t_{1,p} \leq t_{2,p} \leq ... \leq t_{l,p} \leq t_z$ where $t_{l,p}$ denotes transition instance to the safe state $S_{l,p}$. In the following, various steps involved to compute worst-case response time of an old-mode task τ_i when the system transition happens from M_p and M_s are

discussed.

Step 1: As a first step, the release patterns of tasks that maximizes the interference for an old-mode task τ_i for any given t_z and $P^l_{p,z}$ are found. This is required to compute the maximum number of jobs from each interfering task to τ_i in M_p. In subsection 5.4.3, Corollaries 3 and 4 discuss such release patterns under normal and budget overrun conditions.

Step 2: Due to the complexity of the MM-MCS model, the analysis is performed in a structured approach in which for each interfering task τ_j, time instances of a specific valid transition pattern $P^l_{p,z}$ are separated into 3 sets of time intervals based on the τ_j's behaviour (normal, safe and degraded). Interference is analyzed by abstracting the influences from other time intervals of the same or different modes. Depending on the safe state considered, jobs of an interfering task can interfere with their normal, safe or degraded budgets. For the release patterns discussed in Corollaries 3 and 4, the maximum number of jobs that can interfere with normal, safe and degraded budgets need to be found to compute the interference from interfering tasks. For this purpose, equations 5.1, 5.2 and 5.3 are presented in the subsection 5.4.3.

Step 3: From the semantics of the MM-MCS model, it is clear that a task can be allocated with any one of its degraded budgets depending on the overrun tasks. Hence, for the WCRT analysis for τ_i, it is required to determine the maximum possible degraded budget that can be allocated to an interfering task τ_j in each safe state. In the following, we present definitions for terms used to find the maximum value of the degraded budget that can be allocated to an interfering task τ_j in each safe state in both old-mode and new-mode. Let Γ_p and Γ_s denote two independent tasksets consisting of tasks corresponding to modes M_p and M_s respectively.

Definition 16 *For a task $\tau_j \in \Gamma_p$,*

$$DH_{j,p} = \begin{cases} 1, & \forall \tau_k \in \Gamma_p \setminus \tau_j, (\mathcal{L}_k > \mathcal{L}_j) \wedge B^k_{j,p} = Degraded \\ 0, & Otherwise \end{cases}$$

In the above expression, the value of $DH_{j,p}$ is set to 1 if the behaviour of τ_j in M_p is such that it is always degraded when any task belonging to mode M_p with criticality level greater than $\mathcal{L}_j$ overruns its normal budget. Otherwise, $DH_{j,p}$ is considered to be 0.

Definition 17 *For a task $\tau_j \in \Gamma_p$,*

$$DL_{j,p} = \begin{cases} 1, & \forall \tau_k \in \Gamma_p \setminus \tau_j, (\mathcal{L}_k < \mathcal{L}_j) \wedge B^k_{j,p} = Degraded \\ 0, & Otherwise \end{cases}$$

Similarly, the value of $DL_{j,p}$ is set to 1 if the behaviour of τ_j in M_p is such that it is always degraded when any task with criticality level lower than $\mathcal{L}_j$ overruns its normal budget. Otherwise, $DL_{j,p}$ is considered to be 0. The values of $DH_{j,p}$ and $DL_{j,p}$ are derived from $B_{j,p}$ and can be used to determine the maximum budget with which τ_j can interfere in any given safe state belonging to the mode M_p. The values of $DH_{j,p}$ and $DL_{j,p}$ are derived from $B_{j,p}$ and can be used to determine the maximum budget with which τ_j can interfere in any given safe state (except $S_{\mathcal{L}_{j,p}}$). In the safe state $S_{\mathcal{L}_{j,p}}$, τ_j can interfere with its safe budget ($C^s_{j,p}$). This is possible when the system undergoes state transition to $S_{\mathcal{L}_{j,p}}$ due to the budget overrun of τ_j which can lead to degradation of other tasks with criticality level same as that of τ_j. As the semantics of the MM-MCS model does not allow tasks to overrun their degraded budget, τ_j can continue to execute with its safe budget in $S_{\mathcal{L}_{j,p}}$ without getting degraded. Hence, the expressions for computing $DL_{j,p}$ and $DH_{j,p}$ do not consider tasks with criticality level same as that of τ_j.

Definition 18 *$maxB(\tau_j, l, M_p)$ returns the maximum budget that can possibly be assigned to τ_j in the state $S_{l,p}$ belonging to the mode M_p due to the budget overrun of any task $\tau_k \in \Gamma_p \setminus \tau_j$ with $\mathcal{L}_k = l$.*

$$maxB(\tau_j, l, M_p) = \begin{cases} C^n_{j,p}, & (l = 0) \vee ((l < \mathcal{L}_j) \wedge (DL_{j,p} = 0)) \\ C^s_{j,p}, & (l = \mathcal{L}_j) \vee ((l > \mathcal{L}_j) \wedge (DH_{j,p} = 0)) \\ \max\limits_{\substack{\forall \tau_k \in \Gamma_p \setminus \tau_j, \\ \mathcal{L}_k = l}} C^k_{j,p}, & Otherwise \end{cases}$$

Step 4: In the next step, the worst-case response time for τ_i considering interference from jobs of τ_j released in the old-mode M_p for a given t_z and for a given $P^l_{p,z} \in \mathcal{P}_{p,z}$ is computed using Equation 5.7. This value is denoted as $R^{l,z}_{i,p}$. Subsequently, the maximum value among $R^{l,z}_{i,p}$ computed for each $P^l_{p,z} \in \mathcal{P}_{p,z}$ is denoted as $R^z_{i,p}$ can be found using Equation 5.9.

Step 5: For a given t_z, if the value of $R^z_{i,p}$ is greater than t_z, then the worst-case response time for τ_i is extended to include interference in the new-mode M_s. For this, all possible valid patterns of state transition instances that can occur to τ_i in the new-mode M_s is considered. Therefore, to analyse the interference in M_s, let $\mathcal{P}_{s,z} = \{P^1_{s,z}, P^2_{s,z}, ..P^m_{s,z}\}$ denote a set of valid state transition patterns that can occur in the interval $[t_z, T_{i,p}]$ in M_s. $P^k_{s,z} = [t_z, t_{1,s}, t_{2,s}, ..., t_{L,s}]$ denotes an array of state transition

time instances such that $t_z < t_{1,s} \leq t_{2,s} \leq ... \leq t_{l,s}$ where $t_{l,s}$ denotes transition instance to the safe state $S_{l,s}$. It is assumed that for any l, $t_z < t_{l,s}$ since it is not possible for any new-mode task to overrun its normal budget at t_z.

Step 6: Steps 2 and 3 need to be repeated for M_s *i.e.* for a given t_z and $P^l_{s,z}$, release patterns of tasks that maximizes interference for τ_i in the new-mode M_s is required. Theorem 3 presented in subsection 5.4.3 establishes these patterns under normal conditions (no budget overrun). Theorem 4 presented in subsection 5.4.3 establishes these conditions under budget overrun conditions. Similarly, equations 5.4, 5.5 and 5.6 presented in subsection 5.4.3 are used to find the maximum number of jobs of τ_j that can interfere with normal, safe and degraded budgets respectively. Again, the maximum value of the degraded budget that can be allocated to an interfering task τ_j in each safe state in the new-mode M_s can be found using definitions 16, 17 and 18.

Step 7: Next, the worst-case response time for an old-mode task τ_i considering interference from jobs of τ_j released in the new-mode M_s for a given t_z and for a specific state transition pattern $P^k_{p,s} \in \mathcal{P}_{s,z}$. $R^{k,z}_{i,s}$ can be found using Equation 5.8.

Step 8: Finally, the worst-case response time for τ_i denoted as R^o_i is the maximum value of $R^{k,z}_{i,s}$ considering all possible t_z, for all $P^k_{p,z} \in \mathcal{P}_{p,z}$ and for all $P^k_{s,z} \in \mathcal{P}_{s,z}$. R^o_i can be found using Equation 5.10.

5.4.2 Steps To Compute the Worst-Case Response Time of a New-Mode Task

A new-mode task has its release time and deadline both in the new-mode. The worst-case response time of a new-mode task τ_i is denoted as R^n_i. To compute R^n_i, two kinds of analysis must be considered. In the first analysis, the upper bound interference from old-mode tasks as well as the interference from new-mode tasks should be considered. In this analysis, offsets for τ_i as well for each interfering task τ_j are considered. However, for the higher offset value of τ_i, it is possible that all old-mode tasks can finish their execution even before the release of τ_i. Hence, in the second analysis, the upper bound interference for τ_i without considering offsets for new-mode tasks nor the interference from old-mode tasks is considered. To compute R^n_i, the following steps are followed:

Step 1: As a first step, the maximum possible interference for a new-mode task $\tau_i \in \Gamma_s$ that can occur from old-mode tasks is computed. For this, all possible modes from which system transition can happen to a new-mode M_s can happen are considered. It is denoted as $I^o_{i,s}$. At worst-case, for a given old-mode

M_p, $I^o_{i,s}$ is the sum of the interference from a single job of an interfering task executing with safe budget released in the old-mode M_p with deadline in M_s. $I^o_{i,s}$ is found using Equation 5.11. As $I^o_{i,s}$ denotes the maximum interference from old-mode tasks, there is no need to consider different values of t_z. So, to compute interference from new-mode tasks, it is enough to consider all possible valid patterns of state transitions that can occur in the new-mode M_s for $t_z = 0$. Equation 5.13 is used for this computation.

Step 2: For higher offset values for τ_i, equation 5.13 may produce a negative value. In order to accommodate the analysis for higher offset values of τ_i, the WCRT for τ_i is computed using equation 4.5 presented in section 4.4 where neither offset nor interference from old-mode tasks are considered. However, the analysis is done considering all possible state transition combinations that can occur to τ_i in the interval $[0, T_{i,s}]$.

Step 3: Finally, the WCRT for τ_i in the new-mode M_s, denoted as R^n_i, is the maximum value among the results found in steps 1 and 2.

In the following, we provide corollaries and theorems that are used to derive the equations for fixed-priority based sufficient schedulability test for old- and new-mode tasks on pre-emptive uniprocessors for the MM-MCS model.

5.4.3 Worst-Case Scenarios

In the following corollaries, it is shown that the release pattern of any higher priority task τ_j that can maximize its interference to τ_i in the old-mode M_p depends on the valid state transition pattern considered, values of $DH_{j,p}$, $DL_{j,p}$ and on the length of the time interval considered. The following corollary considers the conditions that can maximize the interference from τ_j when both τ_i and τ_j are requested simultaneously.

Corollary 3 *For a given mode transition from the old-mode M_p to the new-mode M_s occurring at t_z, for $t > 0$, for any given valid state transition pattern in M_p, interference in interval [0, t] for a job of any old-mode task τ_i from the releases of a higher priority task τ_j occurring in M_p is maximized when τ_i is requested simultaneously with the request for τ_j under any of the following conditions,*

- *Condition A1 - for any $0 < t \leq t_{\mathcal{L}_{j,p}}$*
- *Condition A2 - for any t in the range $t_{\mathcal{L}_{j,p}} < t$ and if $DH_{j,p} = DL_{j,p} = 1$ and **for any** k in the range $1 \leq k < \mathcal{L}_j$, $0 < t_{k,p} < t_{\mathcal{L}_j,p}$*

Proof: The proof is similar to the proof discussed for Theorem 1 discussed in section 4.4. In Theorem 1, to compute the interference for τ_i, time instances of each valid state transition pattern are generated in the interval [0, T_i]. Similarly, to compute the interference for an old-mode task τ_i from higher priority tasks released in the old-mode M_p, time instances of each valid state transition pattern are generated in the interval [0, t_z]. Further, only those releases of higher priority tasks that occur in the old-mode M_p are considered. Therefore, t_z has no impact on the release pattern and on the conditions considered in Theorem 1 that will maximize the interference for τ_i from any higher priority task τ_j. ■

The following corollary considers the release pattern of τ_j that can maximize the interference from τ_j to τ_i when τ_j overruns its normal budget.

Corollary 4 *For a given mode transition from the old-mode M_p to the new-mode M_s occurring at t_z, for $t > 0$, for any given valid state transition pattern in the M_p, interference in interval [0, t] for a job of any old-mode task τ_i from the releases of a higher priority task τ_j occurring in M_p is maximized when the budget overrun of τ_j occurs at $t_{\mathcal{L}_{j,p}}$ under any of the following conditions,*

- *Condition A3 - for any $t_{\mathcal{L}_{j,p}} < t$ and $DH_{j,p} = 0$*
- *Condition A4 - for any $t_{\mathcal{L}_{j,p}} < t$ and $DH_{j,p} = 1$ and $DL_{j,p} = 0$*
- *Condition A5 - for any $t_{\mathcal{L}_{j,p}} < t$ and if $DH_{j,p} = DL_{j,p} = 1$ and **for all** k in the range $1 \leq k < \mathcal{L}_j$, $t_{k,p} = t_{\mathcal{L}_j,p}$*

Proof: The proof is similar to the proof discussed for Theorem 2 discussed in section 4.4. In Theorem 2, to compute the interference for τ_i, time instances of each valid state transition pattern are generated in the interval [0, T_i]. Similarly, to compute the interference for an old-mode task τ_i from higher priority tasks released in the old-mode M_p, time instances of each valid state transition pattern are generated in the interval [0, t_z]. Further, only those releases of higher priority tasks that occur in the old-mode M_p are considered. Therefore, t_z has no impact on the release pattern and on the conditions considered in Theorem 2 that will maximize the interference for τ_i from any higher priority task τ_j. ■

In the following, expressions to find the upper bound number of releases of τ_j interfering with safe budget (denoted as $N_s^{old}(j, t, P_{p,z}^l)$), normal budget (denoted as $N_n^{old}(j, t, P_{p,z}^l)$) and degraded budgets (denoted as $N_d^{old}(j, t, P_{p,z}^l)$) for a specific valid transition pattern $P_{p,z}^l \in \mathcal{P}_{p,z}$, in the old-mode M_p based on corollaries 3 and 4 are presented.

Expression for $N_s^{old}(j, t, P_{p,z}^l)$:

$$N_s^{old}(j,t,P_{p,z}^l) = \begin{cases} 0, & A1 \vee A2 \\ \min\left\{1+\left\lceil\frac{min(t,t_z)-t_{\mathcal{L}_{j,p}}}{T_{j,p}}\right\rceil_0, \left\lceil\frac{min(t,t_z)}{T_{j,p}}\right\rceil\right\}, & A3 \\ \min\left\{1+\left\lceil\frac{min(t,t_z,t_{\mathcal{L}_j+1,p})-t_{\mathcal{L}_{j,p}}}{T_{j,p}}\right\rceil_0, \left\lceil\frac{min(t,t_z)}{T_{j,p}}\right\rceil\right\}, & A4 \vee A5 \end{cases} \tag{5.1}$$

For any time t in the range $0 < t \leq t_{\mathcal{L}_{j,p}}$ as considered by the conditions $A1$ and $A2$, no job of τ_j can execute with its safe budget. Hence, the value of $N_s^{old}(j,t,P_{p,z}^l)$ is 0. Condition $A3$ considers the value of $DH_{j,p}$ to be 0, and therefore the maximum number of releases of τ_j that can interfere with safe budget and can fit into an interval of length $min(t,t_z) - t_{\mathcal{L}_{j,p}}$ is

$$1+\left\lceil\frac{min(t,t_z)-t_{\mathcal{L}_{j,p}}}{T_{j,p}}\right\rceil_0$$

For smaller values of $t_{\mathcal{L}_{j,p}}$, the above expression can be pessimistic when considering the number of jobs that can fit in an interval of length t. Therefore, the value of $N_s^{old}(j,t,P_{p,z}^l)$ is given as

$$\min\left\{1+\left\lceil\frac{min(t,t_z)-t_{\mathcal{L}_{j,p}}}{T_{j,p}}\right\rceil_0, \left\lceil\frac{min(t,t_z)}{T_{j,p}}\right\rceil\right\}$$

For conditions $A4$ and $A5$, the value of $DH_{j,p}$ is considered to be 1 and therefore the expression corresponding to these conditions upper bounds the number of jobs of τ_j that can interfere with safe budget in the interval of length $min(t,t_z,t_{\mathcal{L}_{j+1,p}})$ - $t_{\mathcal{L}_{j,p}}$. For smaller values of $t_{\mathcal{L}_{j,p}}$, only the interval of length $min(t,t_z,t_{\mathcal{L}_{j+1,p}})$ is considered to reduce the pessimism.

Expression for $N_n^{old}(j,t,P_{p,z}^l)$:

$$N_n^{old}(j,t,P_{p,z}^l) = \begin{cases} \left\lceil\frac{min(t,t_z)}{T_{j,p}}\right\rceil - N_s^{old}(j,t,P_{p,z}^l), & A3 \\ \left\lceil\frac{min(t,t_z,t_{\mathcal{L}_j+1,p})}{T_{j,p}}\right\rceil - N_s^{old}(j,t,P_{p,z}^l), & A4 \vee A5 \\ \left\lceil\frac{min(t,t_z,t_{1,p})}{T_{j,p}}\right\rceil, & A2 \vee (A1 \wedge DL_{j,p}=1) \\ \left\lceil\frac{min(t,t_z)}{T_{j,p}}\right\rceil, & A1 \wedge DL_{j,p}=0 \end{cases} \tag{5.2}$$

As discussed in [19], the upper bound number of releases of τ_j that can interfere τ_i in an interval of length t is given as $\left\lceil\frac{t}{T_{j,p}}\right\rceil$. For condition $A1$, if $DL_{j,p} = 0$ then all jobs of τ_j released in the interval [0,

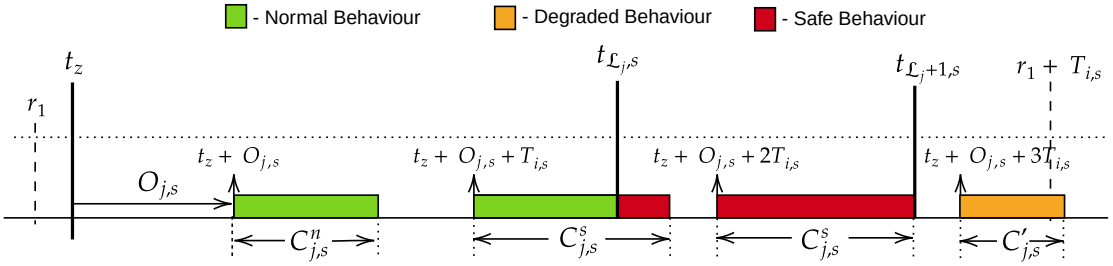

Figure 5.6: Release pattern of τ_j with $DH_{j,s} = 1$.

$t]$ in M_p interfere with normal budget and the value of $N_n^{old}(j,t,P_{p,z}^l)$ is given as $\left\lceil \frac{min(t,t_z)}{T_{j,p}} \right\rceil$. $min(t,t_z)$ is used because only the interference from the jobs of τ_j released in the old-mode M_p is considered. If $DL_{j,p} = 1$ then for conditions $A1$ and $A2$, jobs of τ_j can interfere with normal budget only in the interval [0, $t_{1,p}$] and the value of $N_n^{old}(j,t,P_{p,z}^l)$ is given as $\left\lceil \frac{min(t,t_z,t_{1,p})}{T_{j,p}} \right\rceil$ where $t_{1,p}$ denotes the time instant at which system transition to the safe state $S_{1,p}$ happens. For the condition $A3$, if the value of $DL_{j,p}$ is equal to 0 then jobs of τ_j can interfere with normal budget only in the interval [0, $t_{\mathcal{L}_{j,p}}$]. Hence, the value of $N_n^{old}(j,t,P_{p,z}^l)$ is found by subtracting $N_s^{old}(j,t,P_{p,z}^l)$ from the total number of jobs that can be released in the interval of length $min(t,t_z)$. Conditions $A4$ and $A5$ consider the value of $DH_{j,p}$ to be 1. Therefore, the value of $N_n^{old}(j,t,P_{p,z}^l)$ is found by subtracting $N_s^{old}(j,t,P_{p,z}^l)$ from the total number of jobs that can be released in the interval of length $min(t,t_z,t_{\mathcal{L}_{j+1,p}})$.

Expression for $N_d^{old}(j,t,P_{p,z}^l)$:

Once $N_s^{old}(j,t,P_{p,z}^l)$ and $N_n^{old}(j,t,P_{p,z}^l)$ are found, remaining releases of τ_j can be considered to interfere with the degraded budgets and is given by the following expression,

$$N_d^{old}(j,t,P_{p,z}^l) = \left\lceil \frac{min(t,t_z)}{T_{j,p}} \right\rceil - (N_s^{old}(j,t,P_{p,z}^l) + N_n^{old}(j,t,P_{p,z}^l)) \tag{5.3}$$

In the following theorems, the release pattern and the conditions under which the interference from any higher priority task τ_j to τ_i in the new-mode M_s is maximized are considered. For this, two cases are considered depending on the release time of τ_i. First, in Theorem 3, release time of τ_i is considered to be earlier than $t_z + O_{j,s}$. Second, in Theorem 4, release time of τ_i at or after $t_z + O_{j,s}$ is considered.

Theorem 3 *Let τ_j denote a new-mode task with priority higher than τ_i. Then, for a mode transition from the old-mode M_p to the new-mode M_s occurring at t_z, for any given valid state transition pattern in M_s, for a task τ_i with its release time at r_1 occurring either in M_p or M_s such that $0 \leq r_1 < t_z + O_{j,s}$,*

for any $t > t_z + O_{j,s}$, the interference in interval [$t_z + O_{j,s}$, t] for a job of τ_i from the releases of τ_j occurring in M_s is maximized when the first release of τ_j is requested at $t_z + O_{j,s}$ under the following conditions,

1. *Condition B1 -* $t_z + O_{j,s} < t \leq t_{\mathcal{L}_{j,s}}$
2. *Condition B2 -* $t_z + O_{j,s} < t$ *and* $t_{\mathcal{L}_{j,s}} < t$ *and* $t_{\mathcal{L}_{j,s}} \leq t_z + O_{j,s} + C^n_{j,s}$
3. *Condition B3 -* $t > t_z + O_{j,s}$ *and* $t > t_{\mathcal{L}_{j,s}}$ *and if* $DH_{j,s} = DL_{j,s} = 1$ *and for any* k *in the range* $1 \leq k < \mathcal{L}_j$, $t_{k,s} < t_{\mathcal{L}_j,s}$.

Proof: Let $\tau_1, \tau_2, ..., \tau_i$ denote a set of priority-ordered tasks with τ_i being the task with lowest priority. Consider a particular request for τ_i that occurs at r_1 such that $0 \leq r_1 < t_z + O_{j,s}$ with its deadline beyond $t_z + O_{j,s}$. Suppose that requests for the task τ_j with priority higher than τ_i, occur at $t_z + O_{j,s}$, $t_z + O_{j,s} + T_{j,s}$, $t_z + O_{j,s} + 2T_{j,s}$,..., $t_z + O_{j,s} + kT_{j,s}$, as illustrated in Figure 5.6. Clearly, the pre-emption of τ_i by τ_j, will cause a certain amount of delay in the completion of the request for τ_i that occurred at r_1, unless the request for τ_i is completed before $t_z + O_{j,s}$. It is important to note that advancing the release pattern of τ_j is not allowed since as per the semantics of the MM-MCS model, the first release of τ_j cannot occur at any time earlier than $t_z + O_{j,s}$. For this reason, τ_j cannot interfere τ_i for $t \leq t_z + O_{j,s}$ or if the deadline of τ_i is at or within $t_z + O_{j,s}$.

Condition B1: $t_z + O_{j,s} < t \leq t_{\mathcal{L}_{j,s}}$

If $0 \leq t \leq t_{\mathcal{L}_{j,s}}$, then releases of τ_j can execute either with normal or degraded budget in the interval [$t_z + O_{j,s}$, t]. Therefore, two possibilities can be considered depending on the value of $DL_{j,s}$. From Definition 17, it is clear that if $DL_{j,s} = 0$, then releases of τ_j can execute with their normal budget in the interval [$t_z + O_{j,s}$, t]. Similarly, if $DL_{j,s} = 1$, then releases of τ_j can execute with their normal budget only in the interval [$t_z + O_{j,s}$, $t_{1,s}$] for $t_{1,s} > t_z + O_{j,s}$ and with degraded budgets in the interval ($t_{1,s}$, t]. By delaying the release pattern by ϵ, interference from τ_j gets decreased utmost by ϵ as some execution of τ_j is excluded in the interval [$t_z + O_{j,s}$, t].

The above arguments are valid when the degraded budgets of τ_j are less than or equal to its normal budget, an important assumption made in the syntax of the MM-MCS model.

Condition B2: $t_z + O_{j,s} < t$ and $t_{\mathcal{L}_{j,s}} < t$ and $t_{\mathcal{L}_{j,s}} \leq t_z + O_{j,s} + C^n_{j,s}$

Three possibilities can be considered for this condition as follows.

- It is possible that all releases of τ_j that occur in the interval [$t_z + O_{j,s}$, t] can execute with their safe budget. This can happen if $DH_{j,s} = 0$.
- All releases of τ_j that occur in the interval [$t_z + O_{j,s}$, t] can execute with their degraded budget. This can happen if $DL_{j,s} = 1$ and when system has transitioned from the normal state $S_{0,s}$ to any other safe state $S_{k,s}$ before $t_{\mathcal{L}_j,s}$, for $k < \mathcal{L}_j$.
- If $DH_{j,s} = 1$, then releases of τ_j can execute with their safe budget only in the interval [$t_z + O_{j,s}$, $t_z + O_{j,s} + t_{\mathcal{L}_{j+1,s}}$] beyond which they can interfere only with their degraded budgets.

For all the above possibilities, delaying the release pattern of τ_j by ϵ, interference from τ_j can either decrease or remains unchanged. Interference from τ_j can decrease utmost by ϵ as some execution from releases of τ_j can be excluded in the interval [$t_z + O_{j,s}$, t]. Similarly, interference will remain unchanged when the release pattern is delayed for $\epsilon \leq (T_{j,s} - C^s_{j,s})$ beyond which some execution of τ_j can be excluded in the interval [$t_z + O_{j,s}$, t]. The above arguments are valid when the degraded budgets of τ_j are less than or equal to its normal budget, an important assumption made in the syntax of the MM-MCS model.

Condition B3: $t > t_z + O_{j,s}$ and $t > t_{\mathcal{L}_{j,s}}$ and if $DH_{j,s} = DL_{j,s} = 1$ and for any k in the range $1 \leq k < \mathcal{L}_j$, $t_{k,s} < t_{\mathcal{L}_j,s}$.

If it is the case that there is a time instant $t_{k,s}$ such that $t_{k,s} < t_{\mathcal{L}_j,s}$ for any k in the range $1 \leq k < \mathcal{L}_j$, and if $DL_{j,s} = 1$, then τ_j will be degraded before $t_{\mathcal{L}_j,s}$ since $DH_{j,s} = 1$. This can happen irrespective of where the first release of τ_j occurs in M_s. As the semantics of the MM-MCS model does not allow τ_j to overrun its degraded budget, it is not possible for τ_j to execute with its safe or normal budget beyond $t_{k,s}$. If the time instant $t_{k,s}$ happens to be $t_{1,s}$ and if $t_{1,s} > t_z + O_{j,s}$, then τ_j can execute with its normal budget in the interval [$t_z + O_{j,s}$, $t_{1,s}$]. Otherwise, if $t_{1,s} \leq t_z + O_{j,s}$, then τ_j can interfere with any of its degraded budget in the interval [$t_z + O_{j,s}$, t]. In either case, by delaying the release pattern of τ_j by ϵ, interference from τ_j can either remain unchanged or decreased. Interference can decrease utmost by ϵ as some execution of τ_j gets excluded in the interval [$t_z + O_{j,s}$, t] or the interference from τ_j will remain unchanged for $\epsilon \leq (T_{j,s} - C^s_{j,s})$ beyond which definitely some execution of τ_j can be excluded in the interval [$t_z + O_{j,s}$, t]. The above arguments are valid when the degraded budgets of τ_j are less than or equal to its normal budget, an important assumption made in the syntax of the MM-MCS model.

■

The following theorem considers the release pattern of τ_j that can maximize the interference from τ_j to τ_i in the new-mode M_s when τ_j overruns its normal budget.

Theorem 4 *Let τ_j denote a new-mode task with priority higher than τ_i. Then, for a mode transition from the old-mode M_p to the new-mode M_s occurring at t_z, for any given valid state transition pattern in M_s, for a task τ_i with its release time at r_1 occurring either in M_p or M_s, for any $t > t_z + O_{j,s}$, the interference in interval [$t_z + O_{j,s}$, t] for a job of τ_i from the releases of a higher priority task τ_j occurring in M_s is maximized when the budget overrun of τ_j occurs at $t_{\mathcal{L}_{j,s}}$ under any of the following conditions,*

1. *Condition B4 -* $t_z + O_{j,s} + C^n_{j,s} < t_{\mathcal{L}_{j,s}} < t$ *and* $DH_{j,s} = 0$
2. *Condition B5 -* $t_z + O_{j,s} + C^n_{j,s} < t_{\mathcal{L}_{j,s}} < t$ *and* $DH_{j,s} = 1$ *and* $DL_{j,s} = 0$
3. *Condition B6 -* $t_z + O_{j,s} + C^n_{j,s} < t_{\mathcal{L}_{j,s}} < t$ *and if* $DH_{j,s} = DL_{j,s} = 1$ *and for all* k *in the range* $1 \leq k < \mathcal{L}_j$, $t_{k,s} = t_{\mathcal{L}_j,s}$

Proof: Let $\tau_1, \tau_2, ..., \tau_i$ denote a set of priority-ordered tasks with τ_i being the task with lowest priority. Consider a particular request for τ_i that occurs at r_1 with its deadline beyond $t_z + O_{j,s}$. Suppose that requests for the task τ_j with priority higher than τ_i, occur at $t_z + O_{j,s}$, $t_z + O_{j,s} + T_{j,s}$, $t_z + O_{j,s} + 2T_{j,s}$,..., $t_z + O_{j,s} + kT_{j,s}$, as illustrated in Figure 5.6. Let $t_{\mathcal{L}_j,s}$ denote the time instant at which τ_j overrun its normal budget ($C^n_{j,s}$) in M_s leading to a system transition to the safe state $S_{\mathcal{L}_j,s}$.

Clearly, the pre-emption of τ_i by τ_j, will cause a certain amount of delay in the completion of the request for τ_i, unless the request for τ_i is completed before $t_z + O_{j,s}$. It is important to note that advancing the release pattern of τ_j is not allowed since as per the semantics of the MM-MCS model, the first release of τ_j cannot occur at any time earlier than $t_z + O_{j,s}$. For this reason, τ_j cannot interfere τ_i for $t \leq t_z + O_{j,s}$ or if the deadline of τ_i is at or within $t_z + O_{j,s}$.

Condition B4: $t_z + O_{j,s} + C^n_{j,s} < t_{\mathcal{L}_{j,s}} < t$ and $DH_{j,s} = 0$

From Definition 16, it is clear that if $DH_{j,s} = 0$, all releases of τ_j that are active at or after $t_{\mathcal{L}_j,s}$ can execute with their safe budget. By advancing (shifting left) the release pattern of τ_j by ϵ, interference from τ_j in the interval [$t_z + O_{j,s}$, t] is decreased as the number of jobs interfering with safe budgets is now reduced since the release of τ_j that was previously assumed to overrun its normal budget at $t_{\mathcal{L}_j,s}$ will finish its execution before $t_{\mathcal{L}_j,s}$ with normal budget. By delaying (shifting right) the release pattern by ϵ, interference either remains unchanged or decreased. Interference from τ_j will remain unchanged for $\epsilon < T_j$ beyond which number of releases of τ_j interfering with safe budget decreases leading to decrease in the interference in the interval [$t_z + O_{j,s}$, t].

Condition B5: $t_z + O_{j,s} + C^n_{j,s} < t_{\mathcal{L}_{j,s}} < t$ and $DH_{j,s} = 1$ and $DL_{j,s} = 0$

From Definition 16, we note that if $DH_{j,s} = 1$, all releases of τ_j can interfere with their safe budget only in the interval $[t_{\mathcal{L}_j,s}, t_{\mathcal{L}_{j+1},s}]$ and with their degraded budget in the interval $[t_{\mathcal{L}_{j+1},s}, t]$. Therefore, by delaying the release pattern by ϵ, some instances of τ_j that were previously interfering with safe budgets in the interval $[t_{\mathcal{L}_j,s}, t_{\mathcal{L}_{j+1},s}]$ can be pushed beyond $t_{\mathcal{L}_{j+1},s}$. These releases of τ_j can now interfere only with degraded budget leading to decrease in interference in the interval $[t_z + O_{j,s}, t]$. However, the number of releases of τ_j interfering with safe budgets will remain unchanged in the interval $[t_z + O_{j,s}, t]$ for $0 < \epsilon < T_j$ since the release of τ_j that was considered to overrun its normal budget at $t_{\mathcal{L}_j,s}$ can now overrun its normal budget at any time greater than $t_{\mathcal{L}_j,s}$.

By advancing the release pattern by ϵ, the release of τ_j that was previously considered to overrun its normal budget at $t_{\mathcal{L}_j,s}$, can now finish its execution with its normal budget before $t_{\mathcal{L}_j,s}$. Therefore, the number of releases of τ_j interfering with safe budget decreases leading to decrease in interference in the interval $[t_z + O_{j,s}, t]$.

Condition B6: $t_z + O_{j,s} + C^n_{j,s} < t_{\mathcal{L}_{j,s}} < t$ and if $DH_{j,s} = DL_{j,s} = 1$ and for all k in the range $1 \leq k < \mathcal{L}_j, t_{k,s} = t_{\mathcal{L}_j,s}$

If it is the case that $DH_{j,s} = DL_{j,s} = 1$ and for all k in the range $1 \leq k < \mathcal{L}_j$, $t_{k,s} = t_{\mathcal{L}_j,s}$, then system transition happens directly from the normal state $S_{0,s}$ to the safe state $S_{\mathcal{L}_j,s}$. Under such a state transition, no release of τ_j gets degraded before $t_{\mathcal{L}_j,s}$ even if $DL_{j,s} = 1$.

By delaying the release pattern by ϵ, some instances of τ_j that were previously interfering with safe budgets in the interval $[t_{\mathcal{L}_j,s}, t_{\mathcal{L}_{j+1},s}]$ can be pushed beyond $t_{\mathcal{L}_{j+1},s}$. These releases of τ_j can now interfere only with degraded budget leading to decrease in interference in the interval $[t_z + O_{j,s}, t]$. However, the number of releases of τ_j interfering with safe budgets will remain unchanged in this interval for $0 < \epsilon < T_{j,s}$ as the release of τ_j that was considered to overrun its normal budget at $t_{\mathcal{L}_j,s}$ can now overrun at any time greater than $t_{\mathcal{L}_j,s}$.

By advancing the release pattern by ϵ, release of τ_j that was previously considered to overrun its normal budget at $t_{\mathcal{L}_j,s}$, can now finish its execution with its normal budget before $t_{\mathcal{L}_j,s}$. Therefore, number of releases of τ_j interfering with safe budget decreases leading to decrease in interference in the interval $[t_z + O_{j,s}, t]$. ■

The following corollary considers the release pattern of τ_j that can maximize the interference from τ_j to τ_i in the new-mode M_s when the release time of τ_i is at or after $t_z + O_{j,s}$.

Corollary 5 *Let τ_j denote a new-mode task with priority higher τ_i. Then, for a mode transition from the old-mode M_p to the new-mode M_s occurring at t_z, for any given valid state transition pattern in M_s, for a task τ_i with its release time at r_1 such that $r_1 \geq t_z + O_{j,s}$, for any $t > t_z + O_{j,s}$, the interference in interval $[t_z + O_{j,s}, t]$ for a job of τ_i from the releases of a higher priority task τ_j occurring in M_s is maximized when τ_i is requested simultaneously with the request for τ_j under the following conditions,*

1. *Condition C1 - $t \leq t_{\mathcal{L}_{j,s}}$*
2. *Condition C2 - for any $t > t_{\mathcal{L}_{j,s}}$ and if $DH_{j,s} = DL_{j,s} = 1$ and **for any** k in the range $1 \leq k < \mathcal{L}_j$, $t_{k,s} < t_{\mathcal{L}_j,s}$.*

Proof: The proof is similar to the proof for Theorem 1 presented in section 4.4. As the release time of τ_i in the new-mode occurs at or after $t_z + O_{j,s}$, t_z has no impact on the release pattern of the interfering task τ_j that can maximize its interference to τ_i. ■

Next, the expressions to find upper bound number of releases of τ_j interfering with safe budget (denoted as $N_s^{new}(j,t,P_{s,z}^l)$), normal budget (denoted as $N_n^{new}(j,t,P_{s,z}^l)$) and degraded budgets (denoted as $N_d^{new}(j,t,P_{s,z}^l)$) for a specific valid transition pattern $P_{s,z}^l \in \mathcal{P}_{s,z}$, in the new-mode M_s based on the conditions considered in the Theorem 3, Theorem 4 and corollary 5 are presented. For this purpose, let $t_n = t_z + O_{j,s}$.

Expression for $N_s^{new}(j,t,P_{s,z}^l)$:

$$\mathrm{N}_s^{new}(j,t,P_{s,z}^l) = \begin{cases} 0, & B1 \vee B3 \vee C1 \vee C2 \\ \min\left\{1 + \left\lceil \frac{t - max(t_n, t_{\mathcal{L}_{j,s}})}{T_{j,s}} \right\rceil_0, \left\lceil \frac{t - t_n}{T_{j,s}} \right\rceil_0 \right\}, & (B2 \wedge DH_{j,s} = 0) \vee B4 \\ \min\left\{1 + \left\lceil \frac{min(t, t_{\mathcal{L}_{j+1,s}}) - max(t_n, t_{\mathcal{L}_{j,s}})}{T_{j,p}} \right\rceil_0, \left\lceil \frac{t - t_n}{T_{j,s}} \right\rceil_0 \right\}, & (B2 \wedge DH_{j,s} = 1) \vee B5 \vee B6 \end{cases} \tag{5.4}$$

For conditions $B1$, $B3$, $C1$ and $C2$, no job of τ_j can execute with its safe budget in the interval $[t_z + O_{j,s}, t]$. Hence, the value of $N_s^{new}(j,t,P_{s,z}^l)$ is 0. If the value of $DH_{j,s}$ is 0 then for conditions $B2$ and $B4$, the maximum number of releases of τ_j that can interfere with safe budget and can fit into an interval of length $t - max(t_n, t_{\mathcal{L}_{j,s}})$ is

$$1 + \left\lceil \frac{t - max(t_n, t_{\mathcal{L}_{j,s}})}{T_{j,s}} \right\rceil_0$$

For smaller values of $t_{\mathcal{L}_{j,s}}$,the above expression can be pessimistic when considering the number of jobs

that can fit in an interval of length $t - t_n$. Therefore, the value of $N_s^{new}(j, t, P_{s,z}^l)$ is given as

$$\min\left\{1 + \left\lceil \frac{t - max(t_n, t_{\mathcal{L}_{j,s}})}{T_{j,s}} \right\rceil_0, \left\lceil \frac{t - t_n}{T_{j,s}} \right\rceil_0 \right\}$$

If the value of $DH_{j,s}$ is 1 then for conditions $B2$, $B5$ and $B6$, the expression for $N_s^{new}(j, t, P_{s,z}^l)$ considers the upper bounds number of jobs of τ_j that can interfere with safe budget in the interval of length $min(t, t_{\mathcal{L}_{j+1,s}}) - max(t_n, t_{\mathcal{L}_{j,s}})$. For smaller values of $t_{\mathcal{L}_{j+1,s}}$, only the interval of length $t - t_n$ is considered to reduce the pessimism.

Expression for $N_n^{new}(j, t, P_{s,z}^l)$:

$$\mathrm{N}_n^{new}(j, t, P_{s,z}^l) = \begin{cases} \left\lceil \frac{t - t_n}{T_{j,s}} \right\rceil_0, & DL_{j,s} = 0 \wedge (B1 \vee C1) \\ \left\lceil \frac{min(t, t_{1,s}) - t_n}{T_{j,s}} \right\rceil_0, & (DL_{j,s} = 1 \wedge (B1 \vee C1)) \vee B3 \vee C2 \\ \left\lceil \frac{t - t_n}{T_{j,s}} \right\rceil_0 - N_s^{new}(j, t, P_{s,z}^l), & DL_{j,s} = 0 \wedge B4 \\ \left\lceil \frac{min(t, t_{\mathcal{L}_{j+1,s}}) - t_n}{T_{j,s}} \right\rceil_0 - N_s^{new}(j, t, P_{s,z}^l), & B5 \vee B6 \end{cases} \tag{5.5}$$

As discussed in [19], the upper bound number of releases of τ_j that can interfere τ_i in an interval of length $t - t_n$ is given as $\left\lceil \frac{t-t_n}{T_{j,s}} \right\rceil_0$ where $\lceil x \rceil_0$ returns $\lceil x \rceil$ if $x > 0$, otherwise returns 0. If $DL_{j,s} = 0$ then for conditions $B1$ and $C1$, all jobs of τ_j released in the interval [t_n, t] interfere with normal budget and the value of $N_n^{new}(j, t, P_{s,z}^l)$ is given as $\left\lceil \frac{t-t_n}{T_{j,s}} \right\rceil_0$. Otherwise, if $DL_{j,s} = 1$ then jobs of τ_j can interfere with normal budget only in the interval [t_n, $t_{1,s}$] and the value of $N_n^{new}(j, t, P_{s,z}^l)$ for conditions $B1$, $C1$, $B3$ and $C2$ is given as $\left\lceil \frac{min(t,t_{1,s})-t_n}{T_{j,s}} \right\rceil_0$ where $t_{1,s}$ denotes the time instant at which system transition to the safe state $S_{1,s}$ happens. For the condition $B4$, if $DL_{j,s} = 1$ then jobs of τ_j can interfere with normal budget only in the interval [t_n, t]. Hence, the value of $N_n^{new}(j, t, P_{s,z}^l)$ is found by subtracting $N_s^{new}(j, t, P_{s,z}^l)$ from the total number of jobs that can be released in the interval of length $t - t_n$. Conditions $B4$ and $B6$ consider the value of $DH_{j,s}$ to be 1. Therefore, the value of $N_n^{new}(j, t, P_{s,z}^l)$ is found by subtracting $N_s^{new}(j, t, P_{s,z}^l)$ from the total number of jobs that can be released in the interval of length $min(t, t_{\mathcal{L}_{j+1,s}}) - t_n$.

Expression for $N_d^{new}(j, t, P_{s,z}^l)$:

Once $N_s^{new}(j, t, P_{s,z}^l)$ and $N_n^{new}(j, t, P_{s,z}^l)$ are found, remaining releases of τ_j can be considered to

interfere with the degraded budgets and is given by the following expression,

$$N_d^{new}(j,t,P_{s,z}^l) = \left\lceil \frac{t-t_n}{T_{j,s}} \right\rceil_0 - (N_s^{new}(j,t,P_{s,z}^l) + N_n^{new}(j,t,P_{s,z}^l)) \tag{5.6}$$

5.4.4 Response Time for the Old-Mode Task

In the following corollary, the equation for computing the worst-case response time for an old-mode task in the old-mode M_p is considered.

Corollary 6 *For a given mode transition from the old-mode M_p to the new-mode M_s occurring at t_z, for an old-mode task τ_i, its worst-case response time for a specific valid transition pattern $P_{p,z}^l \in \mathcal{P}_{p,z}$, in the old-mode M_p, is given by the solution to the following iterative equation,*

$$\begin{aligned} R_{i,p}^{l,z} = C_{i,p}^x(R_{i,p}^l) + \sum_{j \in hp(i,p)} \Big\{ & N_s^{old}(j,R_{i,p}^l,P_{p,z}^l) * C_{j,p}^s + N_n^{old}(j,R_{i,p}^l,P_{p,z}^l) * C_{j,p}^n \\ & + N_d^{old}(j,R_{i,p}^l,P_{p,z}^l) * C_{j,p}^{\mathcal{D}} \Big\} \end{aligned} \tag{5.7}$$

where $hp(i,p)$ denotes the set of tasks with priority higher than τ_i belonging to the old-mode M_p. For any $t > 0$, the value of $C_{i,p}^x(t)$ is given based on the following expression.

$$C_{i,p}^x(t) = \begin{cases} C_{i,p}^n, & t < t_{\mathcal{L}_i,p} \\ C_{i,p}^s, & t \geq t_{\mathcal{L}_i,p} \end{cases}$$

For an interfering task τ_j, $C_{j,p}^{\mathcal{D}}$ denotes its maximum degraded budget that can be allocated to it in M_p and is given by the following expression,

$$C_{j,p}^{\mathcal{D}} = max(DH_{j,p} * C_{j,p}^{\mathcal{H}}, DL_{j,p} * C_{j,p}^{\mathcal{L}})$$

*where $C_{j,p}^{\mathcal{H}}$ denotes the maximum degraded budget that can be assigned to τ_j when one or more tasks belonging to M_p with criticality value **higher** than $\mathcal{L}_j$ overrun their normal budget. $C_{j,p}^{\mathcal{L}}$ denotes the maximum degraded budget that can be assigned to τ_j when one or more tasks belonging to the mode M_p with criticality value **lower** than $\mathcal{L}_j$ overrun their normal budget.*

Proof: Since the releases of a higher priority task τ_j can interfere either with normal or safe or degraded budgets, the worst-case response time for τ_i is the sum of its safe budget and the upper bound

interference from releases of higher priority tasks interfering with normal, safe and degraded budgets. ■

In the following corollary, the equation for computing the worst-case response time for an old-mode task τ_i of type changed, unchanged, or old-mode completed for a specific valid transition pattern in the new-mode M_s is considered.

Corollary 7 *For a given mode transition from the old-mode M_p to the new-mode M_s occurring at t_z, for an old-mode task τ_i, its worst-case response time for a specific valid transition pattern $P^k_{s,z} \in \mathcal{P}_{s,z}$ in the new-mode M_s, is given by the solution to the following iterative equation,*

$$R^{k,z}_{i,s} = R^z_{i,p} + \sum_{j \in hp(i,s)} \Bigg\{ N^{new}_s(j, R^{k,z}_{i,s}, P^k_{s,z}) * C^s_{j,s} + N^{new}_n(j, R^{k,z}_{i,s}, P^k_{s,z}) * C^n_{j,s} + N^{new}_d(j, R^{k,z}_{i,s}, P^k_{s,z}) * C^{\mathcal{D}}_{j,s} \Bigg\} \tag{5.8}$$

where $hp(i,s)$ denotes a set of tasks with priority higher than τ_i belonging to the mode M_s and $R^z_{i,p}$ is given by the following expression,

$$R^z_{i,p} = \max_{\forall P^l_{p,z} \in \mathcal{P}_{p,z}} R^l_{i,p} \tag{5.9}$$

For an interfering task τ_j, $C^{\mathcal{D}}_{j,s}$ denotes its maximum degraded budget that can be allocated to it in M_s and is given by the following expression,

$$C^{\mathcal{D}}_{j,s} = max(DH_{j,s} * C^{\mathcal{H}}_{j,s}, DL_{j,s} * C^{\mathcal{L}}_{j,s})$$

Proof: For a given t_z, the maximum value of the response time of τ_i due to the interference from a higher priority task τ_j in M_p for all possible combinations of valid transition patterns in the old-mode M_p is given by $R^z_{i,p}$. Note that only if $R^z_{i,p} > t_z + O_{j,s}$ it is possible for releases of τ_j to interfere τ_i in the new-mode M_s since as per the semantics of the MM-MCS model, the first instance of τ_j is released in the new-mode M_s not earlier than $t_z + O_{j,s}$. Therefore, when $R_{i,p} > t_z + O_{j,s}$, the total interference from releases of τ_j is the sum of interference from releases of τ_j interfering with normal and/or safe and/or degraded budgets can be found. ■

In the following corollary, the equation for computing the worst-case response time for an old-mode task τ_i of type changed, unchanged, or old-mode completed for all possible mode transition and valid state transition pattern that can occur in the interval [0, $T_{i,p}$] is considered.

Corollary 8 *The worst-case response time for an old-mode task τ_i, released in the old-mode M_p is given by the following equation,*

$$R_i^o = \max_{\forall s \in \{1,2,..,r\} \setminus p} \left\{ \max_{\forall t_z} \left\{ \max_{\forall P_{s,z}^k \in \mathcal{P}_{s,z}} R_{i,s}^{k,z} \right\} \right\} \tag{5.10}$$

where $\mathcal{P}_{s,z}$ denotes the set of valid state transition patterns that can occur to τ_i in the new-mode M_s for a specific t_z.

Proof: For an old-mode task task τ_i, its worst-case response time is the sum total of the maximum possible interference from higher priority tasks that can interfere τ_i in the old-mode M_p among all possible valid state transition patterns in M_p and the maximum possible interference from higher priority tasks in M_s among all possible valid state transition patterns in M_s for all possible M_s are considered. ■

5.4.5 Response Time for the New-Mode Task

Let us consider a task $\tau_i \in \Gamma_s$ that is released in the new-mode M_s. In the analysis of the maximum interference that can occur to τ_i for a given mode transition from M_p to M_s, at worst-case, there can be one instance of old-mode completed, unchanged and changed tasks each released in the old-mode M_p and interfering τ_i in M_s. Depending on the system's state just before t_z and criticality of the interfering tasks, old-mode tasks can interfere with their normal or degraded or safe budget. Among all possible safe states that the system can exist just before t_z, we need choose a state such that the total interference experienced by τ_i from higher priority tasks that are released in M_p and active in M_s is maximum. For this, we can use the function $maxB(\tau_j, l, M_p)$ (stated in Definition 18) which returns the maximum possible budget that can be assigned to τ_j in a specific state $S_{l,p}$ in the mode M_p. Therefore, for a task τ_i belonging to the new-mode M_s, the upper bound interference from higher priority tasks from the old-mode is given by the following expression,

$$I_{i,s}^o = \max_{\forall p \in \{1,2,..,r\} \setminus s} \left\{ \max_{0 \le l \le L} \Big(\sum_{\forall \tau_j \in hp(i,p) \setminus hp(i,p)_A} maxB(\tau_j, l, M_p) \Big) \right\} \tag{5.11}$$

where $hp(i,p)_A$ denotes the set of tasks of type aborted with priority higher than τ_i in the old-mode M_p.

In the following corollary, the equation for computing the worst-case response time for a new-mode task τ_i for all possible valid state transition pattern that can occur in the interval [0, $T_{i,s}$] for $t_z = 0$ is considered.

Corollary 9 *For a new-mode task τ_i, its worst-case response time in the new-mode M_s is given by the following equation,*

$$R_i^n = \max\left\{R_i^s, (\max_{\forall P_{s,0}^l \in \mathcal{P}_{s,0}} w_{i,s}^l - O_{i,s})\right\} \tag{5.12}$$

where R_i^s denotes the worst-case response time of τ_i in M_s without considering mode transitions. The value of R_i^s can be computed based on equation 4.5 presented in section 4.4. $O_{i,s}$ denotes the offset for τ_i and $w_{i,s}^l$ denotes the temporal window to enclose the response time for the new-mode task τ_i in the new-mode M_s for a specific valid transition pattern $P_{s,0}^l \in \mathcal{P}_{s,0}$ and is given by the solution to the following iterative equation,

$$\begin{aligned} w_{i,s}^l = C_{i,s}^x(w_{i,s}^l) + I_{i,s}^o + \sum_{j \in hp(i,s)} \Big\{ & N_s^{new}(j, w_{i,s}^l, P_{s,0}^l) * C_{j,p}^s \\ & + N_n^{new}(j, w_{i,s}^l, P_{s,0}^l) * C_{j,s}^n + N_d^{new}(j, w_{i,s}^l, P_{s,0}^l) * C_{j,s}^{\mathcal{D}} \Big\} \end{aligned} \tag{5.13}$$

$\mathcal{P}_{s,0}$ denotes the set of valid state transition patterns that can occur to τ_i in M_s for $t_z = 0$. For any $t > 0$, the value of $C_{i,s}^x(t)$ is given by the following expression,

$$C_{i,s}^x(t) = \begin{cases} C_{i,s}^n, & t < t_{\mathcal{L}_i,s} \\ C_{i,s}^s, & t \geq t_{\mathcal{L}_i,s} \end{cases}$$

Proof: The response time for a task τ_i with both release time and deadline in the new-mode mode M_s is maximized when it is executing with its safe budget and simultaneously experiencing the interference from tasks released in old-mode and new-mode. By considering the upper bound interference from tasks released in the old-mode for all possible old-modes for M_s, it is not necessary to consider all possible values of t_z. Therefore, analysis is done by considering $t_z = 0$. Since τ_j can interfere with safe or normal or degraded budgets, the maximum possible interference from releases of τ_j executing with safe or normal or degraded budgets among all possible valid state transition patterns that can occur in the new-mode M_s is also considered to compute R_i^n.

As tasks may be introduced after their offset, the response time of τ_i is given by the subtraction of this offset $O_{i,s}$. ■

A task is considered to be schedulable if the value of its worst-case response time is less than or equal to its deadline. The value of the worst-case response time that is computed for the task can inform not only whether the task has exceeded its deadline but also by what amount the deadline has been missed. Hence, the test is very useful to decide the offset values for new-mode tasks to achieve the schedulability

Symbols	Meaning
$hp(i,p)$	a set of tasks with priority higher than τ_i belonging to the old-mode M_p.
$R_{i,p}^{l,z}$	the worst-case response time of τ_i for a specific valid transition pattern. $P_{p,z}^{l} \in \mathcal{P}_{p,z}$ in M_p .
$R_{i,p}^{z}$	the maximum value among $R_{i,p}^{l,z}$.
$R_{i,s}^{k,z}$	the worst-case response time for an old-mode task τ_i for a specific state transition pattern $P_{p,s}^{k} \in \mathcal{P}_{s,z}$.
R_i^o	the worst-case response time for an old-mode task τ_i, released in the old-mode M_p.
$I_{i,s}^{o}$	the maximum possible interference for τ_i that can be propagated from the old-mode tasks to the new-mode is computed .
R_i^s	the worst-case response time of τ_i in M_s without considering mode transitions.
$w_{i,s}^{l}$	the temporal window to enclose the response time for the new-mode task τ_i in the new-mode M_s for a specific valid transition pattern $P_{s,0}^{l} \in \mathcal{P}_{s,0}$.
R_i^n	the worst-case response time in the new-mode M_s.
$\mathcal{P}_{s,0}$	a set of valid state transition patterns that can occur to τ_i in M_s for $t_z = 0$.
N_n^{new}	the upperbound number of releases of any task interfering with normal budget in the new-mode.
N_d^{new}	the upperbound number of releases of any task interfering with degraded budget in the new-mode.
N_s^{new}	the upperbound number of releases of any task interfering with safe budget in the new-mode.
$C_{j,s}^{\mathcal{D}}$	the maximum degraded budget that can be allocated to it in M_s in the new-mode.
N_n^{old}	the upperbound number of releases of any task interfering with normal budget in the old-mode.
N_d^{old}	the upperbound number of releases of any task interfering with degraded budget in the old-mode.
N_s^{old}	the upperbound number of releases of any task interfering with safe budget in the old-mode.

Table 5.4: Symbols and their meanings used in the schedulability analysis of the MM-MCS model.

of the mode transition.

As the proposed model provides a higher level of flexibility in the choice of degradation of tasks, there is a need to capture values of many variables to decide on the tasks to be degraded at run-time. Hence, the analysis of the MM-MCS model requires many parameters to represent these variables. In Table 5.4.5, list of symbols and their meanings used in the schedulability analysis of the MM-MCS model are tabulated for reader's reference.

5.4.6 Run-Time Complexity for Computing the Worst-Case Response Time

Since the semantics of the MM-MCS is designed in such a way that interference from old-mode tasks does not impact the states of the new-mode, it is possible to analyse the worst-case response time for the old-mode and new-mode independently. Therefore, the run-time complexity involved in computing the worst-case response time is the same for both old-mode and the new-mode tasks.

The complexity for finding R_i^o in a given old-mode M_p is $O(r * N * T_{i,p}^{L+1})$ where r denotes the total number of modes in which τ_i is active. $T_{i,p}$ denotes the time period of τ_i in the old-mode M_p. N

denotes the maximum number of tasks for all possible combinations of M_p and M_s whose value can be given as $N = \max_{\forall s \in \{1,2,..,r\} \setminus p} max(|\Gamma_p|, |\Gamma_s|)$. L denotes the total number of criticality levels supported by the system. $(T_{i,p})^L$ denotes the maximum possible number of transition instances that needs to be considered for the analysis. The complexity for $maxB(\tau_j, l, M_p)$ is $O(N)$. The complexity for finding R_i^n in a given new-mode M_s is $O(N * T_{i,s}^L)$ while that for finding $I_{i,s}^o$ is $O(r * L * N)$. Since the values of r and L can be regarded as a constant, the resulting complexity is pseudo-polynomial with respect to the number of tasks.

5.4.7 Priority Assignment

An inspection of the worst-case response time equation presented for the old-mode task in equations 5.7, 5.8 and for the new-mode task in equation 5.13 are observed to adhere to the three conditions stated in [77]. These conditions are provided in section 4.4.4. Therefore, the Audsley's algorithm can be employed for the priority assignment. It is clear from these equations that to compute worst-case response time of τ_i in old-mode or in new-mode, only tasks with priority higher than τ_i are considered. The values of the parameters of the higher priority tasks such as budgets, offset, criticality and time period of are considered to compute the worst-case response time of τ_i. These parameters are independent and are not affected by the priority ordering of tasks. In these equations, it can be observed that the budget with which any higher priority task can interfere in a given time interval depends only on the time instances considered in the state transition pattern and the mode. The maximum value of the degraded budget from any higher priority task in a given time interval is dependent only on the criticality level of other tasks that can overrun their normal budget in that interval and independent of their priority ordering thereby satisfying conditions 1 and 2. Since equations 5.7, 5.8 and 5.13 consider only the interference from higher priority tasks, swapping tasks of adjacent priority cannot make the task being assigned the higher priority unschedulable as the number of higher priority tasks that can now interfere at the new priority level decreases. Therefore, the worst-case response time for the task being assigned the higher priority is also reduced. Similarly, the worst-case response time of the task that is assigned with a lower priority value increases due to increase in the number of higher priority tasks and therefore it cannot become schedulable, if it was previously unschedulable at the higher priority.

5.4.8 Feasibility Analysis for the MM-MCS model

Let S denote a MM-MCS with r modes of operation M_1 to M_r. Let Γ_k denote a set of tasks belonging to the mode M_k. S is said to be feasible if the following condition is satisfied,

$$\max_{\forall p \in \{1,2,..,r\}} \left\{ \max_{0 \leq l \leq L} \left(\sum_{\forall \tau_j \in \Gamma_p} maxB(\tau_j, l, M_p)/T_{j,p} \right) \right\} \leq 1 \tag{5.14}$$

The function $maxB(\tau_j, l, M_p)/T_{j,p}$ returns the maximum budget with which τ_j can execute in the safe state $S_{l,p}$ in mode M_p. Therefore, equation 5.14 checks whether the sum of the utilisation of all tasks of the system when it executes only in the safe state $S_{l,p}$ without undergoing any state transitions is less than or equal to 1. When the taskset Γ satisfies equation 4.6, it means that Γ is feasible. It is to be noted that the feasibility condition presented above is only a necessary condition.

The feasibility conditions presented above do not consider degradation information of tasks, the complexity involved to check this condition can be given as $O(r * L * N^2)$ where L denotes the number of criticality levels supported by the system and the value of N can be given as $N = \max_{\forall p \in \{1,2,..,r\}} |\Gamma_p|$.

5.5 Offsets

5.5.1 An Algorithm for Offset Calculation

Algorithm 1 shows the steps to find the offsets for each new-mode task. Vector O_r and O_{max}, has as many components as tasks in the new-mode M_s. Initially, the values of all components in O_r is set to 0. As the MM-MCS model captures the offset parameters, vector O_{max} holds the maximum possible offset that can be assigned for each task in M_s when mode transition from M_p to M_s occurs. The maximum possible interference that can be propagated from the old-mode to the new-mode is when one instance of old-mode tasks that are active in the new-mode (unchanged, changed and old-mode completed tasks) execute with their safe budgets in the new-mode. Hence, the variable O denotes the maximum value of the sum of safe budgets of the old-mode tasks of type changed, unchanged and old-mode completed, among all possible old-modes that can considered for the new-mode M_s. As presented in the syntax of the MM-MCS model, any task τ_i can exist in r possible modes. Hence, the value of the parameter p in M_p can take any value belonging to the set $\{1, 2, ..., r\} \setminus s$.

Initially, all new-mode tasks are assigned with null offsets. If transition fails for the schedulabilty test (presented in section 5.4), then offsets for tasks starting with lowest criticality (*i.e.* those with criticality equal to 1) are updated with a value that is equal to $min(O_{i,s}, O)$ and checked for schedulability. If the transition is still not schedulable, offsets for tasks with next higher criticality (*i.e.* criticality equal to 2) are updated to $min(O_{i,s}, O)$ and again checked for schedulability in the next iteration. The variable $\mathcal{L}^*$ is used to keep track of the latest criticality level up to which tasks with criticality $\mathcal{L}^*$ are assigned with

offsets. Lines 8 to 14 of Algorithm 1 are executed for each value of $\mathcal{L}^*$. This may lead to maximum of $L+1$ iterations as the value of $\mathcal{L}^*$ can be in the range $0 \leq \mathcal{L}^* \leq L$. In each iteration, offsets for tasks are updated depending on their criticality level till the transition is feasible or till $\mathcal{L}^* \leq L$. Here, L denotes the total number of criticality levels supported by the system. It is possible that the transition may not be schedulable even after updating the offset of all new-mode tasks with their given maximum offsets. At this point, the algorithm stops and a new maximum offset value for each new-mode task should be chosen. In the following subsection, an example is provided to illustrate the use of the algorithm.

Algorithm 1 Algorithm to calculate the offsets.

1: **function** UPDATEOFFSET(Taskset Γ_s)
2: O_r := (0, 0, 0,...,0)
3: O_{max} := (0, $O_{1,s}$, $O_{2,s}$,..., $O_{|\Gamma_s|,s}$)
4: $O = \max_{\forall p \in \{1,2,..,r\} \setminus s} \left\{ \sum_{\forall \tau_i \in \Gamma_p \setminus hpA(i,p)} C^s_{i,p} \right\}$
5: $\mathcal{L}^* = 0$
6: **while** (O_r != O_{max}) **do**
7: **if** Γ is not schedulable **then**
8: $\mathcal{L}^* = \mathcal{L}^* + 1$
9: **for** τ_i in Γ **do**
10: **if** $(\mathcal{L}_i \geq 1) \wedge (\mathcal{L}_i \leq \mathcal{L}^*)$ **then**
11: $O_r[i] \leftarrow min(O, O_{i,s})$
12: **end if**
13: **end for**
14: **else**
15: Γ is schedulable
16: **end if**
17: **end while**
18: **end function**

5.5.2 An Example for Offset Calculation

Let us consider a taskset consisting of 5 tasks, τ_1 to τ_5, some of which belonging to both old-mode M_p and the new-mode M_s. The details of the tasks are described in the Table 5.2 in section 5.3.3. Each task belong to one of 2 criticality levels and one of 5 task types. $T_{i,p}$, $\mathcal{L}_i$ and $B^k_{i,p}$ denote time period, criticality, and behaviours of τ_i belonging to M_p. $C^n_{i,p}$ and $C^s_{i,p}$ denote normal budget and safe budgets of τ_i in M_p. In the example considered, a task is degraded only when any other higher criticality task overrun its normal budget. The variation in the worst-case response times and the schedulability of the taskset during mode transition for each value of $\mathcal{L}^*$ for old-mode and new-mode tasks are tabulated as separate iterations in the Table 5.5. Iteration 1 shows that with null offsets, the transition is not

FIRST ITERATION: NULL OFFSETS ($\mathcal{L}^* = 0$)								
Old-Mode Tasks				**New-Mode Tasks**				
τ_i	$T_{i,p}$	R_i^o	Test	τ_i	$T_{i,s}$	R_i^n	$O_{i,s}$	Test
τ_1	5	12.0	X	τ_1	10	32.0	0	X
τ_2	10	12.0	X	τ_2	10	33.0	0	X
τ_3	25	50.0	X	τ_5	70	71.0	0	X
τ_4	49	44.0	✓					
SECOND ITERATION ($\mathcal{L}^*$ = 1)								
τ_i	$T_{i,p}$	R_i^o	Test	τ_i	$T_{i,s}$	R_i^n	$O_{i,s}$	Test
τ_1	5	12.0	X	τ_1	10	31.5	10.0	X
τ_2	10	12.0	X	τ_2	10	33.0	0	X
τ_3	25	48.5	X	τ_5	70	71.0	0	X
τ_4	49	44.0	✓					
THIRD ITERATION ($\mathcal{L}^* = 2$)								
τ_i	$T_{i,p}$	R_i^o	Test	τ_i	$T_{i,s}$	R_i^n	$O_{i,s}$	Test
τ_1	5	3.6	✓	τ_1	10	7.9	10.0	✓
τ_2	10	2.8	✓	τ_2	10	2.8	10.0	✓
τ_3	25	17.2	✓	τ_5	70	34.0	12.0	✓
τ_4	49	44.0	✓					

Table 5.5: Variations in response time values for tasks for each iteration. Each iteration differs with the value of $\mathcal{L}^*$ in Algorithm 1.

schedulable. The algorithm then executes the second iteration of the test by increasing $\mathcal{L}^*$ to 1 and by updating the offsets for new-mode tasks whose criticality equal to 1 to $min(O, O_{i,s})$. The value of $O_{i,s}$ was assumed to be equal to $T_{i,s}$ time units for all new-mode tasks in all iterations whereas the value of O was assumed to be the sum of safe budgets of old-mode tasks that are active in the new-mode. The offset assignment in the second try leads to reduction in the response times for some old- and new-mode tasks. As the mode transition is still not schedulable, the algorithm executes the third iteration of the test by increasing $\mathcal{L}^*$ to 2 and by updating the offsets for new-mode tasks whose criticality is less than or equal to 2 to $min(O, O_{i,s})$. It was observed that at the end of third iteration, taskset is schedulable during the mode transition when all new-mode tasks with both criticality levels are updated with offsets.

Run-time complexity - To compute the offset for any task τ_i, in the worst-case, the algorithm has to run L iterations to find offset values so that the taskset Γ is schedulable. As the run-time complexity of the schedulability test is $O(r * N * T^{L+1})$, we can conclude that the run-time complexity for the offset algorithm and schedulability test is $O(r * L * N * T^{L+1})$ where r denotes the total number of modes in which τ_i can be active.

5.6 Evaluation

In the following, we present results of the experiments that were primarily designed to show the benefit of the parameters of the MM-MCS model such as degraded budgets, behaviour and criticality based offset determination of new-mode tasks to improve the mode transition schedulability. In the worst-case response time analysis presented for old-mode and new-mode task in section 5.4, any task τ_j belonging to a mode M_p can have 4 different combinations of the values of $DH_{j,p}$ and $DL_{j,p}$. Each combination can be considered as a degradation scheme to handle budget overrun of a task. To evaluate the differences between these schemes, we generate 4 categories of tasksets in each mode as follows.

- In category 1, behaviour values of any task τ_j are such that τ_j is always degraded whenever a task with criticality higher than $\mathcal{L}_j$ overruns its normal budget and remains unaffected when any other tasks overrun their normal budget. This is considered as **scheme 1 (degradation for HC overrun)** where $DH_{j,p} = DH_{j,s} = 1$ and $DL_{j,p} = DL_{j,s} = 0$. This is similar to the degradation scheme adopted in the majority of MCS studies that consider dual criticality system where all lower criticality tasks are either suspended or degraded when any higher criticality task overruns its normal budget.

- In category 2, any task τ_j is always degraded whenever any other task overrun its normal budget. This is considered as **scheme 2 (degradation for HC and LC overrun)** where $DH_{j,p}$ = $DH_{j,s}$ = 1 and $DL_{j,p}$ = $DL_{j,s}$ = 1.

- In category 3, no tasks are degraded. This is considered as **scheme 3 (no degradation)** and hence $DH_{j,p}$ = $DH_{j,s}$ = $DL_{j,p}$ = $DL_{j,s}$ = 0.

- In category 4, τ_j is degraded whenever a task with criticality lower than $\mathcal{L}_j$ overruns its normal budget and hence $DH_{j,p}$ = $DH_{j,s}$ = 0 and $DL_{j,p}$ = $DL_{j,s}$ = 1. This is considered as **scheme 4 (degradation for LC overrun)**.

Further, to evaluate the effect of different degrees of task degradation on the schedulability performance, the parameter termed as 'Degradation Factor' is introduced. The value of 'DF' determines the maximum value of the degraded budget for a task in any given mode. The experiments were also performed to evaluate the effect of varying offset values and a few other parameters on the schedulability performance.

5.6.1 Task Set Generation

For a given old-mode M_p and the new-mode M_s, for our experiments, the parameters of the taskset were generated as follows:

- Utilisations of tasks belonging to the old-mode M_p and the new-mode M_s were generated using the UUnifast algorithm [78]. This algorithm ensures that the distribution of the generated utilisation values are unbiased.
- Time periods of tasks were generated in the range 100 to 1000ms and follow a log-uniform distribution [79].
- For changed tasks, a scaling factor of 2.0 was used to assign the parameters in the new-mode.
- Deadlines for tasks were set equal to their periods.
- The safe budget of each task in a mode was determined based on the period and utilisation selected: $C^s_{i,p} = U_{i,p}/T_{i,p}$
- The normal budget of each task in a mode was a determined based on the fixed multiplier of the safe budget: $C^n_{i,p} = C^s_{i,p}/CF$ (e.g., CF = 2.0).
- Each task belong to one of 4 criticality levels (1 to 4) and one of 5 task types (old-mode completed, aborted, unchanged, changed, wholly new). We assume the probability that a generated task τ_i is assigned with a specific criticality level is $1/4$. Similarly, the probability that a generated task τ_i is assigned with a specific task type is $1/5$. Hence tasks are generated with equiprobable criticality and types.
- The degraded budgets for τ_j are generated randomly in the range [0, DF*C^n_j] where DF denotes 'Degradation Factor' whose range is $0 \leq DF \leq 1$ (e.g., $C^k_{i,p}$ = DF*$C^n_{i,p}$ where DF = 0.5). Therefore, degraded budgets are always lower than the normal budget.

For the experiments, utilisation of the task was varied from 0.1 to 1.0[2]. For each utilisation value, 1000 tasksets were generated.

[2]For a given mode x, utilisation here is computed from the normal budget $C^n_{i,x}$ values only

5.6.2 Schedulability Tests Investigated

To the best of our knowledge, there is no existing study that provides a sufficient schedulability test under fixed-priority scheduling scheme for an uniprocessor system for a task model that considers both task degradation and offsets to handle mixed criticality and multi-mode aspects of the real-time system. Therefore, we consider the model and its fixed-priority schedulability test proposed in [7] where only the mode change aspects of a real-time system is considered. Henceforth, we refer to this model as MCO (Mode Change Only). We observe the schedulability of the MCO model by considering only the safe budgets for all tasks contained in the generated tasksets in both the modes.

5.6.2.1 Schedulability Test for MCO model

In the following, we present the equations that were used to compute worst-case response time for old-mode and new-mode tasks for the MCO model. Let us consider a mode transition from M_p (old-mode) to M_s (new-mode). Let Γ_p and Γ_s denote a set of tasks belonging to old-mode M_p and new-mode M_s respectively.

Worst-case response time equation for an old-mode task for the MCO model:

For any time $t > t_z$, the worst-case response time for an old-mode task $\tau_i \in \Gamma_p$ is given by solution to the following iterative equation:

$$\begin{aligned} R_i(t,t_z) = C^s_{i,p} &+ \sum_{j \in \mathbf{hp}(i)_O} \left\lceil \frac{t_z}{T_{j,p}} \right\rceil C^s_{j,p} + \sum_{j \in \mathbf{hp}(i)_N} \left\lceil \frac{R_i(t,t_z) - t_z - O_{j,s}}{T_{j,p}} \right\rceil_0 C^s_{j,s} \\ &+ \sum_{j \in \mathbf{hp}(i)_A} \left\{ \left\lfloor \frac{t_z}{T_{j,p}} \right\rfloor C^s_{j,p} + min(t_z - \left\lfloor \frac{t_z}{T_j} \right\rfloor T_{j,p}, C^s_{j,p}) \right\} \\ &+ \sum_{j \in \mathbf{hp}(i)_U} \left\{ \left\lceil \frac{t_z}{T_{j,p}} \right\rceil C^s_{j,p} + \left\lceil \frac{R_i(t,t_z) - \left\lceil \frac{t_z}{T_{j,s}} \right\rceil T_{j,s} - O_{j,s}}{T_{j,s}} \right\rceil_0 C^s_{j,s} \right\} \end{aligned} \tag{5.15}$$

Worst-case response time equation for a new-mode task for the MCO model:

For a given t_z and for any time $t > t_z$, the worst-case response time for a new-mode task is given by the solution to the following iterative equation:

$$R_i(t,t_z) = C^s_{i,s} + \sum_{j \in \mathbf{hp}(i)_O} C^s_{j,p} + \sum_{j \in \mathbf{hp}(i)_N} \left\lceil \frac{R_i(t,t_z) - O_{j,s}}{T_{j,s}} \right\rceil_0 C^s_{j,s} \tag{5.16}$$

In the above equations, t_z denotes the mode change instant; $\mathbf{hp(i)_O}$ denotes the set of old-mode completed tasks with priority higher than τ_i; $\mathbf{hp(i)_A}$ denotes the set of old-mode aborted tasks with priority higher than τ_i; $\mathbf{hp(i)_N}$ denotes the set of changed or wholly new tasks with priority higher than τ_i that use an offset relative to t_z; and $\mathbf{hp(i)_U}$ denotes the set of unchanged tasks with priority higher than τ_i. The function $\lceil x \rceil_0$ returns zero for $x \leq 0$ and x for $x > 0$ otherwise. The worst-case response time values for the old-mode and for the new-mode are considered by computing the maximum value for all possible t_z.

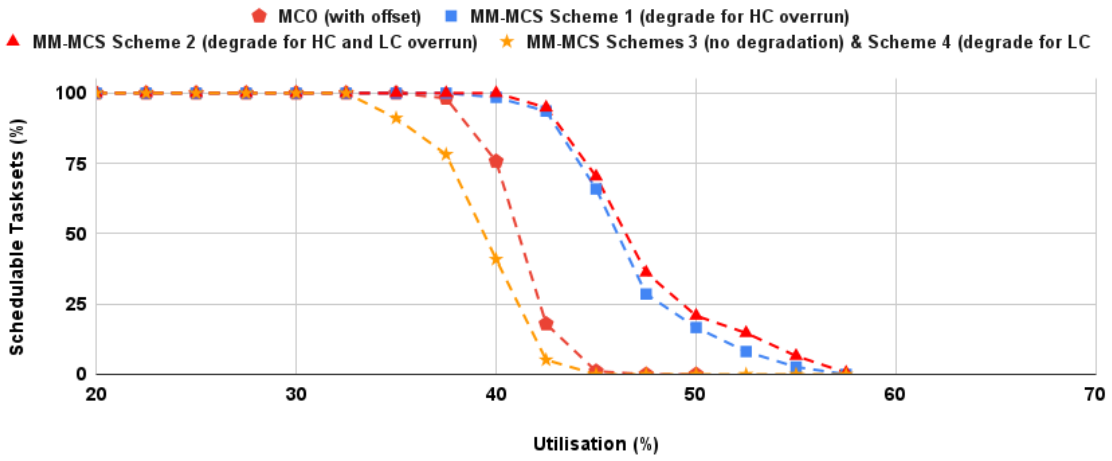

Figure 5.7: Evaluation for different degradation schemes considered for the MM-MCS model.

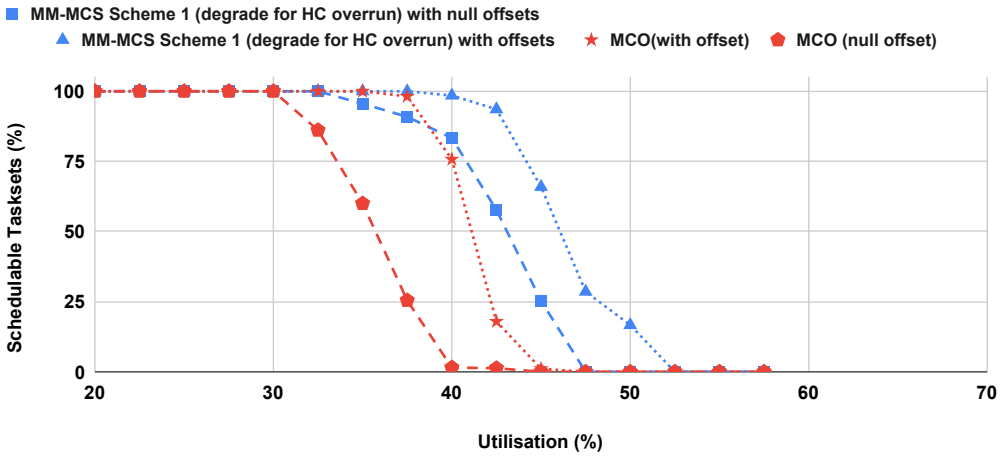

Figure 5.8: Varying the Offset.

5.6.3 Evaluation of the MM-MCS Model

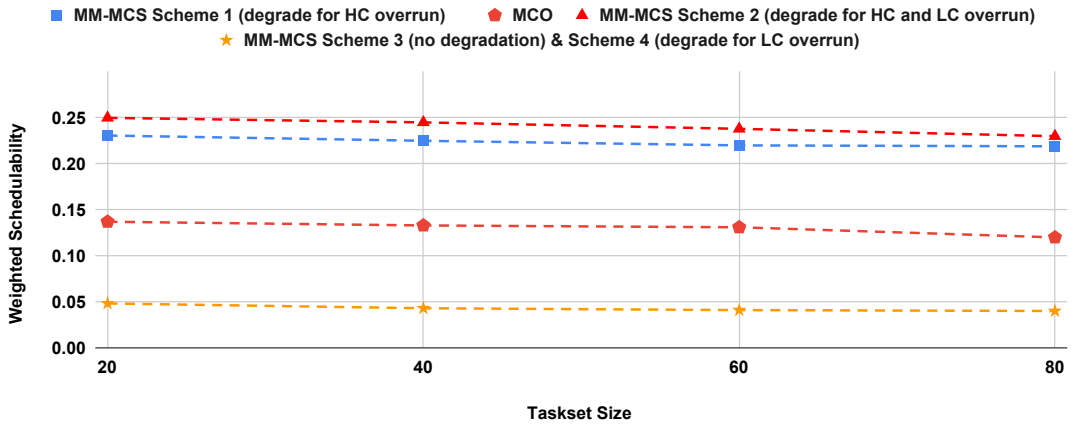

Figure 5.9: Varying the taskset size.

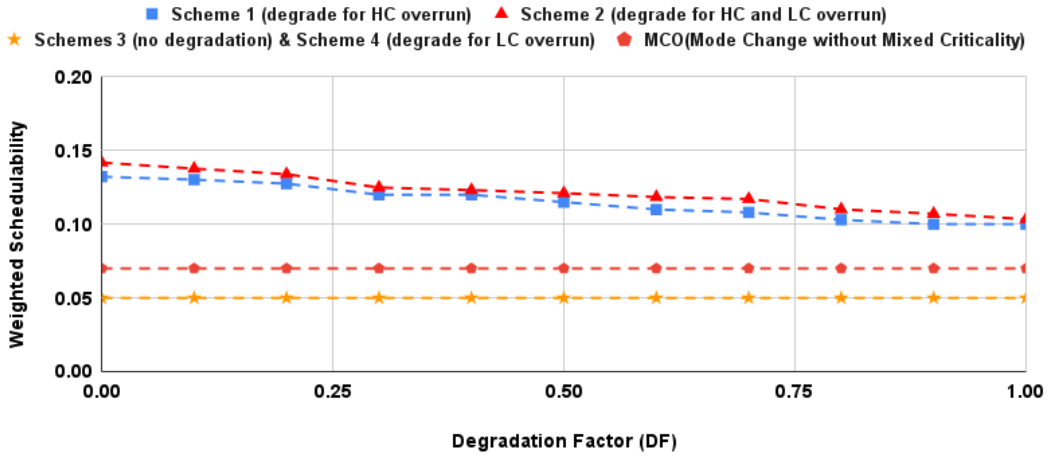

Figure 5.10: Varying the Degradation Factor (DF).

Figure 5.7, plots the percentage of tasksets generated considering different degradation schemes that were schedulable for a system of 20 tasks, with degradation factor of 0.2, task having a safe budget 2.0 times its normal budget (CF = 2.0), for schedulability tests for both MM-MCS and MCO models. We observe schedulability of MM-MCS for scheme 1 and scheme 2 outperforms all other schemes and the MCO model by a higher margin. This is expected as the schedulability test for the MCO model considers only the safe budget of a task in its analysis whereas the schedulability test for MM-MCS model can make use of additional information related to degraded budgets and behaviours. With the proposed

schedulability test for the MM-MCS, the worst-case interference pattern for tasks belonging to tasksets generated for degradation schemes 3 and 4 occur when the system executes in the state corresponding to highest criticality level *i.e.* in $S_{L,p}$ or $S_{L,s}$ where all tasks execute with their safe budgets and no tasks are degraded. For this reason, the performance of MM-MCS for schemes 3 and 4 are observed to be exactly same. It can also be observed that MM-MCS for scheme 2 seems to perform better than scheme 1 only marginally. Surprising, performance of MM-MCS for schemes 3 and 4 are worse than MCO. This can be attributed to the fact that analysis for the MCO model in [7] considers synchronous release at $t = 0$ which is not the case in the analysis for MM-MCS model. As the analysis for old- and new-modes for MM-MCS model are done separately, the assumption related to synchronous release at $t = 0$ is not possible leading to higher pessimism in the interference calculation.

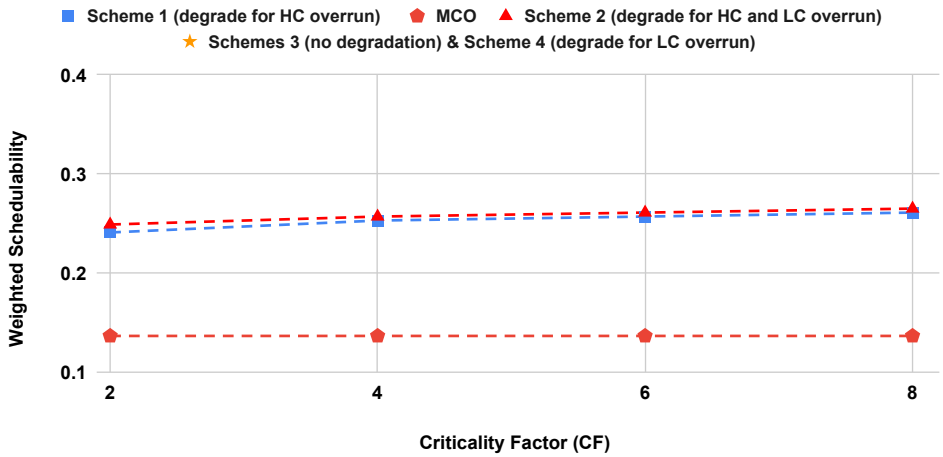

Figure 5.11: Varying the criticality factor (CF).

Figure 5.8, plots the percentage of tasksets generated (in this case only the degradation scheme 1 is considered for taskset generation) that were deemed schedulable for a system of 20 tasks, with a degradation factor of 0.2 and CF = 2.0. We observe that the proposed schedulability test for MM-MCS with null offsets (offset = 0) for all tasks outperforms that for MCO model with null offsets by a higher margin. The analysis for MM-MCS model with offset greater than 0 further improves on the schedulability of MM-MCS model with null offsets. Again, this is expected since with greater offset values interference from new-mode tasks are reduced giving improved schedulability performance. Also, schedulability of MCO - with offset seems to perform better than MM-MCS - without offset for utilisation up to 36 %. This is due to the fact that pessimism introduced in the MM-MCS schedulability test is higher than MCO model for this utilisation range.

In the following figures, the effect of varying each parameter on the weighted schedulability measure $W_x(q)$ [81] as a function of parameter q for schedulability test x is shown. The details of this measure is discussed in section 4.5. Figure 5.11 varies the criticality factor. Figure 5.10 varies the degradation factor (DF) from 0 to 1.0. Note that DF = 0 means that the task is suspended to handle the budget overrun and DF = 1 denotes that the task is degraded but the degraded budget is the same as that of its normal budget.

Figure 5.10 show that there is a decline in the schedulability performance for higher values of DF. They also show that MM-MCS for schemes 1 and 2 perform better than others with higher margin at lower DF values. Figure 5.9 shows that the MM-MCS for scheme 2 perform better than scheme 1 at the lower number of tasks but gradually approaches scheme 1's performance with an increase in the number of tasks. From Figure 5.10, it is clear that when $DF = 1.0$, MM-MCS performs better for schemes 1 and 2 than for schemes 3 and 4. In all of these schemes, even though degraded budgets and normal budgets of all tasks are equal, the difference in the number of releases interfering with safe budgets for each scheme can vary based on the values of $DH_{j,p}$, $DH_{j,s}$ leading to the difference in the schedulability performance.

Overall, the key observation of these figures is that the relationship between schedulability performance of the MM-MCS model for tasksets generated considering different degradation schemes, and the test considered for the MCO model on all graphs remains stable and that the schedulability test proposed for the MM-MCS model is an effective way of scheduling multi mode mixed criticality systems when task degradation and offsets are considered to handle the system overload.

5.7 Conclusion

As the embedded systems are evolving to handle the complexity and diversity of safety-critical functionalities, there is a growing need for models that can capture both mixed-criticality as well as multi-mode behaviour of real-time systems. To address this need, this chapter has introduced a new model termed as Multi-Mode Mixed Criticality Systems (MM-MCS) model. The notion of degradation and offsets are considered to handle the system overload. The proposed model is flexible and can capture requirements of promptness, schedulability and periodicity of new-mode tasks. The other contribution includes an algorithm to compute the offsets to have a safe, schedulable and prompt transition from one mode to another. This algorithm is guided by preferably reducing the offsets of higher criticality tasks. A sufficient schedulability test under fixed-priority scheduling scheme for the proposed model

was derived. The main objectives of the experiments based on synthetic tasksets were to show the usefulness of capturing behaviour parameter, degraded budgets and offset parameter of tasks. From the experiment results, it can be observed that appropriate values of offsets can make the mode transition schedulable. Similarly, it can be observed that the number of tasksets that were deemed schedulable for a given utilisation value depends on the values of DH and DL of tasks.

Chapter 6

Testbed for Practical Considerations in Mixed-Criticality System

Experiments based on synthetic task sets will be able to show only the impact of a specific degradation strategy on the schedulability of the tasksets considered. As the CA-MCS and MM-MCS models presented in this thesis are motivated by considering practical aspects of an automotive real-time system, we realized that mere experiments based on synthetic tasks is not enough and it was important to build a realistic testbed to show the impact on the safety and performance of the automotive functionalities for the degradation strategy considered by the proposed models with appropriate performance metrics. To cater to this need, a testbed[1] with representative automotive applications exhibiting different criticality levels was built. The testbed is highly flexible, scalable and can facilitate the implementation of MCS models. The testbed can be considered as a lab exercise in the university courses related to safety-critical real-time systems or autonomous driving.

The rest of the chapter is organized as follows. In Section 6.1, the background information related to automotive system design is presented. The architecture of the testbed is discussed in Section 6.2. In Section 6.3, details of the implementation of automotive applications running in the testbed along with modifications done to FreeRTOS scheduler to incorporate the proposed models are discussed. In Section 6.4, performance metrics and scenarios considered for experiments and their corresponding results are discussed. Finally, we conclude this chapter in Section 6.5.

[1]The work in this chapter has been published in [84]

6.1 Background

Automotive systems are rapidly transforming from mechanical to electronic platforms. The latest automotive innovations such as intuitive infotainment, self-driving abilities clearly show that automotive functionalities depend less on mechanical ingenuity than on software quality, execution, and integration along with the hardware in which they are hosted. An electronic control unit (ECU), is an embedded system that is used in automotive to control almost everything from dashboard to the engine. It is very common in modern high-end cars that are usually equipped with 70-80 ECUs performing both safety and non-safety critical functionalities. The conventional way of designing ECUs is done by assigning one functionality per ECU [85]. For example, cruise control functionalities are implemented in one ECU, whereas functionalities of collision avoidance are implemented in another, even though they both rely on common sensors like a radar for their operation. With the increase in the number of functionalities, the networking of these ECUs becomes a complex task that is getting tougher with every additional ECU.

6.1.1 Distributed Control Architecture

The traditional architecture of an embedded system in automotive involves ECUs, sensors and actuators connected through dedicated channel following a point-to-point topology. Due to the increase in electronic components, the number of individual connections has grown significantly resulting in an increased harness weight. It also affects the manufacturing time, fuel efficiency and reliability of the vehicle. This has led to the development of distributed control architecture as shown in Figure 6.1. In this architecture, ECUs, sensors modules, and actuator modules are connected through a common network bus such as CAN, LIN and Flexray [86]. The sensor module can send real-time data to all the ECUs connected to the network. The actuator control module can receive and process actuator commands calculated and transmitted by ECUs. The advantage of this architecture is the increased performance of functional aspects of applications such as combined brake systems, suspension, steering control, etc. Applications with similar sensor processing and actuators, controlling requirements can be made to run in the same ECU, thereby avoiding the redundant computations.

6.1.2 Controller Area Network

Controller Area Network (CAN) protocol is a multi-master serial bus standard protocol designed for multiplexing electrical wiring within automotive, which was later incorporated into various contexts

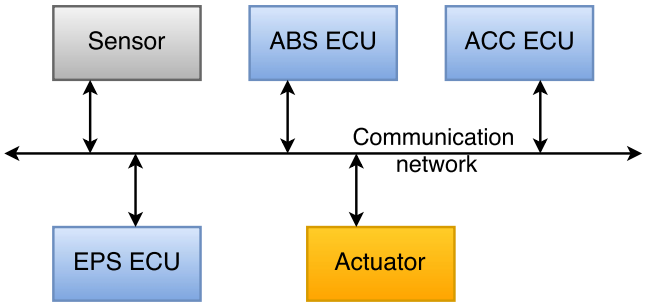

Figure 6.1: Distributed control architecture.

such as industrial automation, agriculture, etc. It was developed by Robert Bosch GmBH [86]. Two or more ECUs are required on the CAN network for the communication to happen. The Figure 6.1 shows the typical communication among different ECUs in the CAN. All ECUs are connected to each other through a two-wire bus. The wires are twisted pair cables terminated with 120-ohm resistors across the ends. Each node/ECU has a central processing unit, a CAN controller, and a transceiver. Sensors, actuators, and controlling devices are connected to the host processor. The CAN controller is responsible for framing CAN message in the data link layer. It handles transmission and reception of serial bits onto the bus. The CAN transceiver is responsible for converting the CAN bus voltage level to CAN controller voltage level and vice versa. CAN data frame format is classified into two types. They are standard frames and extended frames. The standard frame (CAN 2.0A) has an 11 bit CAN identifier (CAN-ID), and the extended frame (CAN 2.0B) has a 29-bit CAN-ID. CAN protocol follow bit-wise wired AND logic in the bus arbitration. The CAN-ID with the dominant bit (0) will have a higher priority than the one with a recessive bit (1). CAN supports bit rates up to 1 Megabits per second (Mbps) for a network length of 40 m. In the experimental platform, CAN speed is limited to 500 Kilobits per second (Kbps). CAN with Flexible Data-Rate (CAN-FD) protocol standard allows the increase in the bit rate by a factor of eight after CAN arbitration.

6.1.3 ECU Consolidation

With the increase in the number of safety functionalities and the number of ECUs, the notion of an ECU per functionality is becoming outdated. This situation has lead to design approaches to consolidate different functionalities onto fewer ECUs, a major challenge haunting the automotive industry. When consolidated, applications running on an ECU may contain safety-critical and non-safety critical software components. For example, an airbag ECU may run a highly critical crash detection code along with a non-critical diagnostics code as well as software related to the dashboard display. Several ap-

S.No.	Aspects of the proposed MCS task models that can be studied with experiments	Experiments based on synthetic tasksets	Experiments based on the experimental testbed
1	Impact of the parameters considered for the generation of tasksets on the taskset schedulability.	✓	X
2	Impact of considering context-aware choice of degradation of tasks on the performance of the safety functionality.	X	✓
3	Impact of the suspension of tasks on the safety of the vehicle.	X	✓
4	Impact of considering different degradation schemes (degradation of higher criticality tasks and/or lower criticality tasks) on the taskset schedulability.	✓	X
5	Impact of considering offsets for tasks on the taskset schedulability.	✓	X

Table 6.1: Types of experiments required to study the significant aspects of the proposed MCS task models.

proaches are being tested to integrate more functionalities onto fewer powerful ECUs without increasing the complexity of testing effort [87]. These requirements make the automotive systems to be considered as typical Mixed Criticality System (MCS). Studies like [57] are purely motivated by the presentation of applications from automotive domain exhibiting mixed-criticality behaviour.

Research projects involving design and implementation of experimental platforms related to automotive have been developed either to test an algorithm or to verify the performance of protocols designed for the detection and diagnosis of network faults. For example in the project Auto Plug [88], the automotive testbed is used to test, update, diagnose and verify controls software at run-time. Similarly, in this research, the purpose of the experimental platform is to evaluate the impact of different degradation strategies considered in the existing Mixed Criticality System (MCS) models and scheduling algorithms, particularly in terms of the practical impact that service degradation has on automotive systems. In the Table 6.1, different aspects of the proposed MCS task models and the type of experiments required to study them are tabulated. In order to study the impact of different values of parameters such as taskset utilisation, number of tasks, different degradation strategies etc., on the taskset schedulability, experiments were done by generating synthetic task sets. But, it was realized that there was a need to build a realistic automotive platform to show the impact on the safety and performance of the automotive

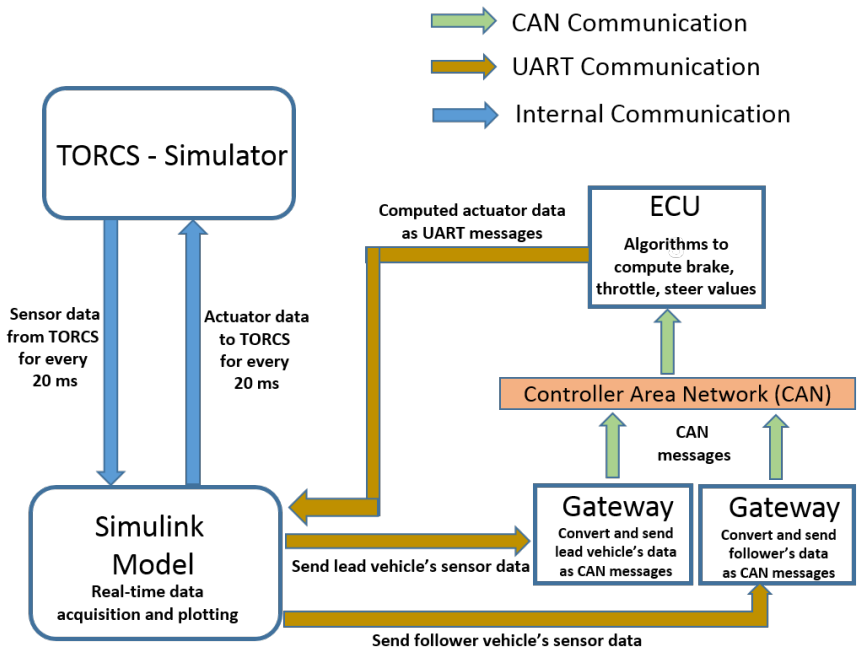

Figure 6.2: Architecture of the Testbed.

functionalities for the degradation strategy considered by the proposed models. To build the experimental platform, appropriate system architecture with hardware and software were chosen so that they comply with ISO 26262 safety and assurance level requirements. In the following sections, selected safety-critical applications, architecture of the experimental platform, connections between hardware components and the communication protocol considered are discussed.

6.2 Testbed Architecture

The testbed is an Hardware-In-The-Loop simulator with the following four major components:

1. The Open Racing Car Simulator (TORCS) [89]
2. Simulink model
3. Gateways
4. Electronic Control Unit (ECU)

The testbed and its architecture is shown in Figures 6.3 and 6.2 respectively. TORCS is an open-

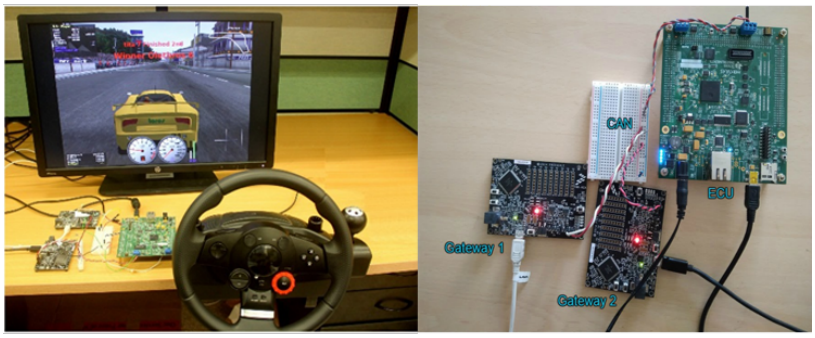

Figure 6.3: Testbed.

source vehicle dynamics simulator and has an ability to provide real-time sensor and actuator data at regular intervals. These components are connected using different communication protocols. Both TORCS and the simulink model are hosted on the same personal computer (PC) and hence they both communicate with existing internal communication facility available in the PC. The ECU and gateways are connected to the same Controller Area Network (CAN). The ECU communicates to the PC using Universal Asynchronous Receiver/Transmitter (UART) communication. These components along with these different communication networks form a distributed control architecture. The description of each component in the testbed is given in the following subsections.

6.2.1 The Open Racing Car Simulator

The Open Racing Car Simulator (TORCS) [89] is an open-source simulator, written in C++ language and is popular in the field of automotive research. It is licensed under the General Public License (GNU GPL). TORCS is designed to enable Artificial Intelligence (AI) drivers to race against one another while allowing the player to monitor and control the racing vehicle using devices such as a keyboard, mouse, or steering wheel. The testbed simulates two vehicles, a lead vehicle controlled by TORCS and a follower vehicle controlled by the Electronic Control Unit (ECU). TORCS simulates these cars from which sensor data such as wheel speeds, vehicle speeds, brake command, etc., can be captured. TORCS has inbuilt automotive applications such as Anti-Slip Regulation (ASR), Anti-lock Braking System (ABS), and Cruise Control. A typical race car in TORCS is shown in Figure 6.4.

Sensor Data		
Parameter	**Range**	**Description**
Velocity X	(-infinity, +infinity)(km/h)	Velocity of the vehicle along the longitudinal axis.
Velocity Y	(- \infty, +\infty) (km/h)	Velocity of the vehicle along the transverse axis.
Velocity Z	(- \infty, +\infty) (km/h)	Velocity of the vehicle along the Z axis.
Position X	(- \infty, +\infty)	X coordinates of the vehicle's position on the map.
Position Y	(- \infty, +\infty)	Y coordinates of the vehicle's position on the map.
Position Z	(- \infty, +\infty)	Z coordinates of the vehicle's position on the map.
Heading Error	(-1, 1) (radian)	Error between track heading and vehicle heading (at current location).
Lateral Error	(-1, 1) (m)	Lateral error between track centreline and car's centre of gravity (at current location)
Road Curvature	(-1, 1)	Curvature of track (at current location), left turns = +ve curvature, right turns = -ve curvature.
Road Distance	[0, 10000]	Total distance covered by the car
Yaw	degrees	Rotation of the car about the z-axis.
Actuator Data		
Parameter	**Range**	**Description**
Acceleration	[0,1]	Acceleration pedal (0 means no gas, 1 full gas).
Brake	[0,1]	Brake pedal (1 means full brake, 0 means no brake)
Steer	[-1,1]	Steering value: -1 means full right and +1 means full left.
Gear	-1, 0, 1,...,6	Gear value.

Table 6.2: Sensor and actuator data from TORCS.

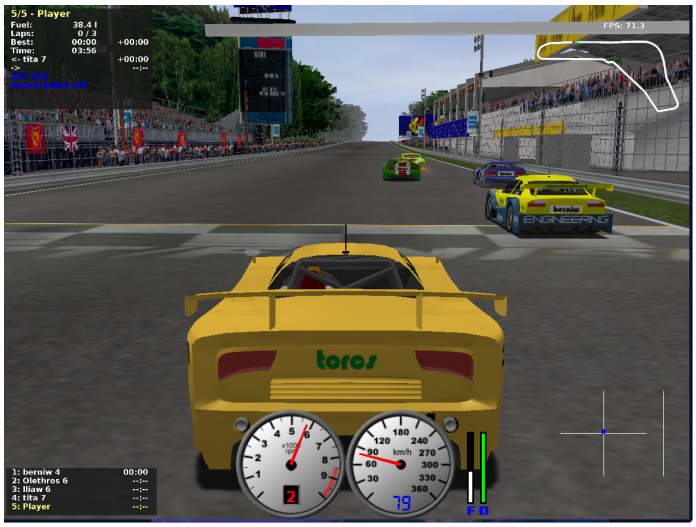

Figure 6.4: TORCS in action.

6.2.2 Simulink Model

The Simulink model, as shown in Figure 6.5 runs in a Personal Computer (PC) and acts as an interface between TORCS and the Gateway. The interface to the TORCS block in the model consists of a code with which TORCS and the Simulink model can be configured to share a common memory location to read and update sensors and actuators values periodically. It was observed that 20 ms was sufficient enough to produce the required output. The values updated by the TORCS are sent to lead vehicle control block which further consists of Simulink blocks to control the leader vehicle. As the algorithms to control the follower vehicle are implemented in the ECU, sensor data need to be sent via gateways to the ECU to compute follower vehicle's actuator commands. Therefore, the sensor values corresponding to the leader and the follower vehicles are first extracted and then sampled using sample and hold block before sending as UART messages to gateways. For this purpose, 'to instrument' block is used. After processing the sensor data, the ECU returns actuator commands back to the Simulink model via UART. The advantage of using Simulink as an interface is that the sensors and actuator commands can be tracked and analysed in real-time. For example, a plot can show both current values and the Root Mean Square (RMS), of sensor values. These values can be later used to analyse the performance of applications in different modes of operation.

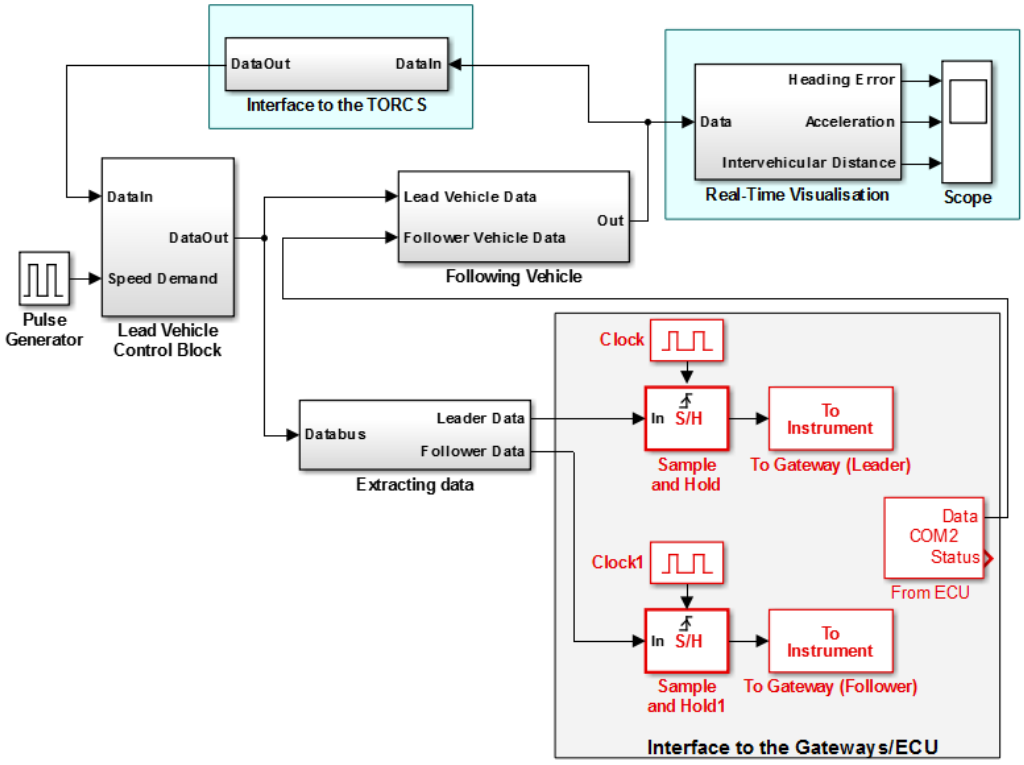

Figure 6.5: Simulink Block Diagram.

6.2.3 Gateway

The Freescale kinetis development board [90] is used as a gateway. The main role of a gateway is to receive the sensor values of both vehicles from the Simulink model as serial data via UART and convert them into CAN messages. Each CAN message is mapped to a unique sensor data and sent via CAN that is received by the ECU where the control algorithms in the form of tasks reside. There are two gateways used in the platform, one for lead vehicle and the other for follower vehicle. The sensor values shown in the Table 6.2 are sent as CAN messages.

6.2.4 ECU

Texas Instruments (TI)-Hercules development board TMS570LS3137 [91] is used as an ECU. The Applications on the ECU are designed to handle the control of the follower vehicle in TORCS. The ECU runs FreeRTOS [92], a pre-emptive fixed-priority based real-time scheduler. The basic control algorithms are running as real-time tasks on FreeRTOS for the applications. One or more tasks collectively achieve control functionality. For example in cruise control, the detection of lead vehicle is done by tasks corresponding to V2V communication and radar. The ECU processes sensor values and outputs the brake, steering and acceleration commands to the Simulink model through the UART communication port.

6.3 Testbed Implementation

The testbed consists of safety-critical functionalities related to lead vehicle detection, longitudinal and lateral control of the vehicle. The Adaptive Cruise Control (ACC) and Collision Avoidance (CA) performs the longitudinal control considering the inter-vehicular distance while the vehicle's lateral control performed by Steering Control (SC) application. The applications considered are of different Automotive Safety Integrity Levels (ASIL) based on ISO 26262's risk classification scheme and share sensor values including velocity, position, acceleration, etc. and actuator commands such as brake, steering and throttle of the vehicle. Algorithms used in [93] were referred and modified for the implementation of these functionalities. Details of the applications implemented in the testbed to observe the performance of the proposed CA-MCS and MM-MCS models are discussed in the following subsections.

Algorithm 2 Algorithm for pseudo-radar task.

function PSEUDORADAR($followerPos$, $leaderPos$)
  $distX$ = $(followerPos[0] - leaderPos[0])^2$
  $distY = (followerPos[1] - leaderPos[1])^2$
  $distance = \sqrt{(distX + distY)}$
  $viewAngle = 1.05rad$
  $range = 70m$
  $targetAngle = \tan^{-1}(\frac{distY}{distX})$
  $leftBound = folYaw - viewAngle$
  $rightBound = folYaw + viewAngle$
  if $distance < range$ **then**
    if $targetAngle \leq rightBound \wedge targetAngle \geq leftBound$ **then**
      $vehicleAhead = 1$
      $return\ distance$
    else
      $vehicleAhead = 0$
    end if
  end if
end function

Algorithm 3 Algorithm for V2V communication task.

function V2V($followerPos$, $leaderPos$)
  $distX$ = $(followerPos[0] - leaderPos[0])^2$
  $distY = (followerPos[1] - leaderPos[1])^2$
  $distance = \sqrt{(distX + distY)}$
  $return\ distance$
end function

6.3.1 Lead Vehicle Detection (LVD)

The lead vehicle detection functionality is implemented using two tasks, namely pseudo-radar and vehicle to vehicle (V2V) communication. They are assumed to be certified to ASIL C and ASIL B respectively. In the testbed, the euclidean distance is calculated based on the position values of the lead and the follower vehicle by both the pseudo-radar and V2V communication task. V2V task calculates the euclidean distance between vehicles in all road conditions (including curve roads). This way, when the follower vehicle is manoeuvring through a curve would have knowledge on where the lead vehicle is and calculates the necessary throttle and brake values accordingly. This results in better performance of the follower vehicle with lesser harsh acceleration or harsh braking while following the lead vehicle.

Pseudo-Radar Sensing, on the other hand, adopts the real-time position data of the lead vehicle and

translates it to a limited range of radar sensing capability. It is assumed to be instantaneous which is why real-time data is used. Even though it always receives the position values of the lead vehicle, it determines whether the lead vehicle is in the range of its pseudo radar capabilities *i.e.* the pseudo radar task computes the Euclidean distance only if the lead vehicle is within 120-degree field-of-view and within a range of 30 meters. This simulates the actual radar since 120 degree field-of-view may not be sufficient to detect the lead vehicle in curves.

Algorithms 2 and 3 present the logic for pseudo-radar and V2V tasks respectively. Here, the euclidean distance is computed based on x and y coordinates of the follower and the leader vehicles. Similarly, the angle between vehicles is also computed based on vehicle co-ordinates, added to the follower vehicle's yaw and then compared with a reference angle to see whether they are within the right bound and left bound *i.e.* within 120 degrees field of view. If the Euclidean distance between the vehicles is less than 70 m and if the leader vehicle is within 120 degree field of view, $vehicleAhead$ is set to 1 otherwise, $vehicleAhead$ is set to 0.

Algorithm 4 Algorithm to compute throttle.

function $Throttle()$ ▷ compute throttle based on the value of $vehicleAhead$
  if ($vehicleAhead == 1$) **then** ▷ condition for constant headway mode
    if ($ONOFF == 0$) **then** ▷ no budget overrun of tasks
      $accDem$ = $distance$ + (vel_l - vel_f) - (vel_f * $timeHeadway$) ▷ Helly's model
      $throttle$ = $accDem$ - acc_f
    end if
    if ($ONOFF == 1$) **then** ▷ Under budget overrun of tasks
      if ($distance > thresholdDistance$) **then**
        $throttle = 1$ ▷ full throttle
      else
        $throttle = 0$
      end if
      if ($distance < brakingDistance$) **then**
        $throttle = -1$ ▷ full braking
      end if
    end if
  end if
  if ($vehicleAhead == 0$) **then** ▷ condition for constant velocity mode
    $accDem = -(vel_f - vel_{set})$
    $throttle = K_p^{'} * (accDem - acc_f) + K_i^{'} * \int (accDem - acc_f) + K_d^{'} * \frac{d(accDem - acc_f)}{dt}$
  end if
  $return\ throttle$
end function

Algorithm 5 Algorithm for the cruise-control task.

function $Cruise\ Control\ ()$
  if (($distTravelled \geq curveTriggerDist$) $\wedge$ ($distTravelled < curveEndDist$)) **then**
    $curveAhead = 1$
  else
    $curveAhead = 0$
  end if
  if ($DynamicSpeedAdaptation == 1 \wedge curveAhead == 1$) **then**
    $throttle_1$ = $Throttle()$
    $accDem = -(vel_f - min(vel_{curve}, vel_{set}))$
    $throttle_2 = K_p * (accDem - acc_f) + K_i * \int (accDem - acc_f) + K_d * \frac{d(accDem - acc_f)}{dt}$
    $throttle = min(throttle_1, throttle_2)$
  else
    $throttle$ = $Throttle()$
  end if
  $return\ throttle$
end function

6.3.2 Adaptive Cruise Control (ACC)

The safety functionality of the Adaptive Cruise Control (ACC) is implemented as cruise control (CC) task which computes the necessary throttle command based on the distance between the lead vehicle and the follower vehicle. Algorithm 5 presents the logic for the CC task. A positive throttle value denotes a positive acceleration leading to increased velocity while a negative throttle value produces a braking effect and therefore decreases the vehicle's velocity. Based on the distance between the follower and the lead vehicle, CC is designed to operate in two modes viz 'constant headway' or 'constant velocity'. Here, headway can be either time headway or space headway. Time headway is the difference between the time when the front of the lead vehicle arrives at a point on the road and the time the front of the follower vehicle arrives at the same point (in seconds) whereas the space headway is the difference in position between the front of a vehicle and the front of the next vehicle (in meters). In constant headway mode, CC implements an algorithm either to maintain a specific time headway or a specific space headway between the vehicles such that under heavy braking of the lead vehicle, there is sufficient time for the follower vehicle to apply brakes to maintain a safe distance. To achieve this, CC uses a Helly's car following model [94]. In this mode, it is possible for the CC task to switch from PID based control mechanism to ON-OFF control mechanism to maintain space headway instead of time headway. The control algorithm based on PID is more precise leading to a smoother acceleration when compared to the ON-OFF controller. The equation used by PID control mechanism to compute the necessary throttle for the follower vehicle to maintain the space headway relies on the distance between the vehicles ($distance$), time headway ($timeHeadway$), the velocity of the follower vehicle (vel_f), acceleration of the follower vehicle (acc_f), the value of the required acceleration demand ($accDem$), proportional gain (K_p), integral gain (K_i) and the derivative gain(K_d). The value of acceleration demand conveys the required acceleration of the vehicle to reach given set point velocity or distance. For PID control mechanism, an algorithm presented as a Simulink model in [93] was implemented. With ON-OFF strategy, the vehicle is either given higher throttle or applied sufficient brakes depending whether the distance between the lead and the follower vehicles is within the given threshold or not (denoted as $thresholdDistance$ whose value is around 30 m) whereas if their distance is less than $brakingDistance$, then brakes are applied (negative throttle). Since PID requires computations related to integral and derivatives of the values, execution time of CC task with PID is more than the ON-OFF control. In constant velocity mode, the vehicle is given sufficient throttle to maintain the desired set velocity and is calculated based on the follower vehicle's velocity (vel_f), set velocity (vel_{set}), proportional gain ($K_p^{'}$), integral gain (K) and the derivative gain($K_d^{'}$) of the PID control mechanism.

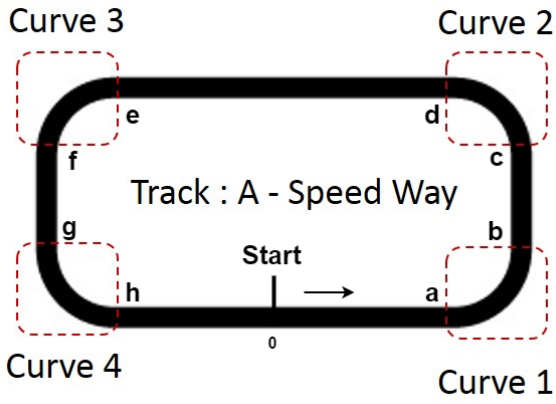

Figure 6.6: Map of the race track.

CC task also implements Dynamic Speed Adaptation (DSA) functionality to detect and indicate the presence of curves and to bring down the vehicle's velocity to a predetermined value (vel_{curve}) for a safe vehicle manoeuvre through the curves. The implementation discussed in [95] is used for DSA which computes the required throttle based on the map data to derive the distance between the vehicle and the curve ahead. The map data consists of the distance from the start point of the track to start and endpoints of each curve present in the chosen track. As shown in Figure 6.6, the start point of the curve is denoted as 0. The distance from the start point of curve 1 can be given as $a - 0 = a$ while its endpoint can be given as $b - 0 = b$. Based on this information, for each curve, the distance from the start point of the track to start and endpoints of each curve is calculated offline. As the vehicle is required to manoeuvre through each curve with a specific velocity (vel_{curve} whose value is 18.0 m/s), based on follower vehicle's velocity, a positive or negative throttle is applied well before the vehicle reaching the start point of the curve to ensure that vehicle's velocity is lower or equal to vel_{curve}. For the track considered, vel_{curve} is assumed to be the same for all 4 curves. If the Dynamic Speed Adaptation (DSA) is active in a specific mode, computation of required throttle command of the follower vehicle is decided by CC task considering the curve ahead, lead vehicle ahead and the constant set point velocity.

Based on the above functionalities of CC, it is possible to consider two ways of degrading the CC task while operating in 'constant headway mode'. With type 1 degradation, CC operates with ON-OFF control instead of PID leading to reduced execution time. With type 2 degradation, CC ignores the DSA functionality also leading to reduced execution time.

6.3.3 Steering Control (SC)

Algorithm 6 Algorithm for Steer Control task.

1: **function** STEER CONTROL
2: **if** $distance > 5$ **then**
3: $K = 0.5 * (angularVelocity + 0.06 * headingError + lateralError)$
4: $gain = -0.06 * \frac{d(headingError)}{dt} - 0.06 * \frac{d(lateralError)}{dt} - K$
5: **end if**
6: **if** distance $\leq$ 5m **then**
7: gain = 0.5
8: **end if**
9: **if** $gain > 0.5$ **then**
10: return 0.5
11: **end if**
12: **if** $gain < -0.5$ **then**
13: return (-0.5)
14: **end if**
15: **if** $-0.5 \leq gain \leq 0.5$ **then**
16: return gain
17: **end if**
18: **end function**

The safety functionality of Steering Control (SC) task is to maintain the vehicle's position at the centre of the lane. The algorithm for steering control is presented in Algorithm 6. Steer command is computed based on the angular velocity (denoted as $angularVelocity$), heading error (denoted as $headingError$) and lateral error (denoted as $lateralError$). The heading error is equal to the difference between path heading and vehicle heading at the reference point along the path. It is a measure of how well the vehicle is aligned with and moving in the direction of the desired path. The vehicle's lateral error is defined as the distance between the vehicle control point and the closest point along with the vehicle desired trajectory.

In 'constant headway mode' or in 'constant velocity mode', based on the current track segment that the vehicle is in and the difference between the angle of the track and the vehicle's heading, a PID based control algorithm determines how much steering is required to keep the vehicle in the centre of the track. In 'collision avoidance mode' the steering control algorithm return steer value to achieve full left steer to avoid a head-on collision with the leader vehicle.

6.3.4 Collision Avoidance (CA)

Algorithm 7 Algorithm for collision avoidance task.

```
1:  function COLLISION AVOIDANCE
2:      if distance ≤ 10m then
3:          if d(acc_l)/dt < 0 then
4:              if d(vel_f)/dt ≥ 0 then
5:                  issue collision warnings
6:              end if
7:          end if
8:      end if
9:      if distance ≤ 5m then
10:         if TTC <= 4 then
11:             apply emergency braking
12:         end if
13:     end if
14: end function
```

Collision Avoidance Application (CAA) provides two safety functionalities. First, to initiate a warning whenever the distance between vehicles is less than 10m so that a potential head-on collision can be avoided. Second, to initiate an emergency braking to avoid an imminent head-on collision between the follower and the lead vehicles. Algorithm 7 presents the logic for CAA and is derived from the theory proposed in [96] that relies on parameters such as distance between the vehicles ($distance$), the velocity of the follower and leader vehicles (vel_f and vel_l) and acceleration of the leader vehicle (acc_l) to initiate a warning. The warning criteria can be summarised as:

1. The leader vehicle continues to decelerate at the current rate to a stop.
2. The follower vehicle maintains the same constant speed.
3. The minimum distance between the vehicles during or after braking is 10 meters.

The above assumptions were incorporated as if-else conditions, which then used to determine the need for warnings in the form of LED blinks in the experiments. To determine the need for emergency braking, apart from the distance between the vehicles, an additional parameter called 'Time To Collision (TTC)' is used. The value of TTC can be calculated from the following expression:

$$TTC = \frac{\textit{Distance between the vehicles}}{\textit{Leader vehicle's velocity} - \textit{Follower vehicle's velocity}}$$

Emergency braking is initiated whenever the following two conditions are satisfied.

1. The minimum distance between the vehicles is less than or equal to 5 meters.
2. TTC is less than 4 seconds (as suggested in [97]).

6.3.5 Controller Area Network Task

The ECU takes in the sensor readings from TORCS from the kinetis board which are sent as messages through the CAN interface. The sensor values tabulated in the Table 6.2 are sent from gateways as CAN messages and then stored in Hercules ECU. When a message is sent out on the kinetis board with message ID in the range 0×81 to $0 \times 8A$, an interrupt is raised on the Hercules, and activates CAN task which then acts to identify the message and updates the variables with respective sensor values. For this purpose, each sensor data is associated with a specific message ID and with a specific variable.

Apart from sensor data, an additional message is assigned to receive the trigger for budget overrun from the gateway. The value of this message triggers an additional 'for loop' in a specific task which simulates the budget overrun scenario. When all tasks finish their execution, the actuator parameters such as throttle, steering and brake values are sent back to Simulink through the Universal Asynchronous Receiver Transmitter (UART) port.

6.3.6 Modes of Operation of the Testbed

Based on the algorithms discussed for various automotive applications we can see that the testbed can operate in one of 3 possible modes. They are,

- 'constant velocity mode' - the major safety goal of the system in this mode is to maintain a fixed velocity of the follower vehicle.
- 'constant headway mode' - the major safety goal of the system in this mode is to maintain a predefined space headway or time headway between the follower vehicle and the lead vehicle.
- 'collision-avoidance mode' - the major safety goal of the system in this mode is to avoid a head-on collision with the lead vehicle.

Figure 6.7 depicts the conditions when the transition from one mode to another happen. All tasks and their corresponding functionalities that are active in each mode are described in the Table 6.3 and their

Mode ↓	Tasks					
	V2V (ASIL B)	**Pseudo-Radar (ASIL C)**	**Cruise Control (ASIL D)**	**Collision Avoidance (ASIL D)**	**Steering Control (ASIL D)**	**CAN (ASIL D)**
Cruise Control - Constant Velocity Mode	Detect the lead vehicle if within 70m.	Detect the lead vehicle if within 70m and within 120 degrees field of view.	1. Apply throttle or brake to maintain a constant vehicle velocity. 2. Dynamic Speed Adaptation for a smooth maneuvering through curves.	Not active in this mode.	Apply steering to ensure vehicle is moving along the center of the track.	Receive CAN messages and store it in the corresponding variables.
Cruise Control - Constant Headway Mode	Detect the lead vehicle if within 70m.	Detect the lead vehicle if within 70m and within 120 degrees field of view.	1. Apply throttle or brake computed as per the PID or ONOFF controller mechanism to maintain constant headway between the lead vehicle and the follower vehicle. 2. Dynamic Speed Adaptation for a smooth maneuvering through curves.	1. Provide warnings if distance between the lead and the follower vehicle is within 10 m.	Apply steering to ensure vehicle is moving along the center of the track.	Receive CAN messages and store it in the corresponding variables.
Collision Avoidance Mode	Detect the lead vehicle if within 70m.	Detect the lead vehicle if within 70m and within 120 degrees field of view.	Not active in this mode.	1. Apply emergency braking if the distance between the lead and the follower vehicle is less than 5m.	Apply left steer or right steer to avoid head-on collision with the lead vehicle.	Receive CAN messages and store it in the corresponding variables.

Table 6.3: Tasks and their functionalities in different modes.

Budgets (Ticks)	**Tasks**							
	Pseudo-Radar	**V2V**	**Cruise Control (constant velocity mode)**	**Cruise Control (constant headway mode)**	**Collision Avoidance (constant headway mode)**	**Collision Avoidance (collision avoidance mode)**	**Steer Control**	**CAN**
Normal	**8771**	8153	151540	162524	5423	5810	4618	9950
Safe	**17861**	16532	305643	324590	9543	10234	8432	10785

Table 6.4: Task budgets in different modes.

corresponding budgets are shown in the Table 6.4. When the distance between vehicles is less than or equal to 70 meters, the value of $vehicleAhead$ variable becomes 1 triggering a mode change request from 'constant velocity' to 'constant headway' mode. Similarly, when the distance between vehicles become greater than 70 meters, the value of $vehicleAhead$ becomes 0 and the mode change request to 'constant velocity mode' is triggered. Whenever the distance between vehicles becomes less than or equal to 10 meters, collision warnings are given and when the distance between vehicles further reduces to 5 meters or below, the system switches immediately to the collision avoidance mode where the follower vehicle is applied emergency brakes along with left steer to avoid a head-on collision with the lead vehicle.

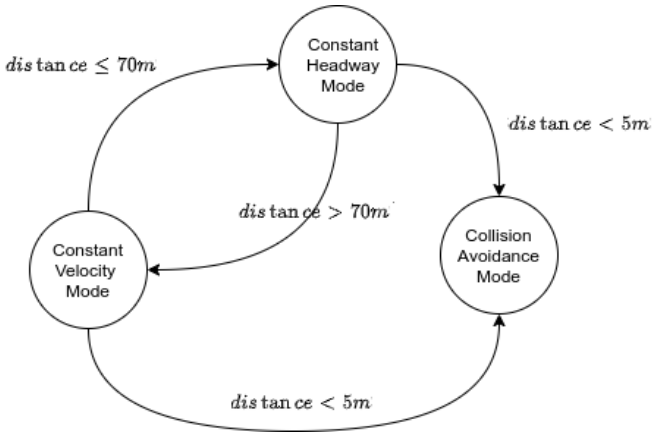

Figure 6.7: Different modes of the follower vehicle.

6.3.7 FreeRTOS Implementation

In the following, modifications that are done to FreeRTOS APIs to consider additional task parameters and to detect and handle budget overrun as per the proposed models for MCS are discussed. The code for task initialization are executed only once when the system starts while that for monitoring of budget overruns and execution of tasks are performed at run-time periodically. Figure 6.8 depicts the flow chart for the FreeRTOS implementation.

A new task can be created and initialized using the function xTaskCreate() that takes arguments to allocate the memory for the stack and the Task Control Block (TCB) and stack. Additionally, this function also initializes look-up tables for each possible mode of operation to store and retrieve information related to budgets, behaviors, offsets and criticality of tasks that are active in a specific mode and to keep track of the change in behaviors of tasks at run-time in the current mode of operation. The flow chart for FreeRTOS implementation is depicted in Figure 6.8.

When the TORCS simulator starts to run, the sensor data that arrive as CAN message from the gateway microcontrollers are received by the CAN controller of the ECU. CAN task is allocated with the highest priority level and is triggered by the interrupt generated upon the message arrival process. The data frame of the message is received and stored. After acknowledging the message, depending upon the message id, the CAN task sends a notification to specific tasks which are in 'pending' state to continue with their execution. As the tasks execute the control algorithms, the variable $ulTotalRunTime$ keeps track of the number of ticks that have occurred from the system start time. The latest time at which context-switch has occurred is stored in $ulTaskSwitchedInTime$ and the

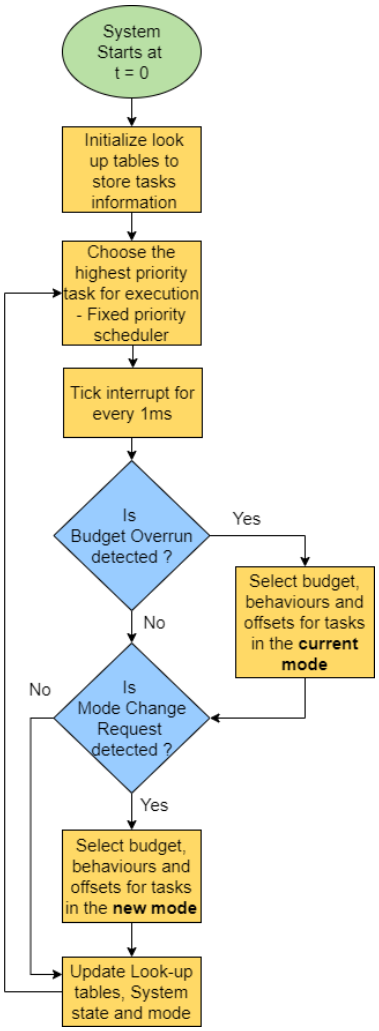

Figure 6.8: Flowchart for FreeRTOS Implementation.

variable $ulRunTimeCounter$ keeps track of the amount of time for which the active task has executed from the previous context-switch instance.

The function $xTaskIncrementTick()$ is configured to execute for every 1 ms to check whether budget overrun of an active task has occurred or not based on the values of $ulTotalRunTime$ and $ulRunTimeCounter$. The budget overrun of an active task is detected by comparing the values of $ulRunTimeCounter$ of the task and the current budget allocated to it. The $budgetoverrun$ flag is set to 1 if the value of $ulRunTimeCounter$ of the active task is greater than the budget allocated to it. Then, depending on the behavior of this overrun task, budgets and behaviors of other tasks are updated accordingly. Depending on the criticality of the overrun tasks, the system state is also updated.

Types	V2V Task		Cruise Control (Constant Headway Mode)	
	Budget (Ticks)	Period	Budget (Ticks)	Period
Type 1	8153	60 ms	146534 (ON-OFF)	20 ms
Type 2	NA		154885 (No DSA)	20 ms

Table 6.5: Budget values for different task degradation.

Degradation Schemes	Scenario 1 (Pseudo-Radar or Collision Avoidance Overrun)	Scenario 2 (Steer Control Overrun)	Scenario 3 (Pseudo-Radar (Collision Avoidance and Steer Control Overrun)
Scheme 1	Suspension of V2V task.		
Scheme 2	Type 1 degradation of V2V task.	Type 1 degradation of V2V task and Type 1 degradation of CC task.	
Scheme 3	Type 1 degradation of both V2V and CC task.	Type 2 degradation of CC task.	1. Type 1 degradation of V2V and CC.
Context Aware - MCS			2. Type 1 degradation of V2V and Type 2 degradation of CC tasks. 3. Type 2 degradation of CC task.

Table 6.6: Scenarios and Task Degradation.

After detection and handling of budget overrun are performed, the system is checked for any active mode change requests by checking the value of the $distance$ variable. Note that mode change request to any other mode (constant-velocity mode or constant-distance mode or collision avoidance mode) depends on the distance between the follower and the leader vehicle. Then, the task with the highest priority is chosen for execution.

6.3.8 Testbed Configuration

The automotive testbed considered in this chapter is essentially a co-simulation environment. During co-simulation, timing artefacts such as delays and skews due to asynchronous communication fabric such as TCP/IP play an important in choosing right configuration to run the simulation without any errors. When values of these timing artefacts are high, the co-simulation environment may not be suitable for testing and validating timing properties of safety-critical CPS controllers. To solve this issue, a synchronised execution of plant and controller is considered in [98] for validating and checking real-time properties of a safety critical system.

In Figure 6.9, the sequence of actions need to happen to produce the output by the controller programs running in the ECU from the instant at which the real-time sensor values are fed by TORCS is presented. First, the real-time sensor values are fed by TORCS to the Simulink. These values are then sent to Gateways via serial communication. Gateways in turn convert these sensor values as CAN messages and send them via CAN to the ECU. Then, the ECU runs the controller code and sends the

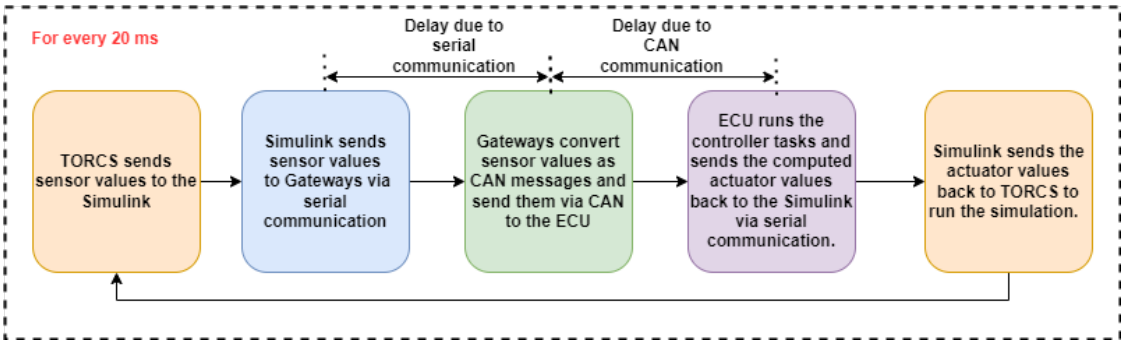

Figure 6.9: Sequence of actions that need to happen for every 20 ms to produce the output by the controller programs running in the ECU from the instant at which the real-time sensor values are fed by TORCS.

computed actuator values back to the Simulink via serial communication. Finally, Simulink sends the actuator values back to TORCS to run the simulation.

After performing several trials, it has been observed that two kinds of delays due to communication between different components in the testbed needs to be accounted. They are the following:

1. Delay induced by the serial communication *i.e.* between Simulink and the gateway and between ECU and the Simulink.

2. Delay due to CAN messages between Gateway and the ECU via CAN network.

After accounting for the above sources of delay involved in the simulation, and after several trials, the testbed has been configured in such a way that sensor values from TORCS are sampled for every 20 ms. This is found to sufficient to produce the required output by controller programs running in the ECU to run the simulation without any issues.

6.4 Evaluation

In the following, experiments and their results to analyse the performance of the CA-MCS and the MM-MCS models are presented. Table 6.5 shows the values of budget and time period of V2V and Adaptive Cruise Control tasks when they are degraded in a specific way. Table 6.6 presents different scenarios and corresponding task degradation.

6.4.1 Degradation Schemes Evaluated for the CA-MCS model

Evaluations are performed for four types of degradation schemes to handle system overload due to budget overrun of a task. Scheme 1 considers the suspension of tasks with relatively lower criticality level [19,47] than the overrun task. Scheme 2 considers a fixed degradation of relatively lower criticality task (reduced budget [26] or increased time period [1]). In Scheme 3, degradation of tasks without considering their criticality parameter but with either reduced budget, increased time period or deadline, etc., is considered [31]. The effect of degradation strategy adopted by the CA-MCS model and the above three schemes are considered for the evaluation of the safety and the performance of the vehicle.

6.4.1.1 Performance and safety parameters monitored

For each degradation scheme, three parameters are monitored. They are: the closest distance maintained between lead and follower vehicles denoted as D_{min}, values of acceleration and heading error of the follower vehicle. A lower value of D_{min} can be extremely dangerous as it may lead to a collision and therefore can be considered as the safety parameter. For a comfortable ride, variation in the values of acceleration and heading error should be gradual and smooth. Higher vehicle velocity in curves is likely to cause more fluctuations in the heading error resulting in increased discomfort to the driver. According to ISO 2631-1 [99], Root Mean Square(RMS) values of the acceleration data can be considered to evaluate vibration exposure to the human body. Hence RMS value of acceleration (A_{RMS}) is considered as the measure of the performance of longitudinal control (performed by ACC and CA) and RMS value of the heading error (H_{RMS}) is considered as a measure of the performance of lateral control (performed by SC).

We consider 3 scenarios to study how different degradation schemes affect the safety and performance parameters. Each scenario differs in the task that overruns its normal budget. With scenarios 1 and 2, the effect of suspending V2V task (lowest criticality) to the safety of the vehicle and the advantage of considering multiple ways of degrading the cruise control task to achieve effective isolation between applications performing longitudinal and lateral control of the vehicle are shown. With scenario 3, a designer has to consider the trade-off between the complexity of degradation models and flexibility to achieve the desired degradation to handle multiple task overruns. The values of parameters under normal conditions when no task has overrun its normal budget are found to be, $D_{min} = 15m$, $A_{RMS} = 4.5ms^{-2}$ and $H_{RMS} = 0.013$ (no units).

6.4.1.2 Effect of suspending a lower criticality task

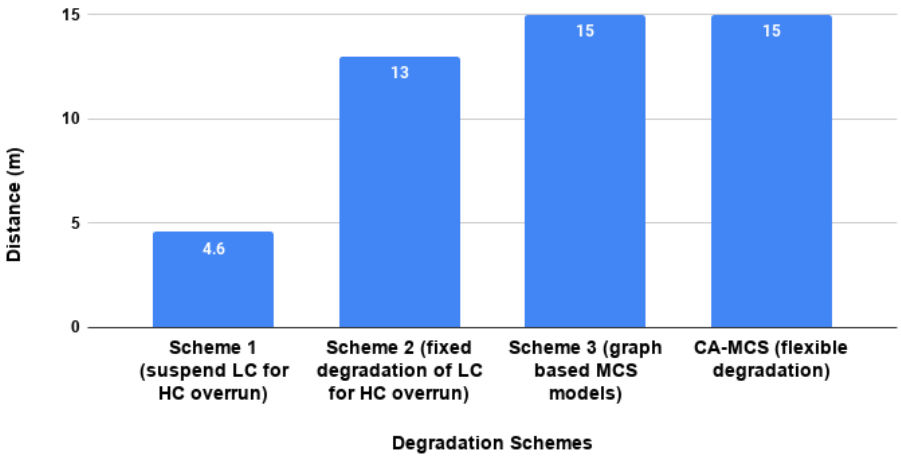

Figure 6.10: Variations in the minimum distance between the follower vehicle and the lead vehicle.

In scenario 1, pseudo-radar or collision avoidance task is considered to overrun its normal budget. Figure 6.10 shows that degradation according to scheme 1 *i.e.* suspension of the V2V task, has led to an extremely low value of D_{min} (around 4.6m) threatening the safety of the vehicle resulting in the transition to the collision avoidance mode. Degradation according to schemes 2, 3 and CA-MCS resulted in a higher value of D_{min} ($\geq$ 13m) and hence can be considered to be safe for this scenario.

6.4.1.3 Benefit of having multiple degradation of a task

Both CA-MCS and scheme 3 consider type 1 degradation of CC task to handle the budget overrun of pseudo-radar or CA task (scenario 1), and type 2 degradation of CC task to handle the budget overrun of SC task (scenario 2). For type 1 degradation of CC, a higher value of A_{RMS} for both scheme 3 and CA-MCS is observed for scenario 1 as shown in Figure 6.11 but only a small change in H_{RMS} is observed as shown in Figure 6.12. Therefore the effect of type 1 degradation of CC to handle budget overrun of pseudo-radar or CA has no effect on the performance of lateral control application, SC. In Figure 6.12, for scenario 3, for scheme 3 and CA-MCS, type 2 degradation of CC task has resulted in a higher value of H_{RMS} than scenario 2, but the value of A_{RMS} is not affected and is nearly equal to the normal state value. This shows that the performance of longitudinal control applications such as CA or ACC has not been affected due to the degradation chosen to handle the overrun of the lateral control application, SC. Clearly, the degradation with context-awareness *i.e.* degradation based on the overrun

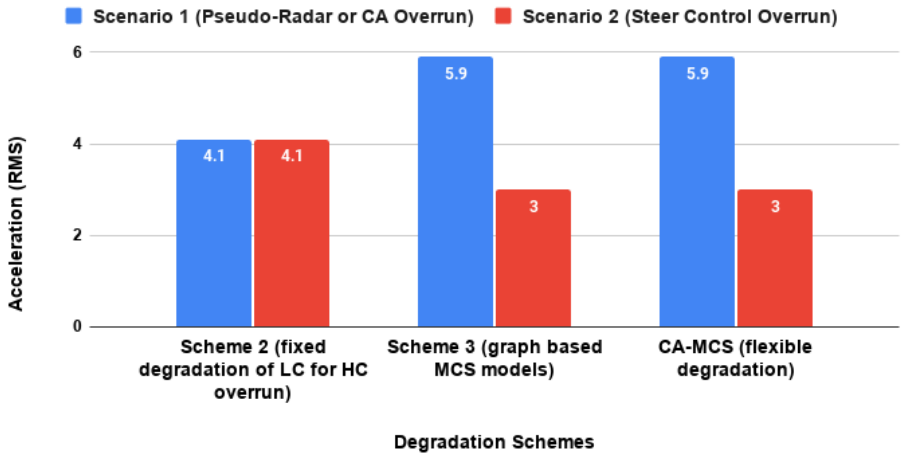

Figure 6.11: Variations in the root mean square (RMS) value of the follower vehicle's acceleration.

task gives the ability to isolate the effects of degradation between applications.

6.4.1.4 Benefit of considering criticality of a task to handle multiple tasks overruns

When multiple tasks overrun their budgets, degradation based on the highest criticality task that has overrun its budget can be useful to ensure performance and safety (for scenarios considered). Scenario 3 considers overrun of pseudo-radar, CA and SC tasks. Effects of 3 different choices of degradation are studied. Choice 1 corresponds to overrun of the pseudo-radar or CA task which is to degrade V2V (reduced time period) and CC task with ON-OFF control (reduced budget). Choice 2 corresponds to the overrun of both pseudo-radar and SC tasks where type 1 degradation of V2V (corresponding to pseudo-radar) and CC tasks by ignoring DSA (corresponding to SC) are considered. Choice 3 corresponds only to the overrun of SC task (higher criticality than pseudo radar) where CC task is degraded by ignoring DSA. Additionally, we consider the suspension of V2V just to see how safety is impacted due to the suspension of tasks (scheme 1). Observations in Figure 6.13 shows that the number of instances where mode transition to collision avoidance mode was active is highest when V2V was suspended while choice 3 led to the lowest number with zero CA mode activation. The number of CA mode activation was higher for choice 1 than for choice 2. This shows that even if there can be a one to one mapping between overrun task and degraded budget, one has to carefully handle system overload due to budget overrun of multiple tasks. CA-MCS handles this efficiently by allowing a designer to choose the degradation corresponding to higher criticality task which in this case is choice 3 degradation

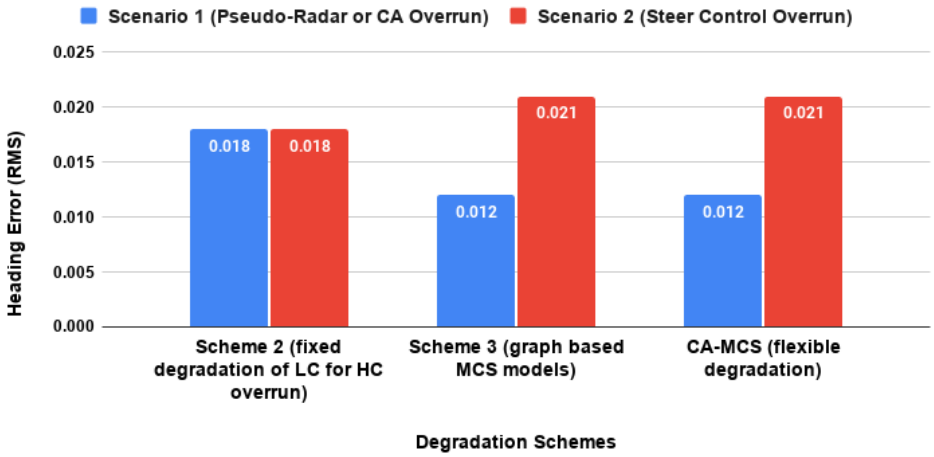

Figure 6.12: Variations in the root mean square (RMS) value of the follower vehicle's heading error.

corresponding to the SC task. Since scheme 3 does not consider criticality level of the task to decide on the task to be degraded, it requires an exponential number of choices to model such a type of degradation. Hence scheme 3 and CA-MCS differ in terms of the complexity of the model to achieve the degradation corresponding to the overrun task with the highest criticality.

6.4.2 Evaluation for the MM-MCS model

Demonstration of the benefit of the MM-MCS model with respect to the performance of the automotive applications running in the testbed was found to be very difficult. As the effects due to failure of a single job execution during mode transition on the performance of the vehicle is practically impossible to observe, experiments were performed only to observe the effect of higher values of offset when the system transition to the collision avoidance (CA) mode was performed. For this, the stopping distance is considered as the performance metric for the evaluation. **Stopping distance** is the total distance travelled from the time instant at which trigger to the collision avoidance mode occurred till the vehicle stops due to the application of the emergency brakes. The values of the offset considered are typically in multiples of time period of the collision avoidance task.

6.4.2.1 Effect of offsets for the collision avoidance task

With null offsets, the vehicle was successfully able to avoid the head-on collision as collision avoidance task executed immediately after the mode transition instant. However, simultaneous budget over-

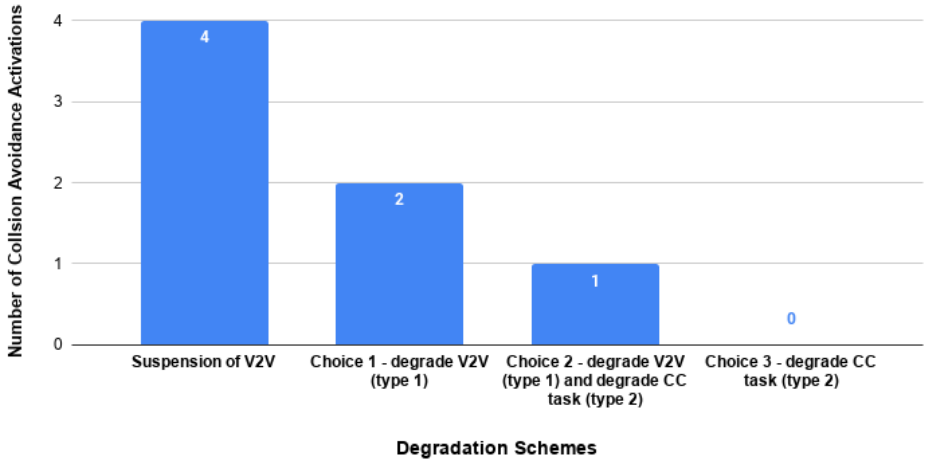

Figure 6.13: Variations in the number of collision avoidance mode activation.

runs of old-mode tasks and new-mode tasks lead to an increase in the system utilization without affecting the execution of tasks. With higher offset values for V2V and pseudo radar tasks, not much change in the values of stopping distance was observed for higher offset values for V2V and pseudo radar as the CA task continued to apply brakes to stop the vehicle when the system transition was triggered to the CA mode. Figure 6.14 shows the variation in the stopping distance for different offset values for the collision avoidance task when the system transition happens to the CA mode at different velocities of the follower vehicle. It is clear that stopping distance increases for higher values of offsets due to the delayed start of the execution of collision avoidance task as soon as the system switches to the CA mode.

6.5 Conclusion

A key aspect of the proposed CA-MCS and MM-MCS models is their ability to achieve graceful degradation in a targeted manner without impacting the safety of the system. As the experiments based on synthetic tasks cannot show the benefit of this aspect, an automotive testbed implemented with applications performing lateral and longitudinal control of the vehicle was built. Experiments were performed by simulating the budget overrun of tasks for different scenarios where each scenario differed from the other in the number of tasks overrun their budgets. These results show that to handle system overload, suspension of lower criticality tasks (as considered by a majority of existing MCS models) can affect the safety of the system whereas, with the CA-MCS and MM-MCS models, one can achieve desired performance degradation in a targeted manner without impacting the system's safety and be

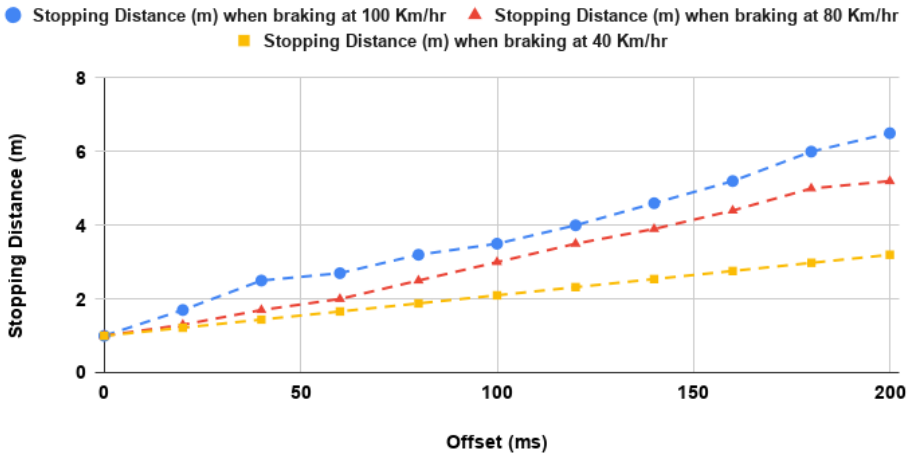

Figure 6.14: Variations in the stopping distance of the follower vehicle.

able to isolate the effects of degradation of applications performing longitudinal and lateral control of the vehicle. The results also show that a higher value of offsets for tasks performing safety-critical functionality can threaten the system's safety. Further, experimental results show that graph-based MCS task models can give a similar level of flexibility in the choice of task degradation but differ from the proposed models only in terms of complexity required in the modelling and timing analysis of the required mixed-criticality behaviour. The source code for the automotive applications and the freertos implementation is made available at https://github.com/vijayaku003/Testbed.git

Chapter 7

Conclusions And Future Work

To accommodate the need to consolidate applications of different criticality onto a common platform (hardware and software), real-time systems have evolved to what is commonly referred to as Mixed Criticality Systems (MCS). A significant number of MCS task models have been motivated to ensure freedom from timing interference between tasks of different criticality. A key property of MCS task models is that the level of pessimism involved in determining the values of parameters, such as the worst-case execution time (WCET) for tasks, become dependent on the criticality level of the task *i.e.*, a higher level of pessimism for higher criticality task. As a result, the platform resources that were provisioned based on the pessimistic WCET estimations are underutilized at runtime.

To improve resource utilization and to ensure freedom from timing interference, MCS task models began to consider multiple execution time estimates for a task estimated at different levels of pessimism. As MCS can also undergo functional mode changes, at run-time, a system overload can occur when any high criticality task violates its service level and/or due to a spike in the resource consumption pattern leading to a temporary system overload when tasks belonging to different modes execute together for a certain time interval when the system switches from one mode to another. To handle the system overload due to budget overrun of a high critical task, a majority of MCS task models consider suspension or degradation of lower criticality tasks which has led to several criticisms. Some real-time task models consider the notion of offset for tasks to postpone new-releases of tasks to handle system overload during mode transitions. However, they do not consider modelling mixed-criticality behaviour of tasks. Some graph-based task models are more expressive and provide flexibility to choose tasks for degradation irrespective of their criticality level and also consider offsets to handle system overload. However, these models require an exponential number of vertices and edges to specify the degraded budget for tasks

for all possible combination of service level violations of tasks and hence such models are harder to analyze. In this chapter, we first present a summary of the main contributions in this thesis and then discuss some possible future research directions.

7.1 Conclusions

The primary focus of this thesis is on developing task models for a mixed-criticality system that can adapt to system overload when service level violation due to budget overrun of tasks and/or functional mode changes happen. To this end, we have introduced two different task models that generalize many classical mixed-criticality system task models and are expressive enough to capture parameters to achieve graceful degradation and offsets both within a mode and across mode transitions, and therefore, can capture the requirements and properties of real-world applications more precisely than the traditional MCS task models. These models are also relatively simple to analyze when compared to existing graph-based MCS task models whose complexity is in the number of vertices and edges which is exponential in the number of tasks. For the proposed task models, we have first established a few properties that are derived by observing existing automotive case studies and from the recommendations of ISO 26262 - a functional safety standard for automotive. The appropriate parameters for the task model are then chosen to satisfy these properties. This is followed by the fixed-priority based schedulability analysis which makes use of parameters to compute the worst-case response time for tasks.

In systems with more than 2 criticality levels, suspension or degradation of tasks with relatively lower criticality than the overloading task may not be acceptable. As pointed out in [24], in a system with 4 criticality levels, to handle system overload due to the budget overrun of any task at criticality level 4 (highest), suspension of tasks with criticality level 3 may lead to catastrophic consequences. To address this issue, in Chapter 4, a new task model referred to as Context-Aware Mixed Criticality System (CA-MCS) model is proposed. The CA-MCS model is also an extension of the classic sporadic task model. The novelty of the CA-MCS model is threefold. First, CA-MCS gives the flexibility to choose tasks that need to be degraded irrespective of their criticality level. This allows a designer to consider the degradation of even higher criticality tasks while keeping relatively lower criticality tasks unaffected. Second, CA-MCS supports multiple ways of degrading a task's budget. This allows a designer to choose a specific way of degrading a task's budget depending on the task(s) that have overrun its budget. The CA-MCS model differs from the graph-based MCS model in the approach to handle budget overruns of multiple tasks. Instead of using criticality to decide on the tasks to be degraded, CA-MCS uses criticality only to choose the degraded budget (among multiple degraded budgets) of the task when

multiple tasks overrun their budgets. When multiple tasks of same criticality l overrun their budgets, for a task that is chosen to be degraded is assigned with the minimum degraded budget among all of the possible degraded budgets that can be allocated to it due to the budget overrun of tasks with criticality l. But, when multiple tasks with different criticality overrun their budgets, only the degraded budget corresponding to the task with the highest criticality among all tasks that have overrun their budgets is allocated. With this approach, CA-MCS can avoid the higher level of complexity in modelling all possible task overrun combinations and their degradations. Experimental results based on the synthetic tasksets show that task degradation can be an effective means to handle system overload within a mode.

To handle system overload during mode transitions, just degradation of tasks may not be sufficient. The existing real-time task models that are designed to capture mode change behaviour do not consider mixed-criticality aspects whereas existing MCS models that provide flexibility in task degradation lack sufficient expressiveness to capture and handle system overload due to both budget overrun and functional mode changes and therefore can easily lead to a higher level of complexity in analysing the timing behaviour of tasks. To address this issue, in Chapter 5 the Multi-Mode Mixed Criticality System (MM-MCS) model is introduced. The novelty of the MM-MCS model is that it combines task degradation and the notion of offsets to handle system overload due to budget overruns and a mode transition. Further, the proposed model establishes precise rules on how to handle budget overrun of tasks both within a mode and across mode transitions. Since it is possible that old-mode tasks active in the new-mode can overrun their normal budgets in the new-mode, offset values for new-mode tasks are computed by considering the maximum possible interference from old-mode tasks that can be active in the new-mode when they execute with their safe budgets. An algorithm to compute the offset values for new-mode tasks based on their criticality value is derived. The algorithm also considers reducing the offsets of higher criticality tasks. This algorithm clearly shows that criticality of tasks can play an important role not only for deciding on the task degradation but also for determining offset values. Experimental results with synthetic tasksets show that task degradation and offsets can be an effective means to handle system overload both within a mode and across mode transitions.

As the models presented in this thesis are motivated by considering practical aspects of an automotive real-time system, it was realized that mere experiments based on synthetic tasks is not enough and it was important to build a realistic testbed to show the real benefits of the proposed models with appropriate performance metrics. To cater this need, in Chapter 6, the details of the experimental platform including its architecture, implementation details of the safety-critical functionalities performing lateral and longitudinal control of the vehicle such as like Adaptive Cruise Control (ACC), Collision Avoidance (CA), Steering Control (SC), Lead Vehicle Detection, exhibiting different criticality levels

are discussed.

7.2 Future Research Directions

The syntax of the proposed CA-MCS and MM-MCS models in this thesis are highly generic and hence can be adopted to any safety critical domain such as avionics, medical electronics etc. However, assumptions regarding the way in which model parameters such as task behaviour, multiple degraded budgets etc., are utilised can be modified based on the domain specific requirements. For example, the notion of criticality of a task is considered even in avionics and medical domains. But, the possibility of multiple degraded budgets may not be common in medical electronics and therefore they can be ignored. In the following subsection, discussions regarding some potential limitations of the proposed models in this thesis and possible ways to mitigate them are considered.

7.2.1 Limitations and potential improvements for the proposed models

The way in which the system's criticality level can be reduced safely without waiting for the idle instant can be considered as the potential future work for the proposed CA-MCS and MM-MCS models. One possible way to achieve this is by keeping track of the number of tasks that has overrun their normal budgets at each criticality level. For example, the semantics of CA-MCS and MM-MCS models can be designed such that while the system executes in a safe state S_l and if no tasks with criticality l has overrun its normal budget for a specific time interval, then the system's safe state can be reduced safely from S_l to S_{l-1}.

Postponing state transitions: In both CA-MCS and MM-MCS models, system transition from one state to another is triggered immediately when the budget overrun of any task occurs. However, it is possible to consider slack based techniques such as the one considered in [51] to monitor the available slack at run time. If sufficient slack is available, transition to the new state can be postponed.

Periodicity of tasks during mode transitions: In the proposed model, to handle mode transition, a synchronous protocol without periodicity is considered *i.e.* both old-mode completed tasks and wholly new tasks are allowed to execute simultaneously and the periodicity of the unchanged tasks are affected during mode transition. Although it seems restrictive, the main advantage of considering mode change protocol without periodicity is that. With appropriate offsets for unchanged tasks, the schedulability of tasksets during mode transition can be improved. As it is possible to abstract influence of unchanged

tasks belonging to old-mode onto the new-mode, there is no need to consider all possible state transition combinations that can occur in both modes simultaneously to compute interference from unchanged tasks. This can drastically reduce the analysis complexity. Otherwise, if a synchronous protocol with periodicity is considered, it is possible to preserve periodicity of the unchanged tasks during mode transition. However, the complexity of the analysis is increased as one has to now consider all possible state and mode transition instances that can occur in both modes together.

Determining system's state during mode transitions: In the MM-MCS model, it is assumed that the system transition to the normal state of the new-mode happens immediately at the instant at which a Mode Change Request (MCR) occurs. The main reason for this assumption is to prevent unnecessary degradation of new-mode tasks to handle system overload due to budget overrun of the old-mode tasks as the functionality of new-mode tasks can be entirely different from that of old-mode tasks. However, to decide the state of the new-mode, many possible ways can be considered. For *e.g.*, one option is to decide on the basis of the behaviour of tasks and their criticality level at t_z *i.e.* old-mode tasks with highest criticality level executing with safe behaviour can found and the system transition to the state corresponding to this criticality level can be done. But in scenarios where the emergency operation (if any) in the new-mode must be completed quickly (a recommendation of ISO 26262), it is useful to assume that system starts directly in any of the safe state or in the normal state of the new-mode.

Handling multiple mode transitions: In the MM-MCS model, we assume that if multiple mode change requests occur at the same time, the system non-deterministically selects one. But, it is possible to consider some rules to handle such scenarios. For example, one can extend the notion of criticality to mode change requests *i.e.* at run-time, only the mode change request with higher criticality can be considered as valid. This can also reduce the number of combinations of simultaneous mode change requests that need to be considered in the offline analysis. Again, this approach still needs to be handled carefully when multiple mode change requests with equal criticality occur.

Limited task pre-emptions: In both CA-MCS and MM-MCS models, it is assumed that the scheduler overheads during task pre-emption such as the computation time required to activate a task, schedule the tasks, and terminate a task are included in the value of budget for tasks. There are approaches discussed in the existing literature [76] to reduce the number of pre-emption of tasks. These are not considered in the proposed models and can be considered as the potential future work.

Resource sharing protocols: The proposed models in this thesis consider tasks to be independent and therefore they do not share any common resource other than the processor. In reality, in safety-critical

domains such as automotive, tasks with different criticality may need to share a common sensor or actuator. Functional safety standards such as ISO 26262 restrict the data flow from low to high criticality tasks unless it is shown that the high criticality is able to manage any data inconsistency caused due to malfunctioning of the low criticality task. Existing protocols such as [100] allow doing this. However, there is a need to formulate resource access control policies and mechanism for the proposed models in this thesis where a task can exhibit both mixed-criticality and functional mode change behaviour. Given the fact that models can support multiple degraded budgets, the techniques to compute the maximum possible blocking time from a task is required which can also lead to a scheduling problem. This can be taken as an important direction.

Extension to other uniprocessor algorithms: The analysis for other scheduling approaches such as, Fixed-Priority Scheduling with Deferred Preemption, Dual Priority Scheduling can be considered for both CA-MCS and MM-MCS models. Future works can be focused in the direction to modify the response time equations for these approaches. Apart from that, other scheduling approaches such as Earliest Deadline First-Virtual Deadline (EDF-VD) [101], slack based scheduling [53] can also be considered as the potential future work.

Support on Multiprocessors: The task models and their corresponding schedulability tests for the proposed models can be to extended to suit multiprocessor platforms. For this, one has to deal with the problems such as partitioning and/or migration of tasks from processors that are experiencing overload to the ones that are executing in the normal state. Solving these problems can offer the possibility of minimizing the number of tasks undergoing degradation or minimizing the offset values. These topics can be very interesting to solve as safety-critical domains such as automotive are starting to adopt multi-core technologies to harness the power and cost benefits [102].

References

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

[2] E. A. Lee and S. Seshia, "Introduction to embedded systems - a cyber-physical systems approach," 2013.

[3] J. A. Stankovic, K. Ramamritham, and M. Spuri, *"Deadline Scheduling for Real-Time Systems: Edf and Related Algorithms"*. USA: Kluwer Academic Publishers, 1998.

[4] G. Bernat, A. Burns, and A. Llamosi, "Weakly hard real-time systems," *IEEE Trans. Comput.*, vol. 50, no. 4, pp. 308–321, Apr. 2001. [Online]. Available: http://dx.doi.org/10.1109/12.919277

[5] M. Becker, N. Khalilzad, R. J. Bril, and T. Nolte, "Extended support for limited preemption fixed priority scheduling for osek/autosar-compliant operating systems," in *10th IEEE International Symposium on Industrial Embedded Systems (SIES)*, 2015, pp. 1–11.

[6] C. Bailey, "Hard real-time operating system kernel. investigation of mode change," *Computer*, 1993.

[7] J. Real and A. Crespo, "Mode change protocols for real-time systems: A survey and a new proposal," *Real-Time Systems*, vol. 26, no. 2, pp. 161–197, Mar 2004. [Online]. Available: https://doi.org/10.1023/B:TIME.0000016129.97430.c6

[8] S. Lui, R. Ragunathan, L. John, and R. Krithi, "Mode change protocols for priority-driven preemptive scheduling," in *The Journal of Real-Time Systems*, 1989.

[9] J. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," *Performance Evaluation*, vol. 2, no. 4, pp. 237–250, Dec. 1982.

[10] R. Bell, "Introduction to IEC 61508," in *Proceedings of the 10th Australian Workshop on Safety Critical Systems and Software - Volume 55*, ser. SCS '05. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2006, pp. 3–12. [Online]. Available: http://dl.acm.org/citation.cfm?id=1151816.1151817

[11] L. A. Johnson, "DO-178B, software considerations in airborne systems and equipment certification," *Crosstalk, October*, 1998.

[12] V. Hilderman and T. Baghi, *"Avionics certification: a complete guide to DO-178 (software), DO-254 (hardware)"*. Avionics Communications, 2007.

[13] P. Sinha, "Architectural design and reliability analysis of a fail-operational brake-by-wire system from ISO 26262 perspectives," *Reliability Engineering & System Safety*, vol. 96, no. 10, pp. 1349–1359, 2011.

[14] "ISO 26262:2011, Road vehicles – functional safety," International Organization for Standardization, Geneva, CH, Standard, Nov. 2011.

[15] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*. IEEE, 2007, pp. 239–243.

[16] "ARINC653 - An Avionics Standard for Safe, Partitioned Systems," Wind River Systems / IEEE Seminar, Standard, Nov. 2008.

[17] H. Li and S. Baruah, "An algorithm for scheduling certifiable mixed-criticality sporadic task systems," in *2010 31st IEEE Real-Time Systems Symposium*, 2010, pp. 183–192.

[18] N. Guan, P. Ekberg, M. Stigge, and W. Yi, "Effective and efficient scheduling of certifiable mixed-criticality sporadic task systems," in *2011 IEEE 32nd Real-Time Systems Symposium*, 2011, pp. 13–23.

[19] S. K. Baruah, A. Burns, and R. I. Davis, "Response-time analysis for mixed criticality systems," in *2011 IEEE 32nd Real-Time Systems Symposium*, 2011, pp. 34–43.

[20] A. Easwaran, "Demand-based scheduling of mixed-criticality sporadic tasks on one processor," in *2013 IEEE 34th Real-Time Systems Symposium*, 2013, pp. 78–87.

[21] A. Mok, "Fundamental design problems of distributed systems for the hard-real-time environment," 08 2005.

[22] S. Baruah and B. Chattopadhyay, "Response-time analysis of mixed criticality systems with pessimistic frequency specification," in *2013 IEEE 19th International Conference on Embedded and Real-Time Computing Systems and Applications*, 2013, pp. 237–246.

[23] A. Burns and S. Baruah, *"Timing Faults and Mixed Criticality Systems"*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 147–166.

[24] A. Esper, G. Nelissen, V. Nélis, and E. Tovar, "An industrial view on the common academic understanding of mixed-criticality systems," *Real-Time Systems*, vol. 54, no. 3, pp. 745–795, Jul 2018.

[25] S. Holzknecht, E. Biebl, and H.-U. Michel, "Graceful degradation for driver assistance systems," *Advanced Microsystems for Automotive Applications 2009, Springer Berlin Heidelberg*, pp. 255–265.

[26] A. Burns and S. Baruah, "Towards a more practical model for mixed criticality systems," in *2013 Workshop on Mixed Criticality (WMC)*, 2013.

[27] A. Mjeda, G. Leen, and E. Walsh, "The autosar standard - the experience of applying simulink according to its requirements," 04 2007.

[28] L. Xiao and F. Gao, "A comprehensive review of the development of adaptive cruise control systems," *Vehicle System Dynamics*, vol. 48, no. 10, pp. 1167–1192, 2010. [Online]. Available: https://doi.org/10.1080/00423110903365910

[29] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The worst-case execution-time problem—overview of methods and survey of tools," vol. 7, no. 3, 2008.

[30] P. Huang, P. Kumar, N. Stoimenov, and L. Thiele, "Interference constraint graph; a new specification for mixed-criticality systems," in *2013 IEEE 18th Conference on Emerging Technologies Factory Automation (ETFA)*, Sept 2013, pp. 1–8.

[31] P. Ekberg and W. Yi, "Schedulability analysis of a graph-based task model for mixed-criticality systems," *Real-Time Systems*, vol. 52, pp. 1–37, 2015.

[32] M. Stigge, P. Ekberg, N. Guan, and W. Yi, "The digraph real-time task model," in *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2011, pp. 71–80.

[33] J. Real, "Mode change protocols for real-time systems," Ph.D. dissertation, University of York, 2000.

[34] M. Joseph and P. Pandya, "Finding Response Times in a Real-Time System," *The Computer Journal*, vol. 29, no. 5, pp. 390–395, 01 1986. [Online]. Available: https://doi.org/10.1093/comjnl/29.5.390

[35] K. Tindell and A. Alonso, "A very simple protocol for mode changes in priority preemptive systems," *Computer*, 1996.

[36] G. C. Buttazzo, G. Lipari, and L. Abeni, "Elastic task model for adaptive rate control," in *Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No.98CB36279)*, Dec 1998, pp. 286–295.

[37] N. Audsley, "Optimal Priority Assignment and Feasibility of Static Priority Tasks with Arbitrary Start Times," in *The University of York Technical Report*, 1991.

[38] S. Baruah and S. Vestal, "Schedulability analysis of sporadic tasks with multiple criticality specifications," in *2008 Euromicro Conference on Real-Time Systems*, July 2008, pp. 147–155.

[39] S. Baruah, V. Bonifaci, G. DAngelo, H. Li, A. Marchetti-Spaccamela, S. van der Ster, and L. Stougie, "The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems," in *2012 24th ECRTS*, 2012.

[40] S. Baruah, H. Li, and L. Stougie, "Towards the design of certifiable mixed-criticality systems," in *2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2010, pp. 13–22.

[41] R. M. Pathan, "Fault-tolerant and real-time scheduling for mixed-criticality systems," *Real-Time Systems*, vol. 50, Jul 2014.

[42] M. Jan, L. Zaourar, and M. Pitel, "Maximizing the execution rate of low-criticality tasks in mixed criticality system," in *WMC*, 2013.

[43] P. Huang, G. Giannopoulou, N. Stoimenov, and L. Thiele, "Service adaptions for mixed-criticality systems," in *in In Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2014.

[44] S. Baruah, A. Burns, and Z. Guo, "Scheduling mixed-criticality systems to guarantee some service under all non-erroneous behaviors," in *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)*, July 2016, pp. 131–138.

[45] O. Gettings, S. Quinton, and R. I. Davis, "Mixed criticality systems with weakly-hard constraints," in *Proceedings of the 23rd International Conference on Real Time and Networks Systems*, ser. RTNS '15. New York, NY, USA: ACM, 2015, pp. 237–246. [Online]. Available: http://doi.acm.org/10.1145/2834848.2834850

[46] J. Y. Chung, J. W. S. Liu, and K. J. Lin, "Scheduling periodic jobs that allow imprecise results," *IEEE Transactions on Computers*, vol. 39, no. 9, pp. 1156–1174, Sep 1990.

[47] X. Gu, A. Easwaran, K. M. Phan, and I. Shin, "Resource efficient isolation mechanisms in mixed-criticality scheduling," in *2015 27th Euromicro Conference on Real-Time Systems*, July 2015, pp. 13–24.

[48] T. Fleming and A. Burns, "Incorporating the notion of importance into mixed criticality systems," in *Workshop on Mixed- Criticality Systems (co-located with RTSS)*, July 2014.

[49] J. Ren and L. T. Xuan Phan, "Mixed-criticality scheduling on multiprocessors using task grouping," in *2015 27th Euromicro Conference on Real-Time Systems*, 2015, pp. 25–34.

[50] R. Ernst and M. D. Natale, "Mixed criticality systems;a history of misconceptions?" *IEEE Design Test*, vol. 33, no. 5, pp. 65–74, Oct 2016.

[51] X. Gu and A. Easwaran, "Dynamic budget management with service guarantees for mixed-criticality systems," in *2016 IEEE Real-Time Systems Symposium (RTSS)*, Nov 2016, pp. 47–56.

[52] F. G. L. P. L. P. H. Fayyad-Kazan and M. Timmerman, "A design that incorporates adaptive reservation into mixed-criticality systems," in *Scientific Programming, vol. 2017*, 2017, p. 20.

[53] K. Lakshmanan, D. d. Niz, and R. Rajkumar, "Mixed-criticality task synchronization in zero-slack scheduling," in *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2011, pp. 47–56.

[54] D. de Niz, K. Lakshmanan, and R. Rajkumar, "On the scheduling of mixed-criticality real-time task sets," in *2009 30th IEEE Real-Time Systems Symposium*, 2009, pp. 291–300.

[55] H. Chwa, H. Baek, and J. Lee, "Physical-state-aware dynamic slack management for mixed-criticality systems," 04 2018, pp. 129–139.

[56] I. Bate, A. Burns, and R. I. Davis, "A bailout protocol for mixed criticality systems," in *2015 27th Euromicro Conference on Real-Time Systems*, 2015, pp. 259–268.

[57] D. de Niz and L. T. X. Phan, "Partitioned scheduling of multi-modal mixed-criticality real-time systems on multiprocessor platforms," in *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2014, pp. 111–122.

[58] L. Phan, I. Lee, and O. Sokolsky, "Compositional analysis of multi-mode systems," *Departmental Papers (CIS)*, 07 2010.

[59] M. Negrean, M. Neukirchner, S. Stein, S. Schliecker, and R. Ernst, "Bounding mode change transition latencies for multi-mode real-time distributed applications," in *ETFA2011*, 2011, pp. 1–10.

[60] P. Rattanatamrong and J. B. Fortes, "Mode transition for online scheduling of adaptive real-time systems on multiprocessors," in *2011 IEEE 17th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2011)*, vol. 1. Los Alamitos, CA, USA: IEEE Computer Society, aug 2011, pp. 25–32. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/RTCSA.2011.71

[61] V. Nelis and J. Goossens, "Mode change protocol for multi-mode real-time systems upon identical multiprocessors," 11 2008.

[62] V. Nelis, J. Goossens, and B. Andersson, "Two protocols for scheduling multi-mode real-time systems upon identical multiprocessor platforms," 07 2009, pp. 151–160.

[63] P. Pedro, "Schedulability of mode changes in flexible real-time distributed systems," Ph.D. dissertation, University of York, 1999.

[64] K. W. Tindell, A. Burns, and A. J. Wellings, "Mode changes in priority preemptively scheduled systems," in *[1992] Proceedings Real-Time Systems Symposium*, Dec 1992, pp. 100–109.

[65] E. Yip, M. M. Y. Kuo, P. S. Roop, and D. Broman, "Relaxing the synchronous approach for mixed-criticality systems," in *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2014, pp. 89–100.

[66] L. Phan, I. Lee, and O. Sokolsky, "A semantic framework for mode change protocols," 04 2011, pp. 91–100.

[67] V. K. Sundar and A. Easwaran, "A practical degradation model for mixed-criticality systems," in *2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC)*, 2019, pp. 171–180.

[68] E. Alexandre, Nelissen, Geoffrey, Nelis, Vincent, Tovar, and Eduardo, "How realistic is the mixed-criticality real-time system model?" in *Proceedings of the 23rd International Conference on Real Time and Networks Systems*, ser. RTNS '15. New York, NY, USA: ACM, 2015, pp. 139–148. [Online]. Available: http://doi.acm.org/10.1145/2834848.2834869

[69] H. Chu, Y. Chen, L. Guo, B. Gao, and H. Chen, "Application of slope sensor in hill-start to amt (automated manual transmission) vehicles," in *SAE Technical Paper*. SAE International, 04 2015. [Online]. Available: https://doi.org/10.4271/2015-01-1108

[70] C. Becker, D. Arthur, and J. Brewer, "Functional safety assessment of a generic, conventional, hydraulic braking system with antilock brakes, traction control, and electronic stability control," 2018.

[71] "Deliverable D2.1: Definition of Use Cases and Scenarios for Safe Adaptation," 2014. [Online]. Available: https://www.safeadapt.eu/en/deliverables.html

[72] "Deliverable D1.4: Impact study of the interference with respect to asil," 2011. [Online]. Available: http://cordis.europa.eu/

[73] A. Burns, "An augmented model for mixed criticality," in *Mixed Criticality on Multicore/Manycore Platforms (Dagstuhl Seminjournalar 15121)*, 2015, vol. 5, no. 3, pp. 92–93.

[74] I. Wenzel, R. Kirner, B. Rieder, and P. Puschner, "Measurement-based timing analysis," vol. 17, 10 2008, pp. 430–444.

[75] R. Kirner and P. Puschner, "Classification of wcet analysis techniques," vol. 2005, 06 2005, pp. 190 – 199.

[76] G. C. Buttazzo, M. Bertogna, and G. Yao, "Limited preemptive scheduling for real-time systems. a survey," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 1, pp. 3–15, 2013.

[77] R. I. Davis and A. Burns, "Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems," *Real-Time Systems*, vol. 47, no. 1, pp. 1–40, Jan 2011.

[78] B. Enrico and B. Giorgio, "Measuring the performance of schedulability tests," in *Real-Time Syst 30, 129–154 (2005)*, 2005, pp. 129–154.

[79] P. Emberson, R. Stafford, and R. Davis, "Techniques for the synthesis of multiprocessor tasksets," in *WATERS workshop at the Euromicro Conference on Real-Time Systems*, Jul. 2010, pp. 6–11, 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems ; Conference date: 06-07-2010.

[80] J. Lee, H. S. Chwa, L. T. X. Phan, I. Shin, and I. Lee, "Mc-adapt: Adaptive task dropping in mixed-criticality scheduling," *ACM Trans. Embed. Comput. Syst.*, vol. 16, no. 5s, sep 2017. [Online]. Available: https://doi.org.remotexs.ntu.edu.sg/10.1145/3126498

[81] A. Bastoni, B. Brandenburg, and J.Anderson, "Cache-related preemption and migration delays: Empirical approximation and impact on schedulability." in *In Proceedings of Sixth International Workshop on Operating Systems Platforms for Embedded Real-Time Applications*, 2010, pp. 33–34.

[82] A. Burns and R. I. Davis, "A survey of research into mixed criticality systems," *ACM Comput. Surv.*, vol. 50, no. 6, Nov. 2017. [Online]. Available: https://doi.org/10.1145/3131347

[83] J. Jansson, "Collision avoidance theory with application to automotive collision mitigation," *Doctoral Dissertation, Linkoping University, Sweden, 2005*.

[84] V. K. Sundar and A. Easwaran, "Testbed for practical considerations in mixed-criticality system design," in *2019 Proceedings of Brief Presentations IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2019, pp. 29–30.

[85] M. Di Natale and A. L. Sangiovanni-Vincentelli, "Moving from federated to integrated architectures in automotive: The role of standards, methods and tools," *Proceedings of the IEEE*, vol. 98, no. 4, pp. 603–620, 2010.

[86] M. D. Natale, H. Zeng, P. Giusto, and A. Ghosal, *"Understanding and Using the Controller Area Network Communication Protocol: Theory and Practice"*. Springer Publishing Company, Incorporated, 2012.

[87] D. George, "A Smart Way to Drive ECU Consolidation," Tech. Rep.

[88] D. U, W. Z, P. Y, and M. R, "Autoplug: An automotive test-bed for electronic controller unit testing and verification," *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*.

[89] B. Wymann, C. Dimitrakakisy, A. Sumnery, and C. Guionneauz, "Torcs: The open racing car simulator," 2015.

[90] "Frdm-kl25z: Freescale freedom development platform for kinetis mcus." [Online]. Available: https://www.nxp.com/products

[91] "Hercules tms570ls31x/21x development kit." [Online]. Available: http://www.ti.com/tool/TMDS570LS31HDK

[92] M. Back, *"FreeRTOS: A Practical Approach with Arduino"*. Independently published, 2018.

[93] https://github.com/VerifiableAutonomy/TORCSLink.

[94] O. Rosas-Jaimes, L. A. Quezada-Téllez, and G. Fernandez-Anaya, "Control designs and stability analyses for helly's car-following model," *International Journal of Modern Physics C*, vol. 29, 03 2018.

[95] C. G. Serna and Y. Ruichek, "Dynamic speed adaptation for path tracking based on curvature information and speed limits," *Sensors (Basel, Switzerland)*, vol. 17, 2017.

[96] F. Bella and R. Russo, "A collision warning system for rear-end collision: a driving simulator study," *Procedia - Social and Behavioral Sciences*, vol. 20, pp. 676 – 686, 2011.

[97] B. Sultan and M. McDonald, "Assessing the safety benefit of automatic collision avoidance systems (during emergency braking situations)," 2003.

[98] S. Sood, A. Malik, and P. Roop, "Robust design and validation of cyber-physical systems," *ACM Transactions on Embedded Computing Systems*, vol. 18, pp. 1–21, 11 2019.

[99] "ISO 2631-5:2018 Mechanical vibration and shock — Evaluation of human exposure to whole-body vibration," International Organization for Standardization, Geneva, CH, Standard, Nov. 1997.

[100] K. Biba, *Integrity Considerations for Secure Computer Systems*. United States Air Force, Electronic Systems Division: Defense Technical Information Center, 1977.

[101] D. Liu, J. Spasic, N. Guan, G. Chen, S. Liu, T. Stefanov, and W. Yi, "Edf-vd scheduling of mixed-criticality systems with degraded quality guarantees," in *2016 IEEE Real-Time Systems Symposium (RTSS)*, 2016, pp. 35–46.

[102] P. Leteinturier, S. Brewerton, and K. Scheibert, "Multicore benefits challenges for automotive applications," in *SAE Technical Paper*. SAE International, 04 2008. [Online]. Available: https://doi.org/10.4271/2008-01-0989