université de bretagne occidentale

THESIS / UNIVERSITÉ DE BRETAGNE OCCIDENTALE

under the seal of the Université Européenne de Bretagne

for the title of

DOCTOR of the UNIVERSITÉ DE BRETAGNE OCCIDENTALE

Distinction: Science and Technology of Information and Communication

École Doctorale Santé, Information, Communication, Mathématique, Matière

presented by

Shuai Li

Prepared at *Laboratoire des Sciences et Techniques de l'Information, de la Communication et de la Connaissance* and *Thales Communications & Security*

Scheduling Analysis of Tasks Constrained by Time-Division Multiplexing: Application to Software Radio Protocols

Thesis defended on Novembre 27th 2014

In front of the following committee:

Shuai Li: *Scheduling Analysis of Tasks Constrained by Time-Division Multiplexing: Application to Software Radio Protocols*

ACKNOWLEDGMENTS

I present my thanks to Frédéric Boniol and Maryline Chetto for taking their time to review my dissertation. It was an honor to have you as referees. I also thank Éric Gressier-Soudan and José Rufino for honoring me by accepting to be examiners of my dissertation.

I would like to thank my doctoral director Frank Singhoff for his outstanding commitment to this work. I thank him for always being available and helpful. His knowledge, insight, and advices improved immensely the quality of the scientific and technical contributions of this work, but also its presentation. I thank him for his guidance, which helped me gain invaluable skills as a researcher. I would also like to thank Michel Bourdellès and Stéphane Rubini for supervising my doctoral thesis. I thank them for the time they took to evaluate my contributions, the problems they noticed, and the discussions we had, which helped me progress in my work and results.

The opportunity to work on this doctoral project was offered by Thales Communications & Security, the Lab-STICC laboratory, and the Université de Bretagne Occidentale. I would especially like to thank Yves-André Caparros, Marc Isard, and Pierre Verrecchia, from Thales Communications & Security, for allowing me to participate in their research projects and base my contributions on problems encountered in real case-studies developed at Thales.

My thanks also go to all of my colleagues. Not only were our technical discussions very interesting, but it was wonderful to work with you everyday. In particular, I would like thank Brigitte D., Bruno C., Christian F., Claire G., Émilie O., Grégory G., Jean-Philippe B., Josselyn K., Julien M., Laurent P., Martin B., Méziane O., Mounir L., Thierry C., Véronique K., Vincent G., Xavier R.

Last, but not least, I would like to thank my family, my friends, and my special someone, who helped me with the review of this dissertation and supported me during these years.

Abstract

The work, presented in this thesis, aims at performing automatically scheduling analysis of Time Division Multiple Access (TDMA) based communication systems. Products called software radio protocols, developed at Thales Communications & Security, are examples of such systems.

TDMA is a channel access method based on the division of time into several time slots. TDMA-based software radio protocols are real-time embedded systems. They are implemented by tasks that are statically allocated on multiple processors. A task may have an execution time, a deadline, and a release time that depend on TDMA. The tasks also have dependencies through precedence and shared resources.

TDMA-based software radio protocols have architecture characteristics that are not handled by scheduling analysis methods of the literature. A consequence is that existing methods give either optimistic or pessimistic analysis results. Furthermore, existing architecture models at Thales do not contain enough information to be used for scheduling analysis. The information is only available in specification documents. These issues impact the possibility to perform scheduling analysis, but also the possibility to perform it automatically.

The propositions of this thesis solve these problems. An experimental architecture model is proposed in the UML MARTE modeling language. The architecture model is transformed to a task model called Dependent General Multiframe (DGMF). The DGMF task model describes, in particular, the different jobs of a task, and task dependencies. To analyze DGMF tasks, they are transformed to another model called tree-shaped transaction. Transactions are precedence dependent tasks. Transactions that result from the transformation have non-immediate tasks. These tasks are not necessarily released immediately by their predecessor task. To consider the effects of non-immediateness, this thesis proposes the WCDOPS+NIM schedulability test for tree-shaped transactions. The general analysis method is implemented as a toolchain that can be used by engineers at Thales.

Experimental results show that the propositions give less pessimistic schedulability results, compared to fundamental methods. The results are less pessimistic for both randomly generated systems and real case-studies from Thales. Furthermore, experiments show that scheduling analysis can be applied automatically to a TDMA-based software radio protocol.

Less pessimistic results are important for engineering work, in order to limit the over-dimensioning of resources. The automatic analysis is a gain in productivity. These are advantages for engineers in the more and more competitive market of software radios.

Résumé

Le travail présenté dans cette thèse vise à analyser automatiquement l’ordonnancement de systèmes de communications basés sur TDMA. Des produits développés chez Thales Communications & Security, appelés protocoles radio logicielle, sont des exemples de tels systèmes.

TDMA est une méthode d’accès au canal basée sur la division du temps en slot temporel. Les protocoles radio logicielle basés sur TDMA sont des systèmes temps-réel embarqués. Ils sont implémentés avec des tâches allouées statiquement sur des processeurs. Une tâche peut avoir un temps d’exécution, une échéance, et un temps d’activation qui dépendent de TDMA. Les tâches sont dépendantes par précédence et ressource partagée.

Les protocoles radio logicielle basé sur TDMA ont des caractéristiques d’architecture qui ne sont pas supportés par les méthodes d’analyse de la littérature. Elles donnent donc des résultats d’analyse optimistes ou pessimistes. De plus, les modèles d’architecture à Thales ne contiennent pas assez d’informations pour être utilisés pour l’analyse. Ces informations ne sont disponibles que dans des documents de spécification. Ces problèmes impactent la possibilité d’appliquer l’analyse mais aussi de l’appliquer automatiquement.

Les propositions de cette thèse règlent ces problèmes. Un modèle d’architecture expérimental est proposé en UML MARTE. Le modèle d’architecture est transformé au modèle de tâche Dependent General Multiframe (DGMF). DGMF décrit, en particulier, les activations d’une tâche et ses dépendances. Pour analyser les tâches DGMF, elles sont transformées en un autre modèle appelé transaction arborescente. Les transactions sont des tâches contraintes par précédence. Les transactions issues de la transformation ont des tâches non-immédiates. Une telle tâche n’est pas nécessairement activée immédiatement par son prédécesseur. Pour prendre en compte l’effet de la non-immédiateté, cette thèse propose le test d’ordonnançabilité WCDOPS+NIM pour transaction arborescente. La méthode d’analyse générale est implémentée comme chaîne d’outils dédiée aux ingénieurs chez Thales.

Des expériences montrent que les propositions donnent des résultats d’ordonnançabilité moins pessimistes, comparés aux méthodes fondamentales. Les résultats sont moins pessimistes pour des systèmes générés aléatoirement et des vrais cas d’étude chez Thales. L’analyse peut aussi être appliquée automatiquement à un protocole radio logicielle basé sur TDMA.

Des résultats moins pessimistes permettent de limiter le surdimensionnement des ressources. L’analyse automatique est un gain de productivité. Ce sont des avantages pour les ingénieurs dans un marché de la radio logicielle de plus en plus compétitif.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACRONYMS

AADL	Architecture Analysis & Description Language
ADL	Architecture Description Language
ASIC	Application-Specific Integrated Circuit
AUTOSAR	AUTomotive Open System ARchitecture
BCET	Best Case Execution Time
BCRT	Best Case Response Time
CAN	Controller Area Network
CBM	Component-Based Model
CCM	CORBA Component Model
CORBA	Common Object Request Broker Architecture
DAG	Directed Acyclic Graph
DGMF	Dependent General Multiframe
DM	Deadline-Monotonic
DP	Dynamic Priority
DPCP	Dynamic Priority Ceiling Protocol
DRT	Digraph Real-time Task
DSP	Digital Signal Processor
DVFS	Dynamic Voltage and Frequency Scaling
EDF	Earliest Deadline First
EMF	Eclipse Modeling Framework
FP	Fixed Priority
FPGA	Field-Programmable Gate Array
FPS	Frames Per Second
GCM	Generic Component Model

GMF	General Multiframe
GPP	General Purpose Processor
GQAM	General Quantitative Analysis Model
GRM	General Resource Model
HDRN	Highly Dynamic Radio Network
HLAM	High Level Application Model
HRM	Hardware Resource Model
IP	Internet Packet
IPCS	Internet Packet Convergence Sub-layer
IT	Information Technology
LLF	Least Laxity First
MAC	Media Access Controller
MANET	Mobile ad-hoc wireless NETwork
MARTE	Modeling and Analysis of Real-Time Embedded systems
MDE	Model-Driven Engineering
MYCCM	Make your CORBA Component Model
OMG	Object Management Group
OS	Operating System
OSI	Open Systems Interconnection
PAM	Performance Analysis Model
PCP	Priority Ceiling Protocol
PHY	PHYsical
PIP	Priority Inheritance Protocol
POSIX	Portable Operating System Interface
QOS	Quality of Service
RLC	Radio Link Control
RM	Rate-Monotonic
RRT	Recurring Real-time Task
RSA	Rational Software Architect
RSN	Radio Sub-Network
RTA	Response Time Analysis
RTES	Real-Time Embedded System
S2C	Service to Cheddar
SAM	Scheduling Analysis Model
SRM	Software Resource Model
SRP	Stack Resource Policy
TDMA	Time Division Multiple Access
UML	Unified Modeling Language
VSL	Value Specification Language
WCBT	Worst Case Blocking Time
WCET	Worst Case Execution Time
WCRT	Worst Case Response Time
XML	Exensible Markup Language

PUBLICATIONS

Some ideas and figures of this thesis have appeared previously in the following publications. In the conclusion of each chapter of this thesis, it will be reminded which publications contain the contributions of the chapter.

BOOK CHAPTER

[26] M. Bourdellès, S. Li, I. Quadri, E. Brosse, E. Gaudin, F. Mallet, A. Goknil, A. Sadovykh, D. George, and J. Kreku, "Fostering analysis from industrial embedded systems modeling," in *Industry and Research Perspectives on Embedded System Design*, IGI Global, 2014.

JOURNAL

[82] S. Li, F. Singhoff, S. Rubini, and M. Bourdellès, "Applicability of real-time schedulability analysis on a software radio protocol," *ACM SIGAda Ada Letters*, vol. 32, no. 3, pp. 81-94, Dec. 2012.

CONFERENCE

[84] S. Li, F. Singhoff, S. Rubini, and M. Bourdellès, "Scheduling Analysis of TDMA-Constrained Tasks: Illustration with Software Radio Protocols," in *Proceedings of the 11th IEEE International Conference on Embedded Software and Systems (ICESS'14)*, Paris, France, 2014. (Acceptance rate = 24.7%; Best paper award)

[83] S. Li, F. Singhoff, S. Rubini, and M. Bourdellès, "Extending Schedulability Tests of Tree-Shaped Transactions for TDMA Radio Protocols," in *Proceedings of the 19th IEEE International Conference on Emerging Technology & Factory Automation (ETFA'14)*, Barcelona, Spain, 2014.

[81] S. Li, S. Rubini, F. Singhoff, and M. Bourdellès, "A Task Model for TDMA Communications," in *Proceedings of the 9th IEEE International Symposium on Industrial Embedded Systems Work in Progress (SIES'14)*, Pisa, Italy, 2014.

WORKSHOP

[80] S. Li, S. Rubini, F. Singhoff, and M. Bourdellès, "Applying Holistic Schedulability Tests to Industrial Systems: Experience and Lessons Learned," in *Proceedings of the 5th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS'14)*, Madrid, Spain, 2014.

[27] M. Bourdellès and S. Li, "PRESTO project, automatic code instrumentation from (Modelio based) models to process performance analysis at the software integration phase," presented at *FITTEST'12*, Paris, France, 2013.

[78] S. Li, M. Bourdellès, A. Acebedo, J. Botella, and F. Peureux, "Experiment on Using Model-Based Testing for Automatic Tests Generation on a Software Radio Protocol," in *Proceedings of the 9th International Workshop on Systems Testing and Validation (STV'12)*, Paris, France, 2012.

[77] S. Li, "PRESTO: Results from execution trace analysis," in *Joint Proceedings of Co-located Events at the 8th European Conference on Modelling Foundations and Applications (ECMFA'12)*, Copenhagen, Denmark, 2012.

[113] I. Quadri and S. Li, "PRESTO: Improvements of Industrial Real-time Embedded Systems Design and Development," in *OMG Workshop on Real-Time*, Embedded and Entreprise-Scale Time-Critical Systems, Paris, France, 2012.

INTRODUCTION

CONTEXT

Real-Time Embedded Systems (RTES) [74] are part of our daily life today. Their usage ranges from specific industrial equipment, like factory robots, to general civilian devices, like smartphones and tablets. A RTES is both an embedded and a real-time system.

An embedded system is a computing system composed of hardware and software, with a specific mission to accomplish, using limited resources. Examples of resources are energy and processors. Embedded systems may interact with the environment in which they execute. The environment may be the physical components of a vehicle to the electromagnetic waves that surround us. The environment in which the embedded system executes, evolves within time. As such, an embedded system is usually subject to real-time computing.

A real-time system is a computing system composed of hardware and software, where computing is subject to time constraints. Therefore the correctness of a real-time system's computing is not only judged by the correctness of the produced data, i.e. functional correctness, but also the time at which the data is produced, i.e. temporal correctness.

Nowadays the software of a RTES is usually multi-tasked. It is composed of several tasks that are units of execution, sharing concurrently the limited processors. The system may be uniprocessor or multiprocessor.

Since the tasks are part of a real-time system, they have time constraints, expressed as deadlines. Once a task is released, we say a job of the task is released, and the job has to complete execution before some deadline.

A task does not execute alone, and several tasks share processors. Scheduling [85] is the method that decides which task has access to the processors during execution and we say that the tasks are scheduled on the processors by a scheduling policy.

Scheduling decisions are taken by considering the priority of a task, which indicates its importance. Priorities may be static or dynamically updated during execution. For multiprocessor systems, scheduling also determines whether tasks may or may not execute on any processor. When tasks can execute on any processor, the scheduling is global. When a task is allocated on a processor and can only execute on that processor, the scheduling is partitioned.

Scheduling analysis [126] is the method used to verify that tasks will meet their deadline, when they are scheduled on the processors. Scheduling analysis can potentially be complex due to the interactions between the tasks. These interactions come in the form of synchronization through software shared resources, which are software entities accessed mutual-exclusively by tasks. The interactions may also come in the form of precedence dependency between tasks, which means the release of a task depends on the completion of a preceding task. Precedence dependencies may be due to communications between tasks.

Some methods, part of scheduling analysis, are schedulability tests. Such tests assesses if all jobs of all tasks will meet their deadlines during execution, when the tasks are scheduled on processors by a specific scheduling policy. Schedulability tests may compute the response time of tasks. The response time is the time between the release of the task and its completion. The response times are compared to deadlines to assess schedulability.

Schedulability tests are associated with task models that abstract the architecture of a RTES for the analysis. There are several task models in the literature that consider more or less interactions between tasks, and characteristics of the execution environment that hosts the tasks.

Scheduling analysis is necessary for the development of some Real-Time Embedded System (RTES). For example, certain radio equipments are typical RTES. This kind of system receives radio electromagnetic waves, maintains a link with other radio stations in the network, and communicates with them. When radio stations communicate, some radio protocol needs to be defined and implemented. Radio protocols may be impacted by the method to access the shared communication medium between several radio stations.

One common method, implemented by radio protocols to access the shared medium, is Time Division Multiple Access (TDMA) [30]. In TDMA, time is divided into several time slots and at each time slot, a radio either transmits its data, or receives some data. TDMA is one reason, among others, that some activities of a radio protocol are divided into time slots. Therefore there are time constraints in systems that use the TDMA method.

Traditionally radios were implemented as dedicated hardware components. Trends in the last two decades have seen the emergence of software radios [99], where more and more components are implemented as software. This is the case for the radio protocol, which is then called a software radio protocol.

In software radio protocols, activities are implemented by software tasks, that are time-constrained. The time constraints must be considered in the development of such a system. Thus scheduling analysis needs to be applied, during the development of a software radio protocol.

Software radios are one of the main products developed at Thales Communications & Security (TCS). There is a will at TCS to integrate scheduling analysis in the development cycle of its products, especially for software radio protocols.

During the development cycle, the system architecture is modeled and the architecture models are exploited either for code generation, or for documentation. Models are made using languages such as Make your CORBA Component Model (MyCCM) [25]. MyCCM is a component-based model developed at Thales. It is dedicated to code generation. Thales also participates in the standardization Modeling and Analysis of Real-Time Embedded systems (MARTE) [42], a profile for the Unified Modeling Language (UML) that extends it for the design and analysis of RTES.

This thesis is on scheduling analysis of software radio protocols. It has been done in collaboration between Thales Communications & Security and the Université de Bretagne Occidentale.

OBJECTIVE AND PROBLEMS

The objective of this thesis is to analyze the schedulability of a software radio protocol that uses TDMA. The analysis method must be automatic, to be integrated into the development cycle.

As we will see, current task models and schedulability tests are not applicable to a software radio protocol that uses a time-division multiplexing method to access the shared medium. Therefore a task model and a schedulability test must be proposed for such a system, and they must be adapted for the characteristics of the system.

In order to automate the analysis, there are two issues to solve. We will see that task models are complex to be used directly by engineers. Task models and their analysis methods must thus be implemented in a tool. But currently not all task models and their analysis methods are implemented in available tools.

Another issue to solve, in order to automate the analysis, is the applicability of an Architecture Description Language (ADL) for scheduling analysis. Indeed, the tasks set for the analysis must be produced automatically from architecture models described with an ADL, used by engineers. We will see that scheduling analysis has not been performed automatically with an ADL specific to the software radio protocol domain.

OVERVIEW OF THE SOLUTION

To solve the issue of task models, a new task model and its analysis method are proposed in this thesis. This work is based on characteristics of real software radio protocols developed at Thales. The task model is an extension of an existing one in the literature. The choice of the task model to extend is made by experiments. The new task model is applicable to the characteristics of a software radio protocol.

In the work of this thesis, the proposed task model and its analysis method are implemented in Cheddar, an open-source scheduling analysis tool. We will see how the the tool is extended.

A model conforming to the task model is then produced automatically from an architecture model of the system. Since scheduling analysis has not been performed with an ADL specific to the software radio protocol domain, this thesis proposes to experiment with UML MARTE, a generic modeling language for the RTES domain. MARTE is chosen because there exists a transformation of its architecture models to development models used by engineers at Thales Communications & Security.

CONTRIBUTIONS

The solution proposed in this thesis is the result of works that propose several contributions.

Extension of the General Multiframe Task Model

This thesis proposes the Dependent General Multiframe (DGMF) task model, an extension of the General Multiframe (GMF) task model [14]. A GMF task is a vector of frames that represent the jobs of the task. The jobs may not have the same parameters, such as deadline and execution time. It is thus possible in the GMF task model to represent individual job parameters of a task.

On the other hand, precedence dependencies and shared resources cannot be modeled in GMF, and thus they are not considered by the schedulability tests for GMF. Furthermore, GMF is applicable to a uniprocessor systems.

The DGMF task model extends GMF with precedence dependencies and shared resources. The DGMF task model is also applicable to a multiprocessor system with partitioned scheduling.

GMF was not originally proposed for a specific domain. This task model was motivated by generic multimedia and control-command systems. Thus DGMF can also be applied to other domains than software radio protocols, for example the mentioned ones.

DGMF to Transaction

The analysis method for DGMF consists in exploiting another kind of model called transaction [143]. A transaction is a group of tasks related by precedence dependency. This thesis proposes an algorithm to transform DGMF tasks to transactions. The transformation solves the issue of difference in semantic between the two models.

Extension of Schedulability Test for Transactions

The transactions that are the results of the transformation have some characteristics not considered by current schedulability tests for transactions. Thus this thesis proposes the WCDSOP+NIM test, which is an extension of the WCDOPS+ test in [118]. The extension considers non-immediate tasks, which have a specific kind of task release.

Consider a successor task that has a precedence dependency with its predecessor task. In the immediate case, the predecessor task releases immediately the successor task, upon completion. In the non-immediate case the predecessor does not necessarily release immediately the successor task. For example, if the predecessor completes before the earliest release time of the successor task, the successor task is not released immediately.

This kind of release must be considered by the schedulability test. Otherwise, as we will see, the response times computed by the test may be overestimated or underestimated. These problems are solved by the schedulability test proposed in this thesis.

Transactions are not domain-specific and thus the schedulability test proposed in this thesis can be applied to other domains than software radio protocols.

Automatic Analysis with UML MARTE Models

To exploit the transaction and DGMF task models, an architectural model is proposed in UML MARTE. MARTE lacks clearly defined semantics for a specific RTES domain, and modeling guidelines.

This thesis thus shows an experience on modeling a software radio protocol with the UML MARTE modeling language. The model is described with UML structural diagrams and activity diagrams. The model is transformed automatically to the task model proposed in this thesis, for scheduling analysis.

Although initially motivated by software radio protocols, the model uses concepts of UML MARTE, which is a generic modeling language for RTES domains.

Available Tools

The GMF, DGMF, transaction models, and their analysis methods, are all implemented in Cheddar. As a reminder, Cheddar is an open-source scheduling analysis tool, available for both the research and industrial communities.

The UML MARTE model was done in Papyrus, an open-source modeler of Eclipse. The transformation of the UML MARTE model is implemented as a generic plug-in for Eclipse. The plug-in works on any model respecting the Eclipse implementation of UML.

THESIS ORGANIZATION

This thesis is divided into six chapters that are organized as follows.

In Chapter 1, a detailed presentation of RTES is given. This chapter focuses on entities of such a system concerned by scheduling. Some basic scheduling policies are presented. The chapter also exposes some development methods for RTES, focusing on models.

Chapter 2 gives a perspective of scheduling analysis. Some fundamental task models are presented and their analysis methods are explained. The chapter shows the relation of generalization between the different task models. The expressiveness of a task model, compared to its analysis difficulty, is discussed.

In Chapter 3, the software radio protocol system is presented. This chapter defines the assumptions and context of the work. It gives some details on the problems faced when trying to apply scheduling analysis to such a system. An overview of the solution is then presented.

Since no task models are applicable, the idea is to extend an existing one from the literature. To decide which task model to extend, an experiment is done in Chapter 4. This experiment consists in applying a fundamental task model to a software radio protocol. The task model is applied by modeling the system in UML MARTE, and transforming it to the task model.

The DGMF task model, extension of GMF, is proposed in Chapter 5. We will see how DGMF can be transformed into transactions and how tests for transactions need to be adapted for specific needs of DGMF tasks.

In Chapter 6, the WCDOPS+NIM schedulability test is presented. The schedulability test is applicable to tree-shaped transactions with non-immediate tasks. The chapter exposes the consequences of non-immediateness, before presenting the algorithm of the test.

DGMF tasks and transactions should be automatically produced from a system architecture model. For this purpose, in Chapter 7, the UML MARTE model of Chapter 4 is extended. The extension is based on existing Thales specification documents. The transformation of the UML MARTE model is then presented.

This thesis concludes by exposing some technical and methodological future works. The appendix sections contain technical details that will be referenced throughout the chapters.

Part I

STATE OF THE ART

Chapter 1

REAL-TIME EMBEDDED SYSTEM

Systems called Real-Time Embedded System (RTES) are now part of our daily life today. Their usage ranges from flight navigation software for avionic systems, to sensor data processing in autonomous vehicles, to everyday communication systems in smartphones. Unlike an Information Technology (IT) system, a RTES has special characteristics.

In this chapter, the concept of RTES is first defined. Then some of the entities of its architecture are presented: software, operating system, and hardware. Afterwards let us focus on the development cycle of a RTES. A development method called model-drive engineering, and its concepts, are presented. The application of this method and its concepts to RTES is the main focus.

1.1 GENERALITIES ON REAL-TIME EMBEDDED SYSTEMS

A RTES is both an embedded system and a real-time system. This sections defines such systems and describes their characteristics.

Definition 1 (Embedded System [74])**.** *An embedded system is a computing system composed of hardware and software, with a specific mission to accomplish, using limited resources (e.g. processing, memory, battery). This kind of system is embedded within a larger system. The embedded system, or the system that hosts it, may have an interaction with the environment in which it executes.*

As embedded systems may interact with their environment, for which the state evolves in time, they are also often real-time systems.

Definition 2 (Real-time System [135, 32, 74])**.** *A real-time system is a computing system composed of hardware and software, where computing is subject to time constraints. A real-time system takes as input some data then launches some behavior, or produces some data as output, within a time limit. As such the correctness of a real-time system's computing is not only judged by the correctness of the produced data (functional correctness), but also the time at which the data is produced (temporal correctness). Time constraints are generally referred as deadlines.*

A RTES can be characterized by its time constraints:
– Hard Real-Time: In hard real-time systems, time constraints must be met at all cost. The violation of time constraints may lead to human or material damage [150]. Examples of hard real-time systems are flight controls, nuclear plant control systems.
– Soft Real-Time: In soft real-time systems, time constraints must be met but their violation can be tolerated [72], sometimes only leading to a decreased Quality of Service (QoS) without any damage on the environment with which the system interacts. Examples of soft real-time systems are video decoders.
– Mixed-Criticality: Mixed-criticality systems [13] have constraints that must be met and constraints for which the violation is tolerable. In terms of time constraints, this means that some time constraints may be missed, but there is a limit to the number of times a time constraint is not respected. An example of a mixed-criticality system is a flight information system that

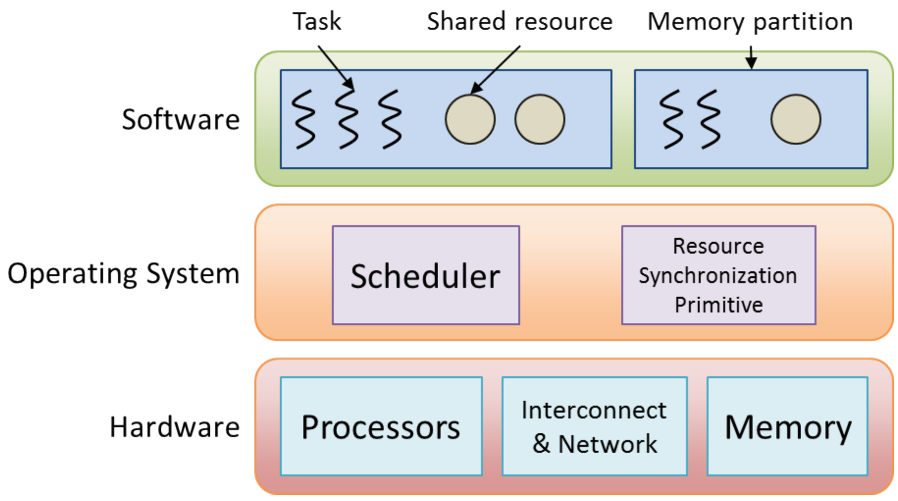


Figure 1.1: RTES Architecture

coexists with a flight control system. The flight control system has a higher criticality than the flight information system.

A RTES can also be characterized by the entities of its architecture. Figure 1.1 shows a very simplified architecture of a RTES. The *software* layer contains some tasks and shared resources in memory partitions. The *operating system* layer contains a component called scheduler, that schedules the tasks of the software. It also contains some primitives to protect the shared resources. Finally the *hardware* layer has some hardware components, like processor, bus and memory. The execution of the software and operating system is supported by these hardware components.

In the following three sections, the three layers of Figure 1.1 are described in detail and their entities are defined.

1.2 SOFTWARE OF A RTES

The software of a RTES accomplishes the mission of the system, with the help of the Operating System (OS) and hardware. It takes as input some data and, executes some operations, and may produce some data as output. These systems are sometimes called control-command systems. The operations must be done within some time constraint which must be met.

There are several ways to design the software of a RTES. Among these designs let us focus on the multi-tasking approach. In the following sections, the task concept is first defined. Then some types of tasks are presented. Afterwards common parameters of a task are defined. Tasks may have dependencies, through precedence and shared resource. These dependencies will be defined. Finally tasks are allocated in a memory partition, which are presented of the last section.

1.2.1 *Concept of Task*

In the multi-tasking design, the software has several units of execution, called tasks. In this section the concept of a task is defined and then its life cycle is exposed.

Definition 3 (Task [3, 126])**.** *A task is a unit of execution in a software. Each time a task is released, we say that a job of the task is released. Once released, a task has a number of instructions to execute sequentially. Instructions are executed on a processor and are either for computing or synchronization with other tasks.*

A task has a life cycle composed of several states. Figure 1.2 shows these states:

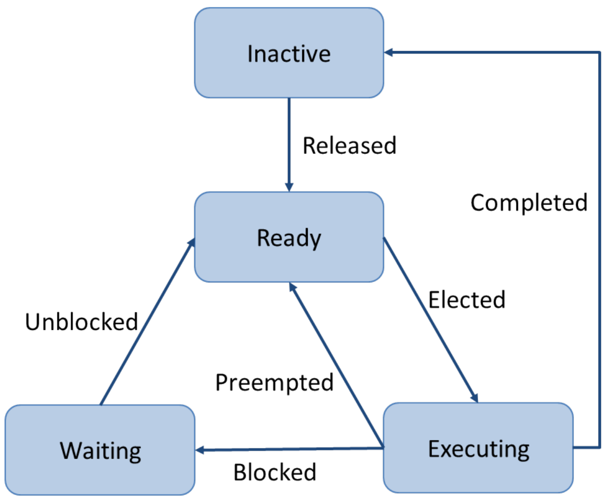


Figure 1.2: Task Life Cycle

- Inactive: Once the task is created, it starts in the inactive state. It does not have any data to handle, nor any operation to execute. When an event occurs, indicating to the task that it must handle some data and execute some operations, it is released and goes to the ready state.
- Ready: In the ready state, the task is waiting to be elected, among other tasks, for execution. The election process will be explained in later sections that describe the OS of a RTES.
- Executing: Once elected, the task executes its instructions on a processor. During execution, if at some point an instruction asks for some unavailable resource, the task goes to the waiting state. Furthermore if a task is executing on a processor, and another task is elected to execute on the same processor, then the executing task is interrupted and goes back to the ready state. It then waits to be elected again to resume it left off its execution. This phenomenon is called preemption. Finally once a task completes execution, it goes back to the inactive state.
- Waiting: When a task is waiting for a resource to become available, it is in this state. We say that the task is blocked. Once the resource becomes available, the task goes to the ready state to be elected for execution.

1.2.2 *Types of Task*

Tasks may be released by events occurring by different patterns. The release pattern can be used to define the type of the task:

- Periodic task [85]: Once a periodic task is released a first time, its next jobs are released in strictly regular intervals called the period. The periodic release mechanism may be implemented as a timer for example. An example of a periodic task is one that polls data from a sensor regularly.
- Sporadic task [16]: A sporadic task is released regularly, without necessarily a strict period. On the other hand the minimum time between two releases is known. A periodic task is a particular case of a sporadic task. An example of a sporadic task is one released by incoming Internet Packet (IP) packets. The IP packets arrive more or less quickly but the rate of arrival cannot be higher than the throughput of the network infrastructure.
- Aperiodic task [133, 137]: An aperiodic task is a task for which the minimum time between two releases is unknown, nor the first release time.

Tasks are said synchronous or asynchronous depending on when the tasks are released for the first time.

Definition 4 (Synchronous [5])**.** *Tasks are said synchronous if the first jobs of the tasks are released at the same time.*

Definition 5 (Asynchronous [17, 108]). *Tasks are said asynchronous if there is at least one first job of a task that is not released at the same time as the other first jobs of other tasks.*

1.2.3 *Parameters of Task*

Let us now see some parameters of a task that describe it. In the previous section, we saw how a task is released. Task releases may be subject to some jitter.

Definition 6 (Jitter [143, 105]). *A task released at earliest at* t*, and experiencing a release jitter of* J*, is released at any time in* $[t; t + J]$.

Jitter may be due to coarse grain clocks in the system. For example a task may have a period of 1 µs but the system clock is only as accurate as every 5 µs. Another example of jitter is when the scheduler in the OS switches from one executing task to another that just got elected. This phenomenon is called context switch.

Since tasks are in concurrence for access to the processor, there must be some mechanism to determine which task is of higher priority than another, for access to the processor, at any given time.

Definition 7 (Priority [3, 126]). *A task's priority indicates its order of importance for scheduling.*

We say that a task is of higher priority than another. At a given time, the highest priority task in the ready state should have access to the processor for execution.

The number of instructions executed by a task may be described by its execution time.

Definition 8 (Execution Time [10]). *The processor time taken by a task to execute its instructions is called an execution time.*

A task may not always execute the same set of instructions from a job to another. This means its execution time can vary from one job to another. The upper and lower bounds of its execution time are called Worst Case Execution Time (WCET) and Best Case Execution Time (BCET).

Definition 9 (Worst Case Execution Time [149]). *The WCET of a task is the longest execution time of a task.*

Definition 10 (Best Case Execution Time [149]). *The BCET of a task is the shortest execution time of a task.*

The execution time does not describe the total time taken by a task to complete its instructions. This time value is described by the response time of a task.

Definition 11 (Response Time [105]). *The time from a task's release to its completion is called a response time.*

The response time of a task is usually expressed relative to the release time of the task. It may be expressed according to another referential. For example for a given timeline, a response time may be expressed as absolute, i.e. a value on the timeline.

Like with execution times, a task also has bounds to its response time. It thus has a Worst Case Response Time (WCRT) and a Best Case Response Time (BCRT).

Definition 12 (Worst Case Response Time [105]). *A task's WCRT is the longest of the response times of any of its jobs.*

Definition 13 (Best Case Response Time [105]). *A task's BCRT is the shortest of the response times of any of its jobs.*

The time constraint of a task concerns its response time and is expressed as a deadline of the task.

Definition 14 (Deadline [3, 126]). *The deadline of a task is the longest allowed response time.*

Like a response time, a deadline is usually expressed relative to the release time of a task. We call this a relative deadline. But like response times, it may be expressed according to another referential. For a given timeline, a deadline may also be expressed as a value on the timeline. We call this an absolute deadline.

1.2.4 *Task Dependencies*

Tasks cooperate to accomplish the main mission of the software so they may have some dependencies between them.

Definition 15 (Dependent Tasks [3, 126]). *Two tasks are said dependent if they may synchronize at some point during their execution. Tasks that do not have dependencies are said independent.*

Dependency comes in two forms: precedence dependency and shared resource.

Definition 16 (Precedence Dependency [34, 3, 36]). *A task I has a precedence dependency with a task J, if I precedes J or J precedes I. I precedes J means that the* k^{th} *job of J can only be released after the completion of the* k^{th} *job of I. Precedence dependencies are transitive. If task I precedes a task J, and task J precedes a task K, then task I precedes task K.*

For example tasks may pass some messages among each other. This means the receiver task waits for a message from the sending task, and is thus released upon arrival of the message.
Tasks may also use shared resources in a mutual exclusive way.

Definition 17 (Shared Resource [127, 117, 3]). *A shared resource is a resource accessed by several tasks, in a mutual exclusive manner to enforce data consistency. The shared resource is protected by some primitive. A task that wants to access a unavailable shared resource is said to be blocked. It is blocked by another task that has access to the shared resource.*

Shared resources are used in critical sections during a task's execution.

Definition 18 (Critical Section [127, 117, 3]). *A (shared resource) critical section of a task, is an interval within a task's execution. At the start of the interval, the task asks for access to the resource by some call to a primitive, and locks the resource if available. At the end of the interval, the task unlocks the shared resource.*

In multiprocessor systems, shared resources may be local [127] or global [117]. A local shared resource is only used by tasks executing on the same processor. A global shared resource is used by tasks that execute on any processor. We then talk about local and global critical sections [117].

1.2.5 *Memory Partition*

The task and shared resource software entities are allocated in another software entity called a memory partition.

Definition 19 (Memory Partition [139]). *A memory partition is a list of memory locations. Supposing a task is allocated on a memory partition, the task can read from and write in the memory locations within the list.*

A shared resource is then allocated in a memory partition and a task executes within a memory partition. This means the task can only access instructions and data stored in the memory locations defined by the memory partition. Sometimes in the literature the term "address space" [139] is also used instead of memory partition.

1.3 OPERATING SYSTEM OF A RTES

The OS plays the role of interface between the software and the hardware. It allows the software to access the resources of the hardware and it may transmit events of the hardware to the software.

1.3.1 *Scheduler*

An important role played by the OS is that it elects which task of the software has access to the hardware processors at a given time. This is called scheduling and is done by a component called the scheduler.

Definition 20 (Scheduling [3, 126]). *Scheduling is a method by which tasks are given access to resources, noticeably computing resources, with the goal of respecting time constraints. Scheduling is done according to a scheduling policy.*

Definition 21 (Scheduling Policy [3, 126]). *A scheduling policy (or scheduling algorithm) is the algorithm which describes how tasks are given access to resources, like computing resources.*

Definition 22 (Optimal Scheduling Policy [3, 10, 126]). *A scheduling policy is said to be optimal among a group of scheduling policies, if a tasks set that is schedulable by a scheduling policy in the group, is also schedulable by the optimal scheduling policy.*

Definition 23 (Schedulable [10]). *A task is said to be schedulable under a scheduling policy, if none of its jobs, during execution, will ever miss their deadline. A tasks set is said schedulable under a scheduling policy, if all of its tasks are schedulable.*

Scheduling is done in the OS by a component called the scheduler. It elects tasks according to their priority. There exists many scheduling policies in the literature. In this thesis they are broken down into:
- Offline/online scheduling policies
- Preemptive/non-preemptive scheduling policies
- Fixed/dynamic priority scheduling policies
- Partitioned/global scheduling policies

The following sections present the difference between these policies.

1.3.1.1 *Offline/Online Scheduling*

Scheduling policies can be characterized as offline or online.

OFFLINE SCHEDULING Offline scheduling establishes a schedule before the system is executing, and thus scheduling decisions are taken offline [3]. The offline schedule must ensure that no deadlines are missed in the schedule. During execution, the role of the scheduler in the OS is simply to repeat infinitely the offline schedule. As such, the scheduler can be implemented as a table [3] where each entry contains a time at which to give a certain task access to the processor. The scheduler loops through the table indefinitely.

ONLINE SCHEDULING Online scheduling takes scheduling decisions during execution [3]. It is an algorithm that elects which task should have access to the processor at any given time. To elect a task, the task parameters are considered. They are used to determine which task has a higher priority than another.

Contrary to offline scheduling, online scheduling is flexible and can adapt to changes in the system [3]. On the other hand, offline scheduling guarantees that deadlines are met before the execution of the system, which is not the case with online scheduling. Thus an offline scheduling

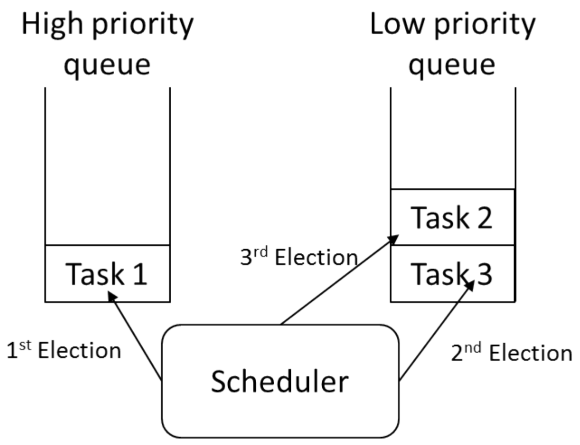


Figure 1.3: Scheduler Electing Tasks in Queues: It is assumed that tasks are all ready and only released once

analysis is necessary before using online scheduling, to guarantee that deadlines will be met during execution.

A scheduler running an online scheduling policy can be implemented as a queue(s) handling component, with queues containing task identifiers. This is the case for Portable Operating System Interface (POSIX) operating systems [28]. The way that the scheduler handles the queue(s), and elects a task, depends on the scheduling policy it implements.

Figure 1.3 shows an example where a simple scheduler elects ready tasks in two queues. There is a *high priority queue* and a *low priority queue*. Tasks in the *high priority queue* are elected first, so *task* 1 is elected first. Then tasks in the *lower priority queue* are elected. First *task* 2 is elected because it is at the head of the queue, then *task* 3 is elected.

1.3.1.2 *Preemptive/Non-preemptive Scheduling*

Scheduling policies can be characterized as preemptive or non-preemptive. Both of these scheduling policies are online.

preemptive In preemptive scheduling, the scheduler can arbitrarily suspend a task's execution, and resume it later [3]. This typically happens when a task, of higher priority than the preempted task, becomes ready when the preempted task is executing.

non-preemptive In non-preemptive scheduling, the scheduler does not suspend a task's execution [3]. For example if a low priority task is executing and a high priority is released during the execution of the low priority task, then the high priority task has to wait until the low priority task completes execution.

preemptive vs. non-preemptive Figure 1.4 shows an example of two schedules illustrating the difference between preemptive and non-preemptive scheduling.

In Figure 1.4a), *low priority task* is released first. During the execution of *low priority task*, *high priority task* is released. The latter must wait until *low priority task* completes execution before it can execute.

In Figure 1.4b), *low priority task* is released first. During the execution of *low priority task*, *high priority task* is released. The latter preempts *low priority task* since it is of higher priority. Then when *high priority task* completes execution, *low priority task* resumes execution.

In systems where there are only local shared resources, non-preemptive scheduling has the advantage that shared resources do not need to be protected [3]. Indeed, a task's execution cannot be

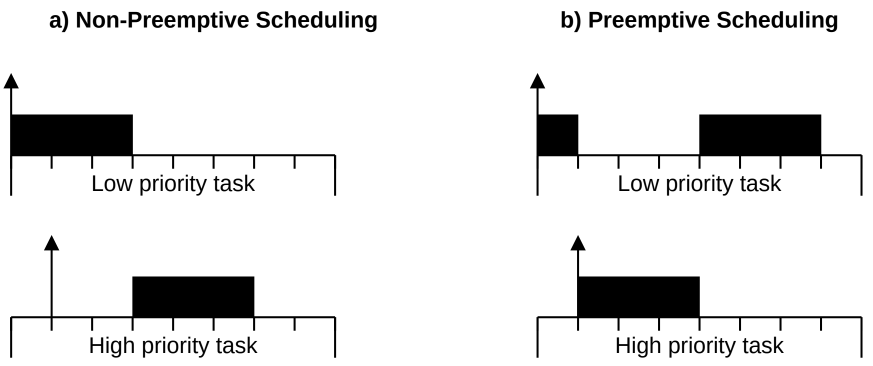


Figure 1.4: Preemptive vs Non-Preemptive Scheduling: Arrows are task releases

suspended, thus it cannot be preempted inside a critical section. On the other hand, preemptive scheduling increases the number of tasks set that are schedulable [3].

1.3.1.3 *Fixed/Dynamic Priority Scheduling*

A scheduling policy determines which task has access to the processor at a given time. The task with the highest priority is elected. Task may have a fixed priority or a dynamic one that may vary during execution. Both of these policies are online policies.

FIXED PRIORITY SCHEDULING In Fixed Priority (FP) scheduling, the priorities of tasks are fixed and assigned offline. The priorities are then used online for scheduling decisions. There are several methods to assign priorities. Among them, there are the rate-monotonic and deadline-monotonic methods:

– Rate-monotonic [85]: The Rate-Monotonic (RM) scheduling method assigns priorities for periodic tasks for which the relative deadline is equal to the period. Task priorities are inversely proportional to their period. Otherwise said, the task with the shortest period has the highest priority.
– Deadline-monotonic [76] The Deadline-Monotonic (DM) scheduling method assigns priorities for tasks. Task priorities are inversely proportional to their relative deadline. Otherwise said, the task with the shortest relative deadline has the highest priority.

RM and DM are methods to assign priorities but they are also called scheduling policies [85]. Preemptive RM and DM are said to be optimal among preemptive FP scheduling policies under certain conditions.

Theorem 1 (Optimality of preemptive RM [85])**.** *When tasks execute on a single processor, preemptive RM is optimal, within preemptive FP scheduling policies, for synchronous independent periodic or sporadic tasks that have a deadline equal to their period.*

Theorem 2 (Optimality of preemptive DM [76])**.** *When tasks execute on a single processor, preemptive DM is optimal, within preemptive FP scheduling policies, for independent periodic or sporadic tasks that have a deadline less than their period.*

DYNAMIC PRIORITY SCHEDULING In Dynamic Priority (DP) scheduling, the priority of a task may vary during execution due to the state of its other parameters. A task is said to have a dynamic priority computed according to its other parameters during execution, like its deadline. Some DP scheduling policies are:

– Earliest Deadline First [85]: At a given time during execution, the Earliest Deadline First (EDF) scheduling policy assigns the highest priority to the ready task with the nearest absolute deadline.

– Least Laxity First [43]: The Least Laxity First (LLF) scheduling policy assigns priority by laxity. For a job of a task, its laxity is defined as the difference between the task's relative deadline and its remaining execution time.

Preemptive EDF and LLF are said to be optimal among preemptive scheduling policies under certain conditions.

Theorem 3 (Optimality of EDF and LLF [85, 43])**.** *When tasks execute on a single processor, preemptive EDF and LLF are optimal, within preemptive scheduling policies, for independent periodic or sporadic tasks that have a deadline less or equal to their period.*

1.3.1.4 *Partitioned/Global Scheduling*

When tasks execute on a multiprocessor system, there are scheduling policies that define how tasks can execute on the different processors. Let us mention two policies:

PARTITIONED SCHEDULING In partitioned scheduling, each task is statically allocated on a processor [102, 86, 35, 40] and all the jobs generated by the task are required to execute upon that processor [11]. Tasks are scheduled locally on each processor as in a uniprocessor system.

GLOBAL SCHEDULING In global scheduling, different jobs of the same task may execute on different processors and a preempted job may resume execution on a processor different from the one it has been executing on prior to preemption [7, 11, 40]. We say that jobs are allowed to arbitrarily migrate across processors during their execution [35].

1.3.2 *Resource Synchronization Primitive*

When tasks use a shared resource, the OS handles the protection of the shared resource by offering primitives that tasks call when they want to access the shared resource. The method used by the OS to protect the shared resource may have a consequence on the scheduling of tasks.

A task may go from the executing state to the waiting state because a shared resource is unavailable during its execution. If no precautions are taken, this may result in unbounded priority inversions and deadlocks in the system. These phenomenons may be defined as follows:

Definition 24 (Priority Inversion [127])**.** *Priority inversion is the phenomenon where a task of higher priority is blocked by a lower priority task.*

Definition 25 (Unbounded Priority Inversion [127])**.** *Unbounded priority inversion is the phenomenon when a higher priority task is blocked by a lower priority task for an indefinite period of time.*

Definition 26 (Deadlock [139])**.** *A set of tasks is deadlocked if each task in the set is waiting for an event that only another task in the set can cause.*

In the case of tasks using shared resources, a deadlock occurs when a task *A* is blocked by a task *B* due to some shared resource *R1* and task *B* is blocked by task *A* due to some other shared resource *R2*.

Figure 1.5 shows an example of two schedules during execution of the system, where a deadlock and a unbounded priority inversion occur. The tasks are scheduled by a FP preemptive policy.

Figure 1.5a) shows a deadlock. A low priority task *L* is released first, it asks for a resource *R1*, and locks it. In the critical section where *L* uses *R1*, it is preempted by a high priority task *H*. Task *H* later asks for *R2*, and locks it. In the critical section where *H* uses *R2*, it asks for *R1* which is already locked by *L* so it waits for *L* to unlock it. *L* then continues execution and asks for *R1* but fails because *R1* is still locked by *H*. This results in a deadlock because *H* is waiting for *R2* locked by *L*, which is waiting for *R1* locked by *H*.

Figure 1.5b) shows a unbounded priority inversion. A low priority task *L* is released first, it asks for a resource *R*, and locks it. *L* is then preempted by a high priority task *H*. During execution of *H*,

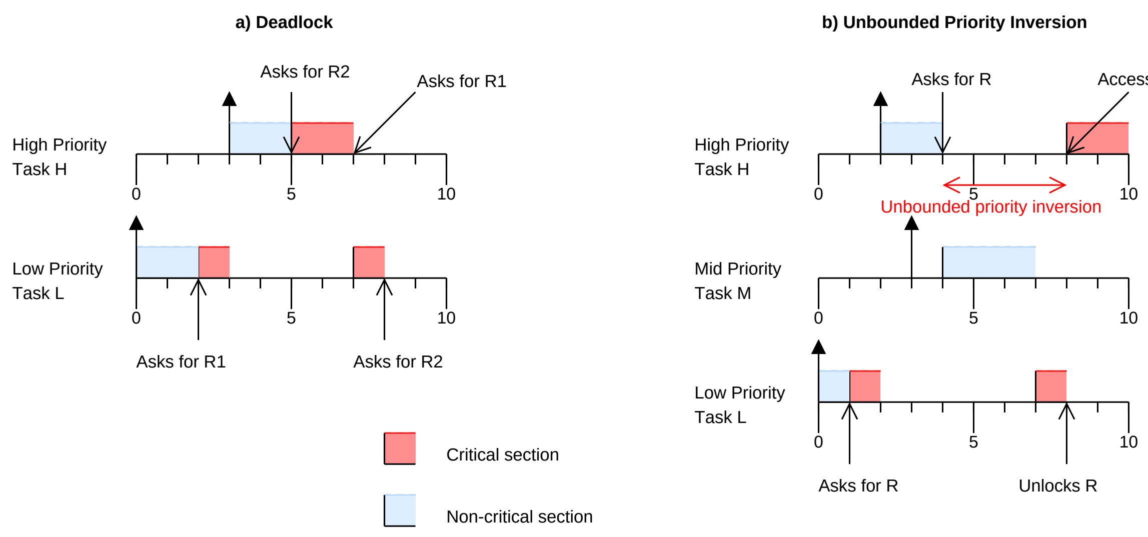


Figure 1.5: Deadlock and Unbounded Priority Inversion Example

a mid priority task M is released. H asks for R during its execution and has to wait for L to unlock it. At this time, the highest priority released task is M so M executes, even if it is of lower priority than H and does not block H due to a shared resource. Therefore the priority inversion is unbounded because H not only has to wait for the end of the critical section of L but also the completion of any jobs of tasks of higher priority than L, released before the end of the critical section, even if the tasks are of lower priority than H.

There are several access protocols for shared resource that prevent unbounded priority inversion. The basic idea of these access protocols is to modify task priorities when they ask for access to a shared resource. With such protocols, the blocking time of a task is bounded.

Definition 27 (Worst Case Blocking Time [127])**.** *The Worst Case Blocking Time (WCBT) of a task is an upper-bound to the blocking time of a task.*

Among the shared resource access protocols, three historical ones are presented in the following sections.

1.3.2.1 *Priority Inheritance Protocol*

The Priority Inheritance Protocol (PIP) was proposed in [127] to solve the problem of unbounded priority inversions. In PIP, when a task gets access to a resource, its priority is modified to the highest priority of tasks that it blocks. The modification of its priority occurs whenever a higher priority task asks for access to the already locked shared resource. When a task unlocks a resource, its priority is re-computed according to resources it is still locking. When a task exits all of its critical sections, it regains its initial priority.

PIP is only applicable to systems with fixed priority. It does not prevent deadlocks but it is possible to compute an upper-bound to the blocking time of tasks. With PIP, there are potentially very long WCBTs [127].

1.3.2.2 *Priority Ceiling Protocol*

The Priority Ceiling Protocol (PCP) proposed in [127] aims at reducing blocking times compared to PIP and prevent deadlocks. It is applicable to FP scheduling and later adapted for EDF in [31], which proposes the Dynamic Priority Ceiling Protocol (DPCP).

A ceiling priority is assigned to each shared resource. The ceiling priority of a resource is the highest priority of tasks that may use it. At a given time during execution, a system priority is defined as the highest ceiling priority of resources locked at the given time. When a task asks for access to a resource, it is given access only if its priority is strictly higher than the system priority. Otherwise the task is blocked and the task that already has access to the resource inherits the priority of the blocked task.

With PCP, a task H can only be blocked by at most one critical section of a task L of lower priority, using a resource with a ceiling priority higher than the priority of H [127].

1.3.2.3 *Stack Resource Policy*

Another resource access protocol applicable to EDF scheduling is the Stack Resource Policy (SRP). This protocol is furthermore applicable to systems where a resource has several instances and a task may access several instances of the resource.

In this protocol, each task is assigned a preemption level, which reflects its relative deadline. For example tasks may be assigned preemption levels by the DM method. During execution, a resource is given a ceiling value, defined as the maximum of preemption levels of tasks that want to access more instances of the resource than what is available. In the same way as with system priority in PCP, in the SRP a system has a system ceiling value during execution.

In PCP, tasks may be blocked when they asks to access resources. In the SRP, tasks may be blocked when they want to preempt another task. A task H can only preempt another task L when the following conditions are satisfied:

– Task H is of higher priority than task L.
– Task H's preemption level is higher than task L's.
– Task H's preemption level is higher than the system ceiling value.

The general idea behind the SRP is thus to prevent a task from starting execution, before all of the resources the task needs are available. Like PCP, the SRP prevents deadlocks and bounds blocking times.

As stated before, the OS is the interface between the software and the hardware. The next section focuses on the latter.

1.4 HARDWARE OF A RTES

This section describes the hardware platform of a RTES, through a model of the hardware from a scheduling point of view. As such, this section shows some relations between entities of the hardware and entities of the software/OS.

1.4.1 *Scheduling and Processors*

The hardware platform has processors for the execution of the software. The processor may either be a general-purpose one or dedicated to a specific kind of computing, like digital signal processing in a Digital Signal Processor (DSP).

The system may be characterized according to the number of processors and how they interact:

– Uniprocessor: There is only one processor.
– Multiprocessor [139]: There are several processors that share a common memory. The processors are connected by interconnects that are optimized for this kind of architecture [139]. The processors are either identical or heterogeneous. For readability, in this thesis a multiprocessor system scheduled by global scheduling is called a global multiprocessor system. One with partitioned scheduling is called a partitioned multiprocessor system.
– Distributed [139]: In a distributed architecture, there are several processors that do not share a common memory, Each has its own memory. The processors are connected by a network. These kind of systems are often referred as multicomputer [139] rather than multiprocessor.

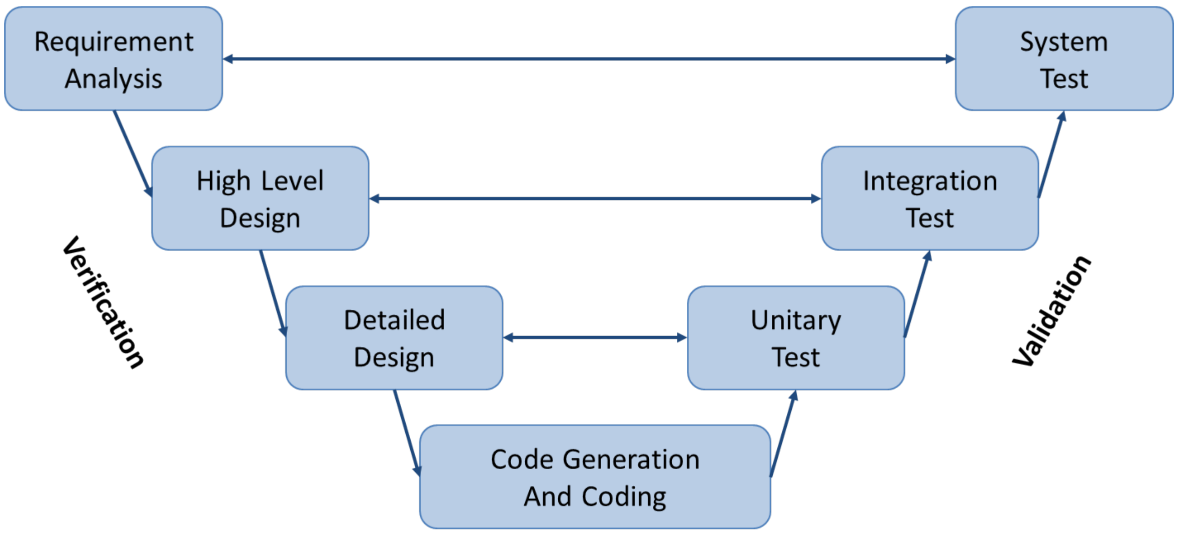


Figure 1.6: V-Model Development Cycle

1.4.2 *Scheduling and Networks*

The interconnects and networks connect the processors. They may schedule messages that pass through them, by a scheduling policy.

For example messages may have a fixed priority if transiting on some bus (e.g. Controller Area Network (CAN) bus [142, 120]). Thus messages are scheduled on network while tasks are scheduled on processors [105].

Through the previous sections, entities of a RTES were presented. The next section focuses on the development of a RTES.

1.5 DEVELOPMENT CYCLE

Due to the complexity of a RTES, the cost and delays of development, and the different expertise necessary for the development of its different entities, the system is developed through several steps described by a development cycle. A development cycle is a structure describing the steps that take place during the development of a product.

There exist several models of development cycles. One common model is the V-model [22], shown in Figure 1.6.

The V-model is separated into two distinct branches. One branch goes down and describes the steps from requirements to design, to implementation. This branch is called the verification branch because it verifies the requirements through reviews.

After implementation, the other branch goes up (hence the V shape) and focuses on testing. This branch is called the validation branch because it validates the development.

Each step of a branch is mirrored by another step of the other branch. For example, a step in the validation branch is supposed to validate a specification in the verification branch. Steps in the V-model are:

- *Requirements Analysis*: Requirements of the system are defined according to understanding of the client's needs. The requirements may be functional and non-functional. System testing can be specified in parallel to requirements analysis.
- *High-Level Design*: An abstract system is defined according to understanding of the requirements. The necessary components of the system are defined. This includes the interfaces of

each component and the functions they must implement. Integration tests can be specified in parallel to high level design.
– *Detailed Design*: Each component of the system is defined in detail. This includes how each function is implemented. Unitary tests can be specified in parallel to detailed design of components.
– *Code Generation and Coding*: The high-level and detailed design steps use models. The models may not only serve as documentation, but also for code generation. The generated code is then completed by manually written code.
– *Unitary Test*: The validation branch starts with unitary test. Each functions of each component is tested alone through functional test vectors.
– *Integration Test*: Once each unitary part of the software is functionally validated, they are assembled and integration tests validate end-to-end functionality between interacting components.
– *System Test*: System test is black box testing. A black box is a component for which the internal structure and functions are masked. In this step requirements are validated, and deviations between the developed system and the specified system are listed. Making sure requirements are met ensures that the client's needs are validated.

In the verification branch, the designs of the system are specified in two steps. Models may be exploited in these steps. In the next section, the concepts of models and model-driven engineering are introduced.

1.6 MODEL-DRIVEN ENGINEERING

Model-Driven Engineering (MDE) is a development methodology that aims to increase productivity by promoting interoperability between systems, simplification of the design process, and communication between teams working at different steps of the development cycle.

Definition 28 (Model-Driven Engineering [122])**.** *MDE is a software/hardware development methodology, where domain-specific models are exploited for verification and implementation, rather than computing concepts (i.e. algorithms).*

In the following sections the concepts of model and model transformation are exposed.

1.6.1 *Introduction to Models*

In the previous section, we saw that MDE is based on the description of a system with models.

Definition 29 (Model)**.** *A model is an abstraction of a system. A system can be described by different models, offering different levels of abstraction, which are related to each other.*

A model may be specific to a domain.

Definition 30 (Domain-Specific Model [67])**.** *A model of an area of interest or a particular development effort.*

A model describes a system with a modeling language.

Definition 31 (Modeling Language)**.** *A modeling language is a language used to express information, knowledge or systems in a structure defined by a set of rules. The rules define the semantics of the entities in the structure.*

A system is described by a model, while a model itself is described with a meta-model.

Definition 32 (Meta-Model)**.** *A meta-model is a model of a model. A meta-model describes the model entities and their relationships. Otherwise said a meta-model is a model of a modeling language.*

During the verification branch of the V-model development cycle, models can be created at each step. The lower in the verification branch we are, the less abstract the model is. Otherwise said a model created at a given step is a refinement of the more abstract model of the previous step. Models can thus be refined at each step, until a sufficiently detailed model is created for the implementation of the system. Refinement of model can also be done through model transformation.

1.6.2 *Model Transformation*

Models can be produced from other models through model transformation.

Definition 33 (Model Transformation [125, 98]). *Model transformation is the automated process of taking one or more source models as input, and producing one or more target models as output, following a set of transformation rules (also called mapping).*

To perform model transformation, a clear understanding of the abstract syntax and semantic of input and output models is necessary. Remember that the abstract syntax between entities of a model, and their relationship, are described by using meta-models. Transformation rules are then defined between meta-models of the input and output models [125]. Once the transformation rules are defined, any model conforming to the input meta-model can be transformed to a model conforming to the output meta-model.

The next section presents some modeling languages for describing the architecture of a system. Such models may be exploited for model transformation.

1.7 ARCHITECTURE DESCRIPTION LANGUAGES

A modeling language can be used to describe the - more or less abstract - architecture of the system. These kind of modeling languages are called Architecture Description Language (ADL) [97, 54], which can be defined as follows:

Definition 34 (Architecture Description Language [97]). *An ADL focuses on the high-level structure of the overall system rather than the implementation details of any specific source module.*

An ADL is a Component-Based Model (CBM) where architectural components, connectors, and architectural configurations are described [97].

Definition 35 (Component [97]). *A component in an architecture is a unit of computation or a data store.*

According to [97], an ADL must also offer the possibility to describe a component's interface.

Definition 36 (Interface [97]). *A component's interface is a set of interaction points between it and the external world. The interface specifies the services a component provides, and services the component requires.*

A service of a component can be defined as follows:

Definition 37 (Service [97]). *A service is a set of functionalities that can be reused. Examples of services are messages, operations, and variables of the component.*

Components are connected together, through their interface, by connectors.

Definition 38 (Connector [97]). *Connectors are architectural building blocks used to model interactions among components and rules that govern those interactions.*

Finally the connected components are shown in architectural configurations.

Definition 39 (Architectural Configuration [97]). *Architectural configurations are connected graphs of components and connectors that describe the architectural structure.*

Some ADLs are generic, while others are domain-specific. In the next sections, a generic ADL is first presented. Afterwards some domain-specific ADLs are briefly introduced.

1.7.1 *Generic ADL for RTES: UML MARTE*

A generic ADL for RTES is based on an extension of the Unified Modeling Language (UML) [121, 103], a standard modeling language of the Object Management Group (OMG).

In the next sections first UML and its extension mechanisms are presented. Then entities of the extension called Modeling and Analysis of Real-Time Embedded systems (MARTE) are introduced. Afterwards an example illustrates how a simple system is described in UML MARTE. Finally tool support for the modeling language is exposed.

1.7.1.1 *UML and Profiles*

For MDE, UML is promoted by the OMG, a standardization organization focusing on models. UML is a generic modeling language, and it the result of an effort to propose a standard software modeling language, by considering those from the 80s and 90s.

An entity in the UML meta-model is called a meta-class. Some examples of meta-classes will now be defined. Since this thesis does not focus on meta-modeling of UML, the meta-classes are defined as how they are used in the modeling work of this thesis.

BASIC META-CLASSES Some basic meta-classes of UML are:
- Class: A class describes a set of entities with the same attributes and operations. An attribute defines some data of the class.
- Operation: An operation is a service of the class. It may have some parameters that are typed by classes.
- Property: A property is a structure that represents an attribute of a class. A property is typed by a class, in which case the property may be an instance of the class.
- Constraint: A constraint represents some condition, restriction or assertion related to one or several UML entities. The entity owns the constraint.

DATA STRUCTURE META-CLASSES Some meta-classes of UML are dedicated to the modeling of data structures:
- Datatype: A datatype is a class that represents some data structure with values. The values may be typed properties. A datatype is typically used to type some property representing an attribute of a class.
- Enumeration: An enumeration is a datatype. Its values are only enumerated with enumeration literals which are not typed properties.

COMPONENT BASED MODEL META-CLASSES Some meta-classes of UML are dedicated to CBM:
- Component: A component is a class representing a re-usable part of a system. In this thesis, it represents a component as defined in an architecture model.
- Interface: An interface is a set of operations that must be implemented by any class that implements the interface.
- Port: A port is a part of a component through which other components access the component's operations, or through which the component access other components' operations. A port is typed by an interface so its operations are defined by an interface. Through a port, a component either provides the operations of the interface or requires them.
- Connector: A connector is a link between two components or two ports of two components. When a connector exists between two components, communication is enabled between the two entities.

BEHAVIOR META-CLASSES Finally some meta-classes are dedicated to modeling behaviors of the system. One such meta-class, used in this thesis, is an activity. An activity of a class or component is a sequence of actions and their conditional activation. The actions are connected by control flows of UML. The activity starts at an initial node and ends at an activity final.

relationships between meta-classes Some meta-classes are dedicated to the modeling of relationships between the entities of UML:

– Association: An association is a relationship between two classes that describes the reason of the relationship and the rules of the relationship. An association may be directed from a client to a supplier.
– Abstraction: An abstraction is a directed association between two classes that describe the same concept at different abstraction levels. The client is an abstraction of the supplier.
– Usage: An usage is a directed association between two classes. It means the full implementation of the client requires the supplier.
– Generalization: A generalization is a directed relationship that indicates that one class is a specific case of the other.
– Extension: An extension is a directed association between a stereotype and a meta-class. A stereotype is an extension mechanism of UML, defined below.

uml diagrams A UML model is described through different graphical diagrams. There are three kinds of diagrams: structural diagrams, behavioral diagrams, and interaction diagrams.

The static structure of the system is shown in structural diagrams. They are useful to model entities at different levels of abstraction of the system. Some examples of structural diagrams are:

– Class diagram: shows the structure of the system and the relationship between entities. Typically UML structural entities and their relations are specified in this kind of diagram.
– Composite structure diagram: shows the internal structure of an entity and how entities are connected together. Typically components, ports and connectors are represented in this kind of diagram.
– Profile diagram: is not used to model a system, but to define a UML profile that may then be used to model a system. In this kind of diagram, UML entities are extended.

The dynamic behavior of entities are shown in behavior diagrams. Some example of behavioral diagrams are:

– Activity diagram: shows sequences of actions executed by an entity, and conditions for actions to execute, i.e. an activity.
– State machine diagram: shows the behavior of the entity as a set of finite states, and conditions to transit from one state to the other.

Finally interaction between entities are shown in interaction diagrams. An example of an interaction diagram is a sequence diagram which shows interaction between entities in the form of messages exchanged between them.

The diagrams may be used to define views, that may be defined as follows:

Definition 40 (View [63])**.** *A view is a representation of the system from the perspective of some specific concerns.*

uml profile The generic modeling language UML is not itself an ADL, nor is it specific to the RTES domain. On the other hand, the concepts of profiles and stereotypes make it possible to define an ADL for RTES with UML.

Definition 41 (UML Profile)**.** *A UML profile extends UML for a specific domain. The extension mechanism is additive, so it does not contradict standard semantics of the original modeling language.*

Profiles are defined with stereotypes, tag definitions and constraints applied to specific entities in UML.

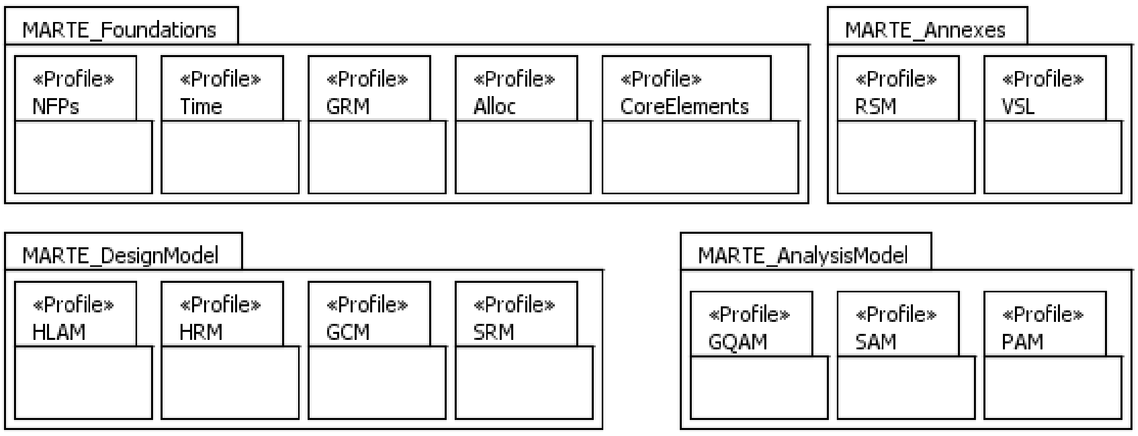


Figure 1.7: MARTE Packages

Definition 42 (Stereotype)**.** *A stereotype is a class of the profile that defines how a meta-class of UML is extended. A stereotype uses the "<<" and ">>" notation. A stereotype is applied to a meta-class in a UML model.*

A stereotype has a number of attributes that are used to tag other entities of the model.

Definition 43 (Tagged Value)**.** *In a UML model, an entity that is specified as the value of an attribute of a stereotype, is a tagged value.*

The next section presents a profile for UML, dedicated to the RTES domain.

1.7.1.2 *Entities of MARTE*

For the RTES domain, one of the most complete UML profile to date is MARTE [62, 42]. MARTE is a UML profile standardized by the OMG. It was specified to cover a large area of RTES, including avionic, automotive or software radio systems.

The profile adds capabilities to UML for MDE of RTES. It provides a modeling language for the design of such a system but also the modeling facilities to annotate the model for different kinds of analysis.

The different packages of MARTE are shown in Figure 1.7. The figure show that MARTE describes the system through a design model and an analysis model.

The design is independent of the analysis and thus the design model is independent of the analysis analysis model. This way it is possible to perform several kinds of analysis on the system, without having to modify its design model that may be used in the development cycle for documentation or to generate code.

MARTE has a number of sub-profiles, which are profiles containing partial entities of the MARTE profile. For example in Figure 1.7, *HLAM* is a sub-profile. The sub-profiles are regrouped within packages, which have the following signification:

– Foundations package: The foundations of MARTE contain fundamental concepts of a RTES and they are a base for the design and analysis packages.
– Design Model package: The design model package of MARTE extends the entities of the foundations package. The sub-profiles in the design model package are dedicated to describe the architecture of the system.
– Analysis Model package: The analysis model package extends entities of the foundations package for different kinds of analysis. There are two sub-profiles in this package. The General Quantitative Analysis Model (GQAM) sub-profile is for generic quantitative analysis. It offers

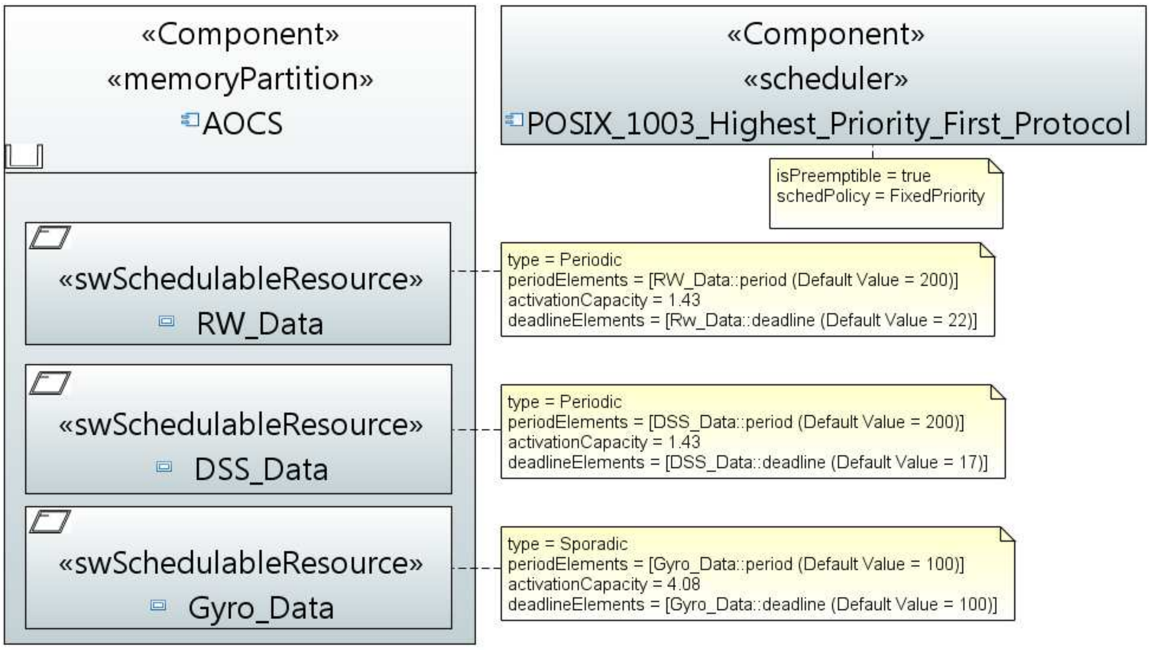

«Component»
«memoryPartition»
AOCS
«Component»
«scheduler»
POSIX_1003_Highest_Priority_First_Protocol
isPreemptible = true
schedPolicy = FixedPriority
«swSchedulableResource»
RW_Data
type = Periodic
periodElements = [RW_Data::period (Default Value = 200)]
activationCapacity = 1.43
deadlineElements = [Rw_Data::deadline (Default Value = 22)]
«swSchedulableResource»
DSS_Data
type = Periodic
periodElements = [DSS_Data::period (Default Value = 200)]
activationCapacity = 1.43
deadlineElements = [DSS_Data::deadline (Default Value = 17)]
«swSchedulableResource»
Gyro_Data
type = Sporadic
periodElements = [Gyro_Data::period (Default Value = 100)]
activationCapacity = 4.08
deadlineElements = [Gyro_Data::deadline (Default Value = 100)]

Figure 1.8: Example of Using of MARTE

a foundation for the Scheduling Analysis Model (SAM) sub-profile that is dedicated to scheduling analysis, and the Performance Analysis Model (PAM) sub-profile for performance analysis. The GQAM sub-profile can be extended for other kinds of analysis.

– Annexes package: The annexes package contains facilities offered by MARTE to annotate non-functional properties in the model. For example, it offers the Value Specification Language (VSL) to express non-functional properties.

MARTE is a profile for UML so to model an entity in MARTE, a stereotype of the profile is applied to a UML meta-class. The next section shows an example of use of MARTE for the modeling of a RTES.

1.7.1.3 *Example of Use*

Figure 1.8 shows an example of task, memory partition and scheduler in MARTE. Tshe three tasks called *RW_Data*, *DSS_Data*, and *Gyro_Data* are allocated on the *AOCS* memory partition.

The memory partition is modeled as a UML component stereotyped <<MemoryPartition>>, while tasks are modeled as properties of the component stereotyped <<SwSchedulableResource>>. Tasks are scheduled by a fixed priority scheduler stereotyped <<Scheduler>>.

The attributes of the MARTE stereotypes tag some properties of the components, to give them a semantic. For example *period* property of *RW_Data* is tagged as the period of the task.

1.7.1.4 *Tool Support*

Since MARTE is a profile for UML, it benefits from the eco-system of UML modelers. For example MARTE is implemented by IBM's Rational Software Architect (RSA) [131] modeler. It is also implemented in Papyrus [104], an open-source UML modeler for Eclipse. Finally MARTE is also implemented in Softeam's Modelio [130], a commercial UML modeler implemented above Eclipse, with an open-source version that also exists.

1.7.2 *RTES Domain-Specific Architecture Description Languages*

Beside UML MARTE, there are several ADLs that are dedicated to a specific domain within the RTES domain. In the following sections three standard domain-specific ADLs are briefly introduced. For each ADL, entities and tool support are described.

1.7.2.1 *AADL*

Architecture Analysis & Description Language (AADL) [49] is a standard ADL of the Society of Automotive Engineers, proposed in 2004. AADL has strong roots in the avionics domain.

ENTITIES OF AADL MODEL An architecture in AADL is described as a set of connected components, and interfaces. The components of AADL are separated according to three categories. The entities of a RTES, presented previously in this chapter, can be organized into these categories:
– Software: tasks and data
– Hardware: processors, memories and buses
– System: a component called "system" that wraps the software and hardware components

TOOL SUPPORT There exists several modelers that support AADL. For example OSATE [124] is an Eclipse plug-in that lets the user describe an AADL model either graphically, in textual format, or in the Exensible Markup Language (XML) format.

1.7.2.2 *EAST-ADL*

EAST-ADL [41] is an ADL dedicated to the automotive domain. EAST-ADL is designed to complement the AUTomotive Open System ARchitecture (AUTOSAR) [73]. AUTOSAR is an open and standardized software architecture for the automotive domain.

EAST-ADL complements the AUTOSAR standard by offering a modeling language to describe a higher level of abstraction of the vehicular system.

ENTITIES OF EAST-ADL MODEL An EAST-ADL model contains entities that are divided into four levels of abstraction:
– Vehicle level: entities to represent functionality of the system, without knowledge of the solution to realize the functionalities (i.e. independent of software and execution platform)
– Analysis level: abstract decomposition of the vehicle into set of functions, represented by components with internal and external interfaces
– Design level: contains the functional architecture (set of functions represented by components with internal and external interfaces), hardware architecture (hardware components), and allocations of functional components onto hardware components
– Implementation level: relies on AUTOSAR design model standards

TOOL SUPPORT EAST-ADL currently exists as a UML profile. Papyrus, the open-source UML modeler of Eclipse also offers an implementation of the EAST-ADL profile, with diagrams and palettes. Some domain-specific tools, such as MetaCase MetaEdit+ [144], were also adapted for EAST-ADL within the scope of some European projects.

1.7.2.3 *MyCCM*

The ADL of Make your CORBA Component Model (MyCCM) is part of the MyCCM framework [25]. MyCCM provides tools for modeling and code generation. It is based on the the concept of separation of wrapper, communication, and business code of a CBM, shown in Figure 1.9.

The wrapper code of a component is the code used for interfacing of the component with other components. The communication code implements the communication between components, i.e.

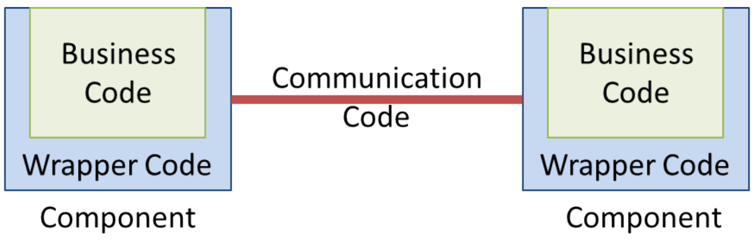


Figure 1.9: Component-Based Model

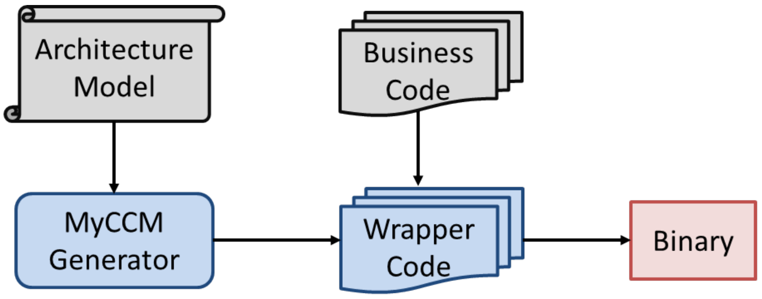


Figure 1.10: MyCCM Code Generation

code of the connectors. The business code of the component is the functions and procedures implemented by the component.

Usually the business code is written manually by the developers at the coding step of the development cycle. The wrapper code of a component, and the communication code between components, are generated from the architecture model, by MyCCM.

Development with MyCCM is summarized in Figure 1.10. The *architecture model* of the system is described in the ADL of MyCCM. Afterwards the model is input into the *MyCCM generator* that generates the *wrapper code*. *Business code* is then integrated into the components and compiled to get the *binary*.

The ADL of MyCCM is based on the CORBA Component Model (CCM) [148] modeling language. CCM is dedicated to the modeling of components in Common Object Request Broker Architecture (CORBA) [146, 23], a standard of the OMG for communication between components on heterogeneous execution platforms.

ENTITIES OF MYCCM MODEL Entities of a CCM model are:

– Component: As defined by components of an ADL
– Port: A mechanism to interact with other components
– Container: A container wraps a component and offers it some services.

There exists four kinds of ports defined in [148]:

– Facet: A facet is an interface that specifies services provided by the component.
– Receptacle: This kind of port specifies services needed by the component to function correctly. They are provided by other components.
– Event source/sink: A component may produce events through a source port, and observe events through a sink port.
– Attributes: This kind of port represents attributes of the component.

Ports are connected together: a facet is connected to a receptacle, an event source is connected to an event sink.

TOOL SUPPORT The MyCCM framework is implemented as a plug-in for the SpectraCX [112] modeler. SpectraCX is a modeler implemented above IBM's RSA [131], itself implemented above Eclipse.

1.8 CONCLUSION

In this Chapter various concepts related to RTES were introduced. The chapter focused on the software, and execution platform (OS and hardware) of a RTES. Some important concepts were defined, especially concerning tasks and scheduling.

Afterwards the chapter focused on the development of a RTES through its development cycle. A development methodology called MDE was presented. This methodology exploits models of the system, to serve the purposes of the different steps of the development cycle. The concept of architecture models, and ADLs dedicated to the RTES domain, were then exposed.

One important step part of the development of a RTES, is to verify its time constraints. This can be done with various methods called scheduling analysis, which will be presented in the next chapter.

Chapter 2

SCHEDULING ANALYSIS

The previous chapter showed that a RTES has several time-constrained tasks scheduled by some scheduling policy, on some processors. These tasks may have dependencies through precedence and shared resources, for which there exist a number of access protocols. The time constraints of tasks must be verified during the development of the system. This can be done with scheduling analysis. In this chapter scheduling analysis is presented. This kind of analysis determines if tasks of a RTES respect their deadlines.

In the next sections of this chapter, methods of scheduling analysis are first presented. Afterwards we will focus on the concept of task models for scheduling analysis. Then some task models of the literature, and their analysis method, will be exposed in detail.

2.1 METHODS OF SCHEDULING ANALYSIS

Scheduling analysis either determines the feasibility or schedulability of a system. These terms are defined as follows:

Definition 44 (Feasibility [10, 12])**.** *For a given tasks set executing on a given set of processors, the tasks set is said to be feasible if the tasks set is schedulable on the processors by at least one existing scheduling policy.*

Definition 45 (Schedulability [10, 12])**.** *For a given tasks set executing on a given set of processors, and a given scheduling policy, schedulability is the assessment that the tasks set is schedulable by the scheduling policy on the processors.*

There are several scheduling analysis methods in the literature. Some of these methods are shown in Figure 2.1.

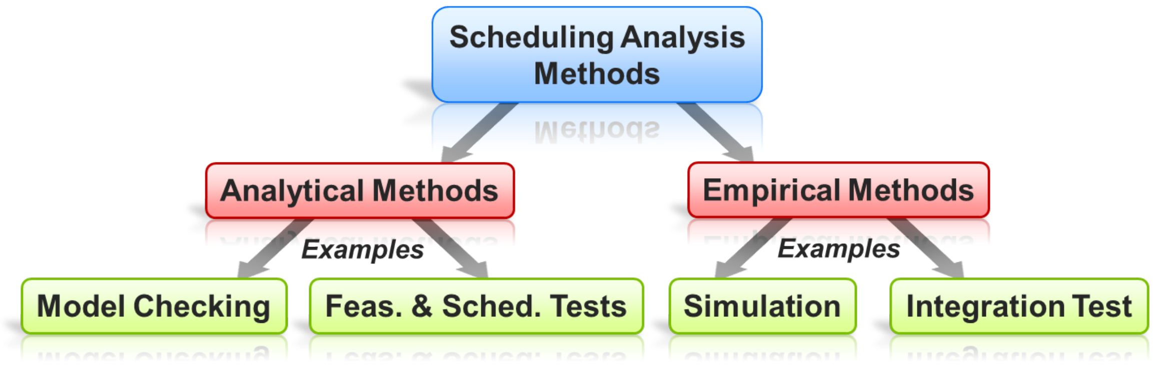


Figure 2.1: Scheduling Analysis Methods

2.1.1 *Empirical Methods*

Empirical scheduling analysis methods do not always guarantee schedulability. On the other hand they may determine unschedulability: there is at least one case where deadlines are missed. Some empirical methods are: mention two:

– Simulation: Simulation is the imitation of real world operations. In the case of scheduling, it is the computation of a schedule according to an input tasks set, processors set, and scheduling policy of the processors [36]. When not exhaustive, simulation is not able to determine schedulability. Scheduling simulation tools usually use the WCET of tasks [145] but recent trends see the emergence of simulators that do not always use the WCET of tasks to determine the performance of a scheduler [33].
– Integration Test: Integration test is a step in software testing where individual software pieces are combined and tested together. As such integration tests may execute individual tasks together and check for missed deadlines. Contrary to simulation, integration tests are generally performed on an implemented system, or partially implemented.

2.1.2 *Analytical Methods*

Analytical scheduling analysis methods are based on equations, or formal methods to determine schedulability (or feasibility). Contrary to empirical methods, these methods do guarantee schedulability, given the assumptions of the system to analyze. Among analytical methods, let us mention two:

– Model Checking: Model checking takes as input the model of a system and some of its properties. The model of the system is expressed in a mathematical language. The analysis then proceeds to check, exhaustively, whether the model meets its properties. Properties may contain time constraints. Some examples of model checking tools for real-time systems are UPPAAL [18] and ROMEO [53]. UPPAAL uses timed automatas [19] to model the system, while ROMEO uses timed petri nets [110].
– Feasibility and Schedulability and Tests: This thesis focuses on this kind of scheduling analysis method. Feasibility and schedulability tests are defined below.

Definition 46 (Feasibility Test [10, 12])**.** *A feasibility test assesses if a tasks set, executing on a given set of processors, is feasible.*

Determining the feasibility of a tasks set may be done by imposing some constraints on the set of scheduling policies that may schedule the tasks set [15]. For example example feasibility may be determined only for preemptive against non-preemptive, FP against DP policies.

Definition 47 (Schedulability Test [10, 12])**.** *A schedulability test assesses if a tasks set, executing on a given set of processors, with a given scheduling policy, is schedulable.*

To illustrate feasibility and schedulability, consider an example of a uniprocessor system, with a set of synchronous independent periodic tasks with their deadline less or equal to their period.

Since EDF is an optimal scheduling policy, among preemptive scheduling policies feasibility determination, for preemptive policies, consists in determining the schedulability of the tasks set by the EDF policy [10]. If the tasks set is schedulable by EDF then there is at least one policy that can schedule them. Therefore the tasks set is feasible.

Feasibility determination for preemptive FP policies is different. Determining if the tasks set can be scheduled by at least one preemptive FP policy, comes in two forms [10]:

– Priority testing: Given a tasks set with fixed priorities assigned, does there exist a preemptive FP scheduling policy that can schedule the tasks set, on a given set of processors, without any missed deadlines?

– Priority assignment: Given a tasks set, does there exists a priority assignment method (e.g. RM) that assigns priorities so that the tasks set can be scheduled by a preemptive FP scheduling policy, on a given set of processors, without any missed deadlines?

Feasibility and schedulability tests have conditions that assess whether a tasks set's feasibility or schedulability is met. These conditions may either be sufficient or necessary or both:

– Sufficient: A tasks set that satisfies these conditions is enough to be assessed as feasible (resp. schedulable). If a tasks set does not satisfy these conditions, it may still be feasible (resp. schedulable).
– Necessary: A tasks set that does not satisfy these conditions is not feasible (resp. schedulable). If it does satisfy these conditions, it may still not be feasible (resp. schedulable).
– Necessary and sufficient: A tasks set is feasible (resp. schedulable) only if it respects these conditions. We say that the conditions are exact.

Feasibility and schedulability tests are equations and algorithms that computed different values:

– Processor utilization: Processor utilization is the fraction of time the processor is executing tasks.
– Response time: Response times are computed and compared to deadlines to assess schedulability. The method to compute response times is called Response Time Analysis (RTA).
– Processor demand: Tasks demand access to the processor. Tests based on processor demand computes the cumulative demands of a processor by tasks, in a time interval. This is usually done with a demand bound function. A tasks set's schedulability is then assessed by studying their demand bound on some intervals that are enough to assess the tasks set's schedulability.
– Speedup bound: Determining exact feasibility of a tasks set, on a global multiprocessor, is intractable [24]. Thus the idea is to compute an approximate result. The approximation is based on the concept of schedulability tests defined by a speedup bound, A speedup bound is an increase of the processors' speed. If the tasks set is feasible on the processors with unitary speed, then the test determines that it is schedulable on the processors with increased speed [24].

Feasibility and schedulability tests take as input a system described with a model called a task model. Some task models of the literature are presented in the next section.

2.2 TASK MODELS AND TESTS

A task model is scheduling analysis domain-specific model used for feasibility and schedulability tests. There exists numerous task models in the literature, which have been proposed either for a specific RTES system, or a specific hardware architecture. Figure 2.2 shows some of the most well-known task models.

From Figure 2.2, we see that task models usually have a relation of generalization. When a task model generalizes another, it increases its expressiveness, at the cost of an increase in analysis difficulty. The order by expressiveness and analysis difficulty, is also a chronological one, as task models with more expressiveness have been proposed later, motivated by the increasing complexity of RTES.

The task models in Figure 2.2 can be grouped together, Each group is highlighted by its color in the figure. These groups are:

– Fundamental: periodic and sporadic task models
– Transaction: linear, tree-shaped and graph-shaped transactions
– Multiframe: multiframe, General Multiframe (GMF) and non-cyclic GMF
– Directed Acyclic Graph (DAG) task: Recurring Real-time Task (RRT), non-cyclic RRT, sporadic DAG, and Digraph Real-time Task (DRT)

The fundamental and transaction models are dedicated to the modeling of individual tasks, and their dependencies. The multiframe and DAG models are dedicated to the modeling of jobs of tasks.

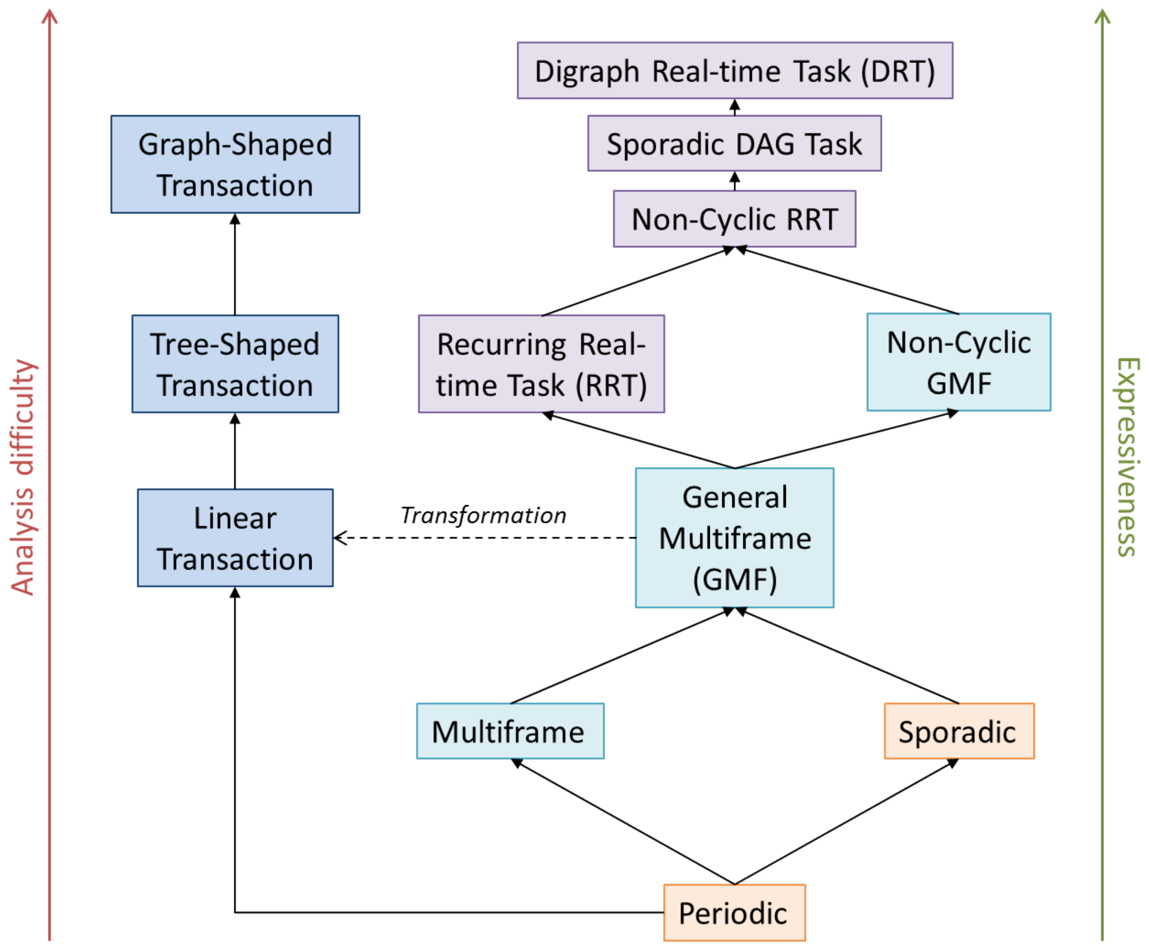


Figure 2.2: Task Models: Filled arrows between task models indicate generalization

In the next section each group is presented through one of its task models. Tests for the task model are also exposed.

2.3 FUNDAMENTAL MODELS: PERIODIC AND SPORADIC

The periodic task model is based on the seminal Liu and Layland task model proposed by in [85] for RM scheduling. Since then the periodic task model has been extended with release jitter and WCBT. Furthermore some constraints of parameters of the model in [85] have been relaxed, such as the implicit "deadline equal to period" constraint. A periodic task is only a particular case of a sporadic task [16], and thus the two task models may be defined with the same parameters.

In the following sections, first the definition, of the periodic and sporadic task models, is given. Afterwards three types of tests are presented: test based on processor utilization, response time, and processor demand.

2.3.1 *Definition*

A periodic task, denoted by τ_i, is a task with parameters:

- C_i is the WCET.
- D_i is the relative deadline.
- T_i is the period.
- $\mathrm{prio}(\tau_i)$ is the fixed priority (for FP scheduling).
- J_i is the release jitter.
- B_i is the WCBT due to shared resources.

A sporadic task is defined with the same parameters as a periodic task, with some differences in semantics and constraints. The definition of the T_i parameter changes: T_i is the minimum-separation time between two releases of a sporadic task. Furthermore a sporadic task does not have the constraint of $D_i = T_i$ since T_i is not a strict period.

Although semantically different, a sporadic task can be defined with the same parameters as a periodic task. The period parameter is the minimum-separation time of the sporadic task [134]. For readability, this section on the periodic and sporadic task models, will only refer to the periodic task model.

All of the parameters are set arbitrarily - while still respecting constraints of the scheduling policy, e.g. constraints of RM - except B_i, which is computed. The following paragraphs show how to compute the B_i parameter of a task, when shared resources are protected by the PIP or PCP access protocols, proposed in [127].

Theorem 4. *[127] Under PIP, the WCBT B_i of a task τ_i is:*

$$B_i = \sum_{R} \max_{j \neq i} \left(\textit{Critical section duration of } \tau_j \right) \tag{1}$$

with R denoting a shared resource.

Theorem 5. *[127] Let R denote a shared resource, $C(R)$ the ceiling priority of R, and $D_{j,R}$ the duration of the longest critical section of task τ_j using shared resource R. Under PCP, the WCBT B_i of a task τ_i is:*

$$B_i = \max_{j,R} \left(D_{j,R} \mid \mathrm{prio}(\tau_j) < \mathrm{prio}(\tau_i), C(R) \leqslant \mathrm{prio}(\tau_i) \right) \tag{2}$$

In the following sections, some feasibility and schedulability tests for the periodic task model are presented.

2.3.2 *Processor Utilization Feasibility Tests*

The processor utilization is the fraction of time the processor is executing tasks. The processor utilization is expressed as a value between 0 and 1.

The utilization of the processor by a task τ_i is computed as:

$$U_i = \frac{C_i}{T_i} \tag{3}$$

For example if a task τ_i has a period of $T_i = 4$ and a WCET of $C_i = 2$, then it uses, in the worst case, the processor during 2 time units, every 4 time units. Thus its processors utilization is $2/4 = 0.5 = 50\%$.

The total processor utilization of a tasks set of n tasks (allocated on the same processor) is then the sum of processor utilization of each task:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \tag{4}$$

The following sections show some feasibility tests that determine if there is at least one preemptive FP or DP scheduling policy that can schedule a set of synchronous independent tasks. The tasks have some constraints on their deadline and period, which will be mentioned in the description of the tests. The tests are applicable to a uniprocessor system.

2.3.2.1 *Preemptive FP Scheduling*

Assessing if a preemptive FP scheduling policy can schedule synchronous independent tasks, with $D_i = T_i$, is done with the following theorem:

Theorem 6. *[85] A tasks set of n synchronous independent periodic tasks, executing on a uniprocessor, and with $D_i = T_i$, is schedulable by preemptive RM if:*

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leqslant n\left(2^{1/n} - 1\right) \tag{5}$$

where U is the processor utilization.

Therefore if the condition of Equation 5 is respected, then at least preemptive RM can schedule the synchronous independent tasks set, with $D_i = T_i$. The tasks set is then feasible on a uniprocessor.

Similarly, assessing if a preemptive FP scheduling policy can schedule synchronous independent tasks, with $D_i \leqslant T_i$, is done with the following theorem:

Theorem 7. *[76] A tasks set of n synchronous independent periodic tasks, executing on a uniprocessor, and with $D_i \leqslant T_i$, is schedulable by preemptive DM if:*

$$U = \sum_{i=1}^{n} \frac{C_i}{D_i} \leqslant n\left(2^{1/n} - 1\right) \tag{6}$$

where U is the processor utilization.

Therefore if the condition of Equation 6 is respected, then at least preemptive DM can schedule the synchronous independent tasks set, with $D_i \leqslant T_i$. The tasks set is then feasible on a uniprocessor.

Notice that the difference between Equation 6 and Equation 5 is that the deadline D_i is used in Equation 6, while the period T_i is used in Equation 5.

Property 1. *[85] The processor utilization by a set of periodic tasks under RM and DM tends towards 0.69, when the number of tasks increases.*

$$\lim_{n\to\infty} n\left(2^{1/n} - 1\right) = \ln 2 \approx 0.69 \tag{7}$$

Later in [127], the B_i parameter is integrated into the processor utilization based test for preemptive RM:

Theorem 8. *[127] Under RM scheduling, a tasks set $\tau_1..\tau_n$, ordered by increasing priorities, of tasks using shared resources protected by PCP, is schedulable if:*

$$\forall i \in [1;n], \frac{B_i}{T_i} + \sum_{j=1}^{i} \frac{C_j}{T_j} \leqslant i\left(2^{1/i} - 1\right) \tag{8}$$

2.3.2.2 *Preemptive DP Scheduling*

Assessing if a preemptive scheduling policy can schedule synchronous independent tasks, with $D_i = T_i$, is done with the following theorem:

Theorem 9. *[85] A tasks set of n synchronous independent periodic tasks, executing on a uniprocessor, and with $D_i = T_i$, is schedulable by preemptive EDF if, and only if:*

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leqslant 1 \tag{9}$$

Therefore if the condition of Equation 9 is respected, then at least preemptive EDF can schedule the synchronous independent tasks set, with $D_i = T_i$. The tasks set is then feasible on a uniprocessor.

Compared to optimal preemptive FP scheduling policies like preemptive RM and DM, it is possible to achieve more processor utilization with an optimal preemptive DP scheduling policy like preemptive EDF [85]. Indeed with EDF it is possible to achieve 100% utilization when the number of tasks tends towards infinity, while with RM or DM it is possible to achieve about 69% when the number of tasks tends towards infinity.

In this section feasibility tests that compute processor utilization were exposed. In the next section a schedulability test that computes response times is presented.

2.3.3 *Response Time Schedulability Test*

As a reminder, a response time of a task is the time between the task's release and its completion. To assess if a task is schedulable, its WCRT is compared to its deadline. The computation of response times is called RTA and it has been studied extensively for periodic tasks on a uniprocessor system (e.g. [64, 4, 21]). RTA is based on concepts like busy period, critical instant, and workload. The following section first define these basic concepts. Then a schedulability test is presented.

2.3.3.1 *Basic Concepts*

Definition 48 (Busy Period of Processor [75]). *The processor busy period is the time interval during which the processor is busy executing jobs of tasks.*

Definition 49 (Busy Period of Task [75]). *The busy period of a task is the time interval during which the processor executes jobs of the task and jobs of other tasks of higher priority or equal priority.*

Definition 50 (Critical Instant [143]). *The time at which the busy period (of a processor or task) starts is called the critical instant.*

Definition 51 (Workload [14]). *The workload of jobs of a task within a time interval is the total processor time that the execution of its jobs occupy.*

The workload of jobs of a periodic task τ_l in an interval $[0;t]$ is the sum of the WCETs of its jobs:

$$\forall t > 0, W_l(t) = \left\lceil \frac{t}{T_l} \right\rceil \cdot C_l \tag{10}$$

In Equation 10, $\left\lceil \frac{t}{T_i} \right\rceil$ computes the number of jobs of a task in $[0;t]$. The workload of a processor in an interval $[0;t]$ is then the sum of workloads of n tasks allocated on the processor:

$$\forall t > 0, W(t) = \sum_{i=1}^{n} \left\lceil \frac{t}{T_i} \right\rceil \cdot C_i \tag{11}$$

The $W(t)$ function is used to compute the busy period of a processor. The critical instant leading to the highest processor busy period starts when all tasks are released at the same time [85]. For n tasks, the length of the busy period of the processor is denoted by L and is computed by iteration:

$$\begin{aligned} L^{(0)} &= \sum_{i=1}^{n} C_i \\ \forall k > 0, L^{(k)} &= W(L^{(k-1)}) \end{aligned} \tag{12}$$

Equation 12 converges only if the processor utilization is less or equal to 1 ($U \leqslant 1$).

The following section shows how these concepts are applied in a schedulability test for the periodic task model.

2.3.3.2 *Schedulability Test for Periodic Tasks*

The following paragraphs present a RTA schedulability test for tasks scheduled by a preemptive FP scheduling policy on a uniprocessor. The tasks are assumed synchronous. They are also assumed independent but we will see how the test can be extended for jitters and WCBTs.

The WCRT of a periodic task τ_i is computed according to the following theorem:

Theorem 10. *[75] The WCRT of a task τ_i is the response time of one of its jobs that occur in the busy period of τ_i, starting at a critical instant where all tasks are released at the same time.*

By applying Theorem 10, the WCRT of a task τ_i is obtained by the following steps:

– Compute the length of the busy period of τ_i
– Compute the number of jobs of τ_i that occur in the busy period
– Compute the response time of each job and take the maximum response time as the WCRT of τ_i

In the following paragraphs, let us denote by hp_i the set of tasks, different to τ_i, of higher or equal priority to task τ_i.

LENGTH OF BUSY PERIOD Let us denote by W_i the function that computes workload of jobs of a periodic task τ_i and tasks in hp_i, in an interval $[0;t]$ [64]:

$$\forall t > 0, W_i(t) = \sum_{\tau_j \in hp_i \cup \{\tau_i\}} \left\lceil \frac{t}{T_j} \right\rceil \cdot C_j \tag{13}$$

The length of the busy period of τ_i, denoted by L_i, is computed similarly to the way the length of the busy period of a processor is computed (Equation 12). This time the computation only takes the workload of jobs of tasks that are of higher or equal priority to τ_i, plus jobs of τ_i itself:

$$\begin{aligned} L_i^{(0)} &= \sum_{\tau_j \in hp_i \cup \{\tau_i\}} C_j \\ \forall k > 0, L_i^{(k)} &= W_i(L^{(k-1)}) \end{aligned} \tag{14}$$

NUMBER OF JOBS IN BUSY PERIOD The number Q of jobs that occur in the busy period of τ_i of length L_i is:

$$Q = \lceil L_i/T_i \rceil \tag{15}$$

WCRT OF TASK τ_i By respecting theorem 10, and assuming tasks are synchronous, the completion time w_i of the p^{th} job of τ_i is composed of two parts:

– Workload of jobs of τ_i upto the p^{th} job
– Workload of jobs of tasks in hp_i in the busy period of τ_i

The completion time of job p of τ_i is also computed as:

$$\begin{aligned} &\forall 0 < p < Q, w_i^{(0)}(p) = p \cdot C_i \\ &\forall 0 < p < Q, \forall k > 0, w_i^{(k)}(K) = p \cdot C_i + \sum_{\tau_j \in hp_i} \left(\left\lceil \frac{w_i^{(k-1)}}{T_j} \right\rceil \cdot C_j \right) \end{aligned} \tag{16}$$

where Q is the number of jobs of τ_i that occur in its busy period of length L_i.
Since the p^{th} job of τ_i is released at $(p-1) \cdot T_i$, its response time is:

$$\forall 0 < p < Q, R_i(p) = w_i(p) - (p-1) \cdot T_i \tag{17}$$

After computing response times of jobs 1 to Q of τ_i, the WCRT R_i of τ_i is:

$$R_i = \max_{p \in [1;Q]} R_i(K) \tag{18}$$

The WCRT of τ_i is compared to its deadline D_i to determine if the task is schedulable. Under RM or DM scheduling, only the response time of the first job of τ_i is computed, which simplifies the schedulability condition of a task:

Theorem 11. *[64] A synchronous independent periodic task τ_i, executing on an uniprocessor, scheduled by RM or DM, with $D_i \leqslant T_i$, is schedulable if the following iterative equation is satisfied:*

$$\begin{aligned} &R_i^{(0)} = C_i \\ &\forall k > 0, R_i^{(k)} = C_i + \sum_{\tau_j \in hp_i} \left(\left\lceil \frac{R_i^{(k-1)}}{T_j} \right\rceil \cdot C_j \right) \leqslant D_i \end{aligned} \tag{19}$$

EFFECTS OF WCBT AND JITTER To extend the RTA for shared resources, protected by a protocol like PCP, the authors in [127] proved that the WCBT B_i of a task τ_i is simply added to Equation 16:

$$\forall 0 < p < Q, \forall k > 0, w_i^{(k>0)}(p) = p \cdot C_i + B_i + \sum_{\tau_j \in hp_i} \left(\left\lceil \frac{w_i^{(k-1)}}{T_j} \right\rceil \cdot C_j \right) \tag{20}$$

If tasks may experience release jitter J_i, Equation 20 can be extended according to [4]:

$$\forall 0 < p < Q, \forall k > 0, w_i^{(k)}(p) = p \cdot C_i + B_i + \sum_{\tau_j \in hp_i} \left(\left\lceil \frac{J_j + w_i^{(k-1)}}{T_j} \right\rceil \cdot C_j \right) \tag{21}$$

Equation 17 is also extended:

$$\forall 0 < p < Q, R_i(p) = w_i(p) + J_i - (p-1) \cdot T_i \tag{22}$$

In this section, a schedulability test that computes response times was exposed. In the next section, a feasibility test that computes demand bounds is presented.

2.3.4 *Processor Demand Feasibility Tests*

Processor demand based tests also compute workloads of the system. Contrary to RTA based tests, tests that compute processor demand will determine the general schedulability of the system rather than the schedulability of each individual task by computing its WCRT. The schedulability of a tasks set, scheduled by an optimal policy, is used to determine its feasibility.

The following paragraphs show how to determine the feasibility of tasks scheduled by a preemptive scheduling policy on a uniprocessor. The tasks are assumed independent. Assessing if a preemptive scheduling policy can schedule independent tasks, with $D_i \leqslant T_i$, is done by assessing the schedulability of tasks set under EDF.

Let us now focus on a processor demand based schedulability test for EDF. The function that computes processor demand by jobs of n tasks released in $[t1; t2]$, with their deadlines in $[t1; t2]$, is denoted by $\mathtt{dbf}(t1, t2)$ and computed in [16, 17]:

$$\mathtt{dbf}(t1, t2) = \sum_{i=1}^{n} n_i(t1, t2) \cdot C_i \tag{23}$$

where $n_i(t1, t2)$ is the number of jobs of task τ_i released in $[t1; t2]$, and with their absolute deadline in $[t1; t2]$.

Since the tasks set is scheduled by EDF, the $\mathtt{dbf}(t1, t2)$ function gives the amount of workloads that must be completed in $[t1; t2]$ for the tasks set to be schedulable. A necessary and sufficient condition of schedulability of tasks scheduled by EDF is then [17]:

$$\mathtt{dbf}(t1, t2) \leqslant t2 - t1 \tag{24}$$

The condition in Equation 24 means all intervals $[t1; t2]$ must be checked, which is impossible since there is an infinite number of intervals. We must thus find a sufficient interval $[t1; t2]$ in which to assess schedulability. A sufficient interval can be found due to the periodic behavior of the system. Assuming that tasks are released at $t = 0$ (synchronous system) and the processor utilization is $U < 1$, in [17] the authors proved that the schedulability can be assessed in an interval $[0; t]$ bounded by the longest busy period of the processor. The bound of the interval, denoted L, is computed as follows:

$$L = \frac{U}{1-U} \cdot \max_{i \in [1;n]} (T_i - D_i) \tag{25}$$

The condition in Equation 24 then becomes:

$$\forall L > 0, \mathtt{dbf}(0, L) \leqslant L \tag{26}$$

To compute $\mathtt{dbf}(0, L)$, the number $n_i(0, L)$ of jobs of each task τ_i released in $[0; L]$, and with their absolute deadlines in $[0; L]$, must be computed. In [17], $n_i(0, L)$ is computed as:

$$\forall L > 0, n_i(0, L) = \max\left(0, \left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1\right) \tag{27}$$

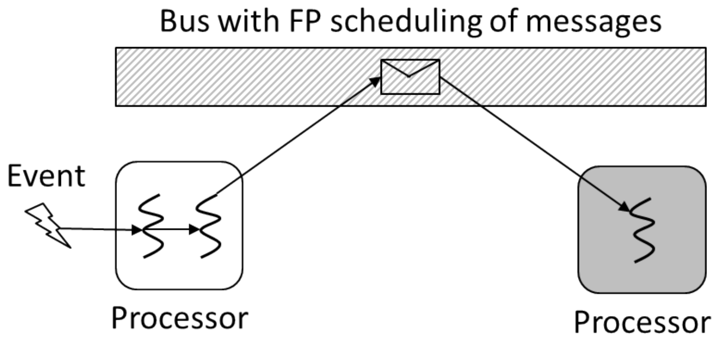


Figure 2.3: Distributed System

The value of $\mathtt{dbf}(0, L)$ is then the sum of processor demands by jobs of each task τ_i released in $[0; L]$, and with their absolute deadlines in $[0; L]$:

$$\forall L > 0, \mathtt{dbf}(0, L) = \sum_{i=1}^{n} \max\left(0, \left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1\right) \cdot C_i \tag{28}$$

If the condition of Equation 26 is respected, then at least EDF can schedule the synchronous independent tasks set, with $D_i \leqslant T_i$. The tasks set is then feasible.

The next section presents a model where the periodic task model is generalized by a model called transaction.

2.4 TRANSACTION MODELS: ILLUSTRATION WITH TREE-SHAPED

Most of the results, presented for the periodic task model, assume synchronous tasks. To extend the analysis to asynchronous tasks, where tasks releases are delayed, the periodic task model was extended with offsets [143, 56, 109], resulting in the transaction model [143].

Figure 2.3 shows an example of a distributed system where tasks are not all released at the same time due to precedence dependencies. In the system, a task on the left processor releases another task also on the left processor. This task then sends a message, through a bus, to a task on the right processor.

There are end-to-end flows in such distributed systems. An end-to-end flow starts with the release of an entity to the completion of execution of another entity. For example in Figure 2.3, an end-to-end flow starts with the release of the first task in the left processor, to the completion of the last task in the right processor.

In this section the transaction model is first defined. Afterwards a historical perspective of schedulability tests for transactions will be exposed. Finally a test that exploits precedence dependencies between tasks in transactions, will be presented in detail.

2.4.1 *Definition*

According to [141], "a transaction is a group of related tasks (related either through some collectively performed function, or through some shared timing attributes whereby it is convenient to collect these tasks into a single entity)". In [105], transactions are used to model groups of tasks related by precedence dependency. Let us see some definitions and notations for the transaction model, taken from [141, 143, 105, 106, 118].

A transaction is denoted by Γ_i and its tasks are denoted by τ_{ij}. A transaction is released by a periodic event that occurs every T_i. A particular instance of a transaction is called a job. A job of a task in a transaction corresponds to a job of the transaction. If the event that releases the p^{th} job of Γ_i occurs at t_0, then the p^{th} jobs of its tasks are released after or at t_0. The release time of the first job of Γ_i is denoted by r_i. A task τ_{ij} has the following parameters:

- C_{ij} is the WCET.
- C^b_{ij} is the BCET.
- O_{ij} is the offset, a minimum time that must elapse after the release of the job of Γ_i before τ_{ij} is released [105]. Otherwise said τ_{ij} is released at least O_{ij} units of time after t_0. Value $r_{ij} = r_i + O_{ij}$ is the absolute release time of the first job of τ_{ij}.
- d_{ij} is the relative deadline. Value $O_{ij} + d_{ij}$ is the global deadline [106] of τ_{ij}. Value $r_i + O_{ij} + d_{ij}$ is the absolute deadline of the first job of τ_{ij}.
- J_{ij} is the maximum jitter, i.e. τ_{ij} is released in $[t_0 + O_{ij}; t_0 + O_{ij} + J_{ij}]$.
- B_{ij} is the WCBT [127].
- $\mathtt{prio}(\tau_{ij})$ is the fixed priority.
- $\mathtt{proc}(\tau_{ij})$ is the processor on which τ_{ij} is allocated on.
- R^w_{ij} is the global WCRT, which is the WCRT relative to the release of the transaction [105]. A global response time is the response time of a task plus its offset. As a reminder, a response time of a task is relative to its release, in this case its offset.
- R^b_{ij} is the global BCRT, which is the BCRT relative to the release of the transaction [105].

Tasks may use shared resources in critical sections. A critical section is denoted by (τ, R, S, B) where τ is the task using the resource R. Task τ asks for R at S of its execution time, and locks it during the next B of its execution time.

Tasks in a transaction are related by precedence dependencies [106]. A precedence dependency between two tasks is denoted by $\tau_{ip} \prec \tau_{ij}$. As a reminder, the precedence dependency means that a job p of τ_{ip} must complete before a job p of τ_{ij} can be released. τ_{ip} (resp. τ_{ij}) is called the predecessor (resp. successor) of τ_{ij} (resp. τ_{ip}). According to the precedence dependencies that may exist between tasks, transactions are of different type.

Definition 52 (Linear Transaction [52]). *In the linear transaction model, each event that releases a transaction, generates a response in the form of a sequence of actions of the transaction. Each action is released by the event generated by the previous action. An action may be a task (or a part of a task) or a message sent through an interconnect. Normally, the first action of a transaction is a task. Actions can only be released from a single event, and can only generate one event that may in turn release another single action.*

Definition 53 (Tree-Shaped Transaction [118]). *A tree-shaped transaction Γ_i has a root task, denoted by τ_{i1}, which leads to the releases of all other tasks, upon completion. A task τ_{ij}, of a tree-shaped transaction, is said to have at most one immediate predecessor (denoted by $\mathtt{pred}(\tau_{ij})$) that releases it upon completion. A task τ_{ij} may have several immediate successors (denoted by $\mathtt{succ}(\tau_{ij})$) that it releases upon completion. The root task τ_{i1} has not predecessor.*

Definition 54 (Graph-Shaped Transaction [66]). *A graph-shaped transaction is a generalization of a tree-shaped transaction, with less constraints. A graph-shaped transaction does not have an unique root task. A task τ_{ij}, of a graph-shaped transaction, has a set of predecessors, denoted by $\mathtt{pred}(\tau_{ij})$, and thus it can have more than one immediate predecessor.*

Note that a linear transaction is a more constrained tree-shaped transaction where a task can have at most one immediate successor, instead of several. On the other hand, a graph-shaped transaction is less constrained than a tree-shaped transaction.

Figure 2.4 shows some examples of the different types of transactions defined above. Figure 2.4a) is a linear transaction, Figure 2.4b) is a tree-shaped transaction, and Figure 2.4c) is a graph-shaped transaction.

Generally an analyzed task is denoted by τ_{ab}, and it belongs to a transaction denoted by Γ_a. The set $\mathtt{hp}_i(\tau_{ab})$ is the set of tasks in transaction Γ_i with a priority higher than or equal to $\mathtt{prio}(\tau_{ab})$ and allocated on the same processor as τ_{ab}. A respective definition is given for lower priority tasks

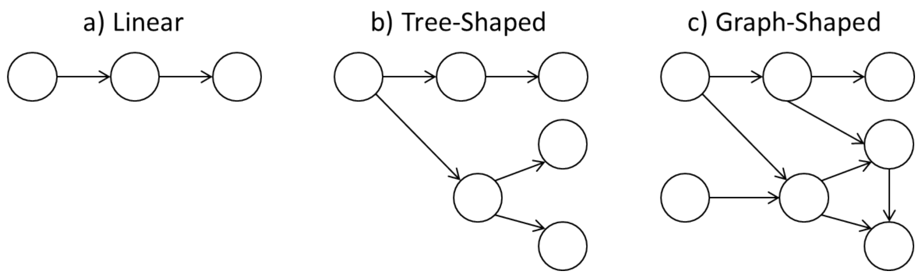


Figure 2.4: Transaction Types: Circles are tasks; Arrows are precedence dependencies

$lp_i(\tau_{ab})$. For readability issues, when analyzing τ_{ab}, sometimes $hp_i(\tau_{ab})$ is denoted by hp_i, and similarly for any other notation that has the τ_{ab} parameter.

The following section gives a historical perspective on the different schedulability tests for different types of transactions.

2.4.2 *Offset-Based Response Time Analysis: A Historical Perspective*

Schedulability tests for transactions use the offset-based RTA method to compute response times. For example, it can compute the WCRTs of tasks.

Figure 2.5 shows a timeline of some schedulability tests for transactions based on their RTA. Most of the tests in Figure 2.5 contribute to extend previous tests focusing on several improvements:

– Lesser response time pessimism
– Faster analysis time
– Applicability to EDF scheduling
– Applicability to new types of transactions (e.g. tree-shaped)

The following sections describe the contributions of each test in Figure 2.5.

2.4.2.1 *Fundamental Tests*

Schedulability test for transactions based on RTA was first proposed by Tindell et al. [143], where the authors use the linear transaction model. This test improves previous tests (e.g. [64]), in terms of response time pessimism, when applied to systems where tasks are asynchronous.

Although precedence dependencies between tasks of a same transaction are implicitly modeled, the authors use static offsets, i.e. a fixed value for the offset. Precedence dependencies aren't fully specified [118], nor exploited. Furthermore the transaction model proposed by Tindell restricts the parameters of its tasks: a task is not allowed to have a deadline, offset, or jitter higher than the period of the transaction.

In [105] Palencia et al. generalize Tindell's work, for systems where task offset, jitter and deadline may be higher than their period so several jobs of tasks in a linear transaction may interfere. The authors also introduce dynamic offset to fully model precedence dependency between tasks. A dynamic offset is an offset whose value is within an interval. Indeed, the offset of a task depends on response time of its preceding task. Their test, called WCDO, computes an upper-bound to the WCRT of a task.

Both [143] and [105] propose a method to compute the exact and approximate response times of tasks. The exact response times are computed in exponential time, while the approximation is computed in pseudo-polynomial time. For the large systems with many tasks, the computation of exact response times is intractable [143, 105]. Thus the approximation is necessary in this case.

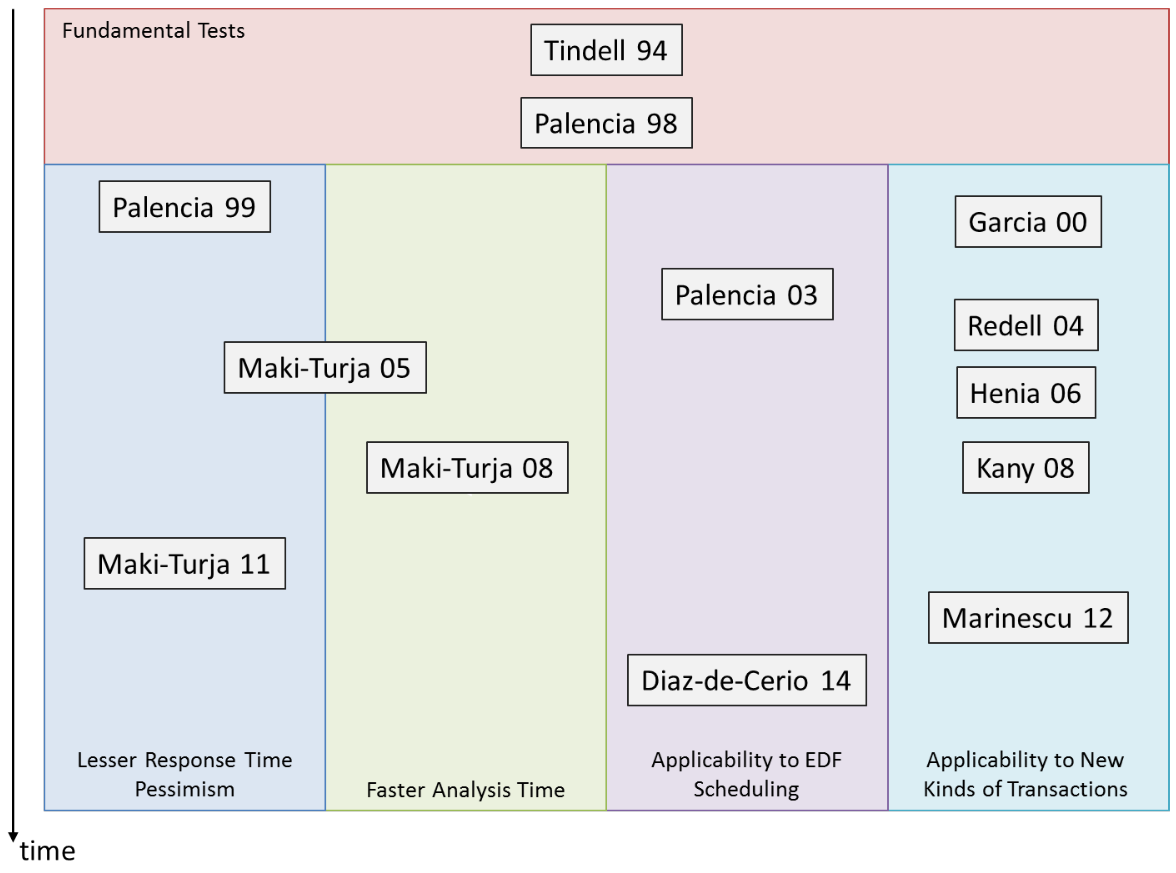


Figure 2.5: Schedulability Tests Based on RTA for Transactions

2.4.2.2 *Lesser Response Time Pessimism*

Having proposed dynamic offsets and jitters to fully model precedence dependencies, Palencia et al. proposed an improvement of the WCDO test, called WCDOPS in [106]. This test gives less pessimistic WCRTs than the WCDO test, for systems where tasks are released immediately after the preceding one. The authors exploit the precedence dependencies between tasks of different priorities to identify scenarios where not all higher priority tasks may interfere an analyzed task.

In [90, 91], Mäki-Turja et al. observe that previous RTA methods introduce pessimism in the response time computation due to the computation of interference.

Take the example of Equation 13 where we have the term $\left\lceil \frac{t}{T_j} \right\rceil \cdot C_j$. Let us assume that $C_j = 3$, $T_j = 4$, and we have an interval size of $t = 5$. Due to the ceiling operation, we would get an interference of $2 \times C_j = 6$ for a task τ_j in $[0; 5]$, although the interference contained within $[0; 5]$ cannot be higher than the length of the interval, i.e. 5. The actual interference in $[0; 5]$ is 4.

The authors thus exploit the fact that that the interference in an interval cannot be higher than the length of the interval. Their approach for computing interference is thus equivalent to float addition, while previous methods are equivalent to interger addition. Their method reduces the pessimism of computed WCRTs.

Later, the authors also propose in [93] to compute interference in systems where tasks have some kind of dependency between their execution times. One task executing at its WCET may mean that another does not execute at its WCET. Thus the WCET value is not always considered for the computation of interference.

2.4.2.3 *Faster Analysis Time*

The analysis time taken by schedulability tests based on RTA is also a factor that can be improved. In [89, 91, 92], Mäki-Turja et al. propose an efficient implementation of the fundamental schedulability tests based on RTA proposed by [143, 105]. Their implementation computes response times faster, thus making the analysis time shorter.

Their idea comes from the observation that the interference from a transaction to a task shows a periodic pattern. Therefore instead of computing the interference of a given transaction to a given task, whenever this value is needed, the interference of the transaction to the task, in a given period, is computed beforehand and stored in a table. Each time the interference of the transaction is needed, a lookup function checks the value in the table and sums it accordingly by the number of jobs of the transaction that may interfere. The interference from each transaction to each task is thus stored beforehand to make the analysis time faster.

2.4.2.4 *Applicability to EDF Scheduling*

All of the previous tests are applicable to a system with FP scheduling. Some tests have also been proposed for systems scheduled by EDF. In [107], the WCDO is adapted for global clock EDF (GC-EDF). In a system scheduled by GC-EDF, there is a global clock synchronized between all processors. In [44], the authors adapt the WCDO and WCDOPS tests for local clock EDF (LC-EDF). A system scheduled by LC-EDF is one where the clocks of each processor is not synchronized.

2.4.2.5 *Applicability to New Types of Transactions*

All previous tests are only applicable to linear transactions. There exists tests for tree-shaped and graph-shaped transactions.

Early works by Garcia et al. in [52] already propose to analyze transactions with multiple-event synchronization: a task may have several predecessors and several successors. Among N predecessors, a task may wait for 1-among-N to complete execution, or all-N to complete execution, before being released. Upon completion, a task may release 1-among-N successors or all-N successors.

The approach of the authors is to transform these synchronization semantics into linear transactions. The authors state in [52] that the analysis may be pessimist, because of the transformation to linear transactions.

Later tests consider directly the type of the transactions to analyze. The WCDOPS+ test proposed by Redell in [118], adapts the original WCDOPS test for tree-shaped transactions, with tasks scheduled by FP. It also reduces further the pessimism of WCDOPS by further eliminating interferences that are not possible, due to precedence dependencies and how priorities are specified.

Response times computed by the WCDOPS+ test are improved in [58], where the concept of relative offsets is proposed. Previously offsets were relative to a single event, which is the release of the transaction. With a relative offset, the offset value may be relative to any event, for example the release of any task in the transaction instead of the transaction only. In [58], when a task τ_{ij} has a predecessor τ_{ip} that releases several tasks, an offset is specified for τ_{ij} relative to the release of τ_{ip}. This allows the authors to further eliminate interferences that are not possible.

WCDOPS+ is adapted in [95] for time partitioned systems, where some tasks can only execute within a given time slot but must complete before the end of the time slot. In [66], WCDOPS+ is adapted for graph-shaped transactions.

2.4.3 *Basic Concepts of RTA of Transactions*

As a reminder, the RTA schedulability test for synchronous periodic tasks scheduled by RM or DM, considers that there is only one critical instant for the whole system, when tasks are all released at the same time. Furthermore, because of RM or DM scheduling, only the response time of the first job of a task needs to be computed to assess schedulability. Finally RTA tests for RM and DM assume tasks do not have precedence dependencies, so the impact of this kind of dependency on response times is not considered.

These three assumptions do not hold true in the transaction model, where tasks have precedence dependency, and are released with an offset. Thus several issues are raised:

– Tasks have precedence dependency which must be modeled with the task parameters, and considered by the RTA.
– There are several potential critical instants, and thus several potential worst case scenarios that lead to the WCRT of a task.
– Due to precedence dependencies and how task priorities are specified, not all higher priority tasks may be part of a same busy period of an analyzed task.

In the following sections, let us first see how these issues are solved by the tests in [143, 105, 106, 118]. Throughout these sections, examples are illustrated with the sub-figures in Figure 2.6. Three jobs of tasks of a transaction Γ_i are shown in Figure 2.6a). In Γ_i there are three tasks: $\tau_{i1} \prec \tau_{i2} \prec \tau_{i3}$. The analyzed task is denoted by τ_{ab}.

2.4.3.1 *Dynamic Offset*

As stated before, an issue is that tasks have precedence dependency which must be modeled with the task parameters, and considered by the RTA.

Consider two tasks with a precedence dependency. The successor task is released after the completion of the predecessor task. Therefore the earliest release time of the successor is the earliest completion of the predecessor. The offset is a parameter to specify the earliest release of a task. This parameter is thus used to model precedence dependency.

According to [106], when a task is released with an offset that is constant, independent of the execution of other tasks in the system, the offset is said to be static. An offset is said to be dynamic if it can vary between some minimum and maximum interval: $[O_{ij}^{min}; O_{ij}^{max}]$. This variation is due to the completion of other tasks for which the released task must wait.

Dynamic offsets are necessary to model precedence dependent tasks. To analyze tasks with dynamic offsets, the authors in [105] propose to transform the concept of dynamic offset into static

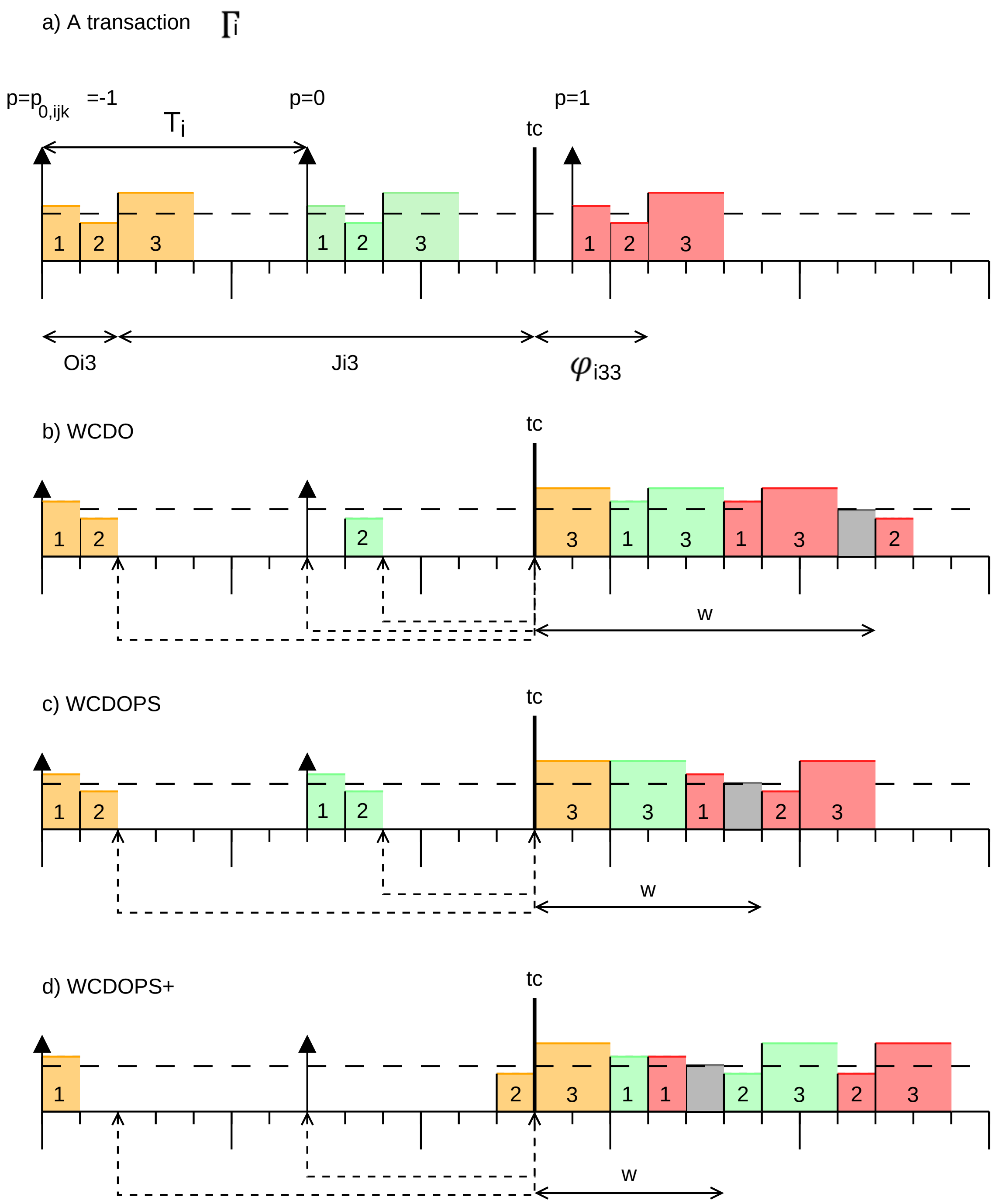


Figure 2.6: Transactions and Interference: Tasks are boxes with height proportional to their priority; A numbered box indicates a task index, i.e. $1 = \tau_{i1}$, $2 = \tau_{i2}$, $3 = \tau_{i3}$; Grey box with no name is analyzed task τ_{ab}; Dashed line is priority level of τ_{ab}; Dotted double arrows are jitters; Up arrows are transaction releases; All tasks are on same processor

offset combined with jitter. The lower bound of a dynamic offset is the the static offset: $O_{ij} = O_{ij}^{min}$. The difference between the lower and upper bounds of the dynamic offset is the jitter: $J_{ij} = O_{ij}^{max} - O_{ij}^{max}$.

When dynamic offsets are used to model precedence dependent tasks, the lower and upper bounds of the interval are defined by the response times of tasks. Thus when transformed to static offsets and jitters, the values of offset and jitter are also defined by the response times of tasks. Let us have $\tau_{ip} = pred(\tau_{ij})$. The parameters of offset and jitter are then:

$$O_{ij} = R_{ip}^{b} \tag{29}$$

$$R_{ij}^{b} = O_{ij} + C_{ij}^{b} \tag{30}$$

$$J_{ij} = R_{ip}^{w} - R_{ip}^{b} = R_{ip}^{w} - O_{ij} \tag{31}$$

$$R_{ip}^{w} = O_{ip} + C_{ip}^{w} \tag{32}$$

The offset O_{ij} of τ_{ij} is the global BCRT of its immediate predecessor τ_{ip} (Equation 30). The BCRT of a task τ_{ij} is its offset plus its BCET (Equation 30). The jitter of a task τ_{ij} is the difference between the WCRT of τ_{ip} and the BCRT of τ_{ip}. The BCRT of τ_{ip} is also the offset of τ_{ij} (Equation 31). Initially the WCRT of a task τ_{ip} is its offset plus its WCET (Equation 32).

To illustrate dynamic offsets and jitters, let us take the example of the transaction Γ_i in Figure 2.6a) and 2.6b). We have $C_{i1} = C_{i2} = 1$. For readability issues, let us assume that the BCETs are equal to the WCETs, i.e. $C_{i1}^{b} = C_{i2}^{b} = 1$. As indicated by Figure 2.6a), the offset of τ_{i3} is:

$$\begin{aligned} O_{i3} &= R_{i2}^{b} \\ &= O_{i2} + C_{i2}^{b} \\ &= R_{i1}^{b} + C_{i2}^{b} \\ &= C_{i1}^{b} + C_{i2}^{b} \quad = 2 \end{aligned} \tag{33}$$

In Equation 33, notice that the offset of τ_{i3} is the sum of WCETs of its predecessors. Now let us assume the WCRT of τ_{i2} is $R_{i2}^{w} = 13$ due to some interference from other transactions in the system. In Figure 2.6a), if $R_{i2}^{w} = 13$, the jitter of τ_{i3} is:

$$\begin{aligned} J_{i3} &= R_{i2}^{w} - O_{i3} \\ &= 13 - 2 \\ &= 11 \end{aligned} \tag{34}$$

Jitters are updated as response times are computed. Furthermore during the analysis, the response time of a task is computed by considering the jitter of the task. Therefore we have response times that depend on jitters and jitters that depend on response times. The analysis thus iterates until a convergence of values is reached [143].

2.4.3.2 *Worst Case Scenario*

As stated before, an issue is that there are several potential critical instants, and thus several potential worst case scenarios that lead to the WCRT of a task. The busy period of a task and a critical instant are defined in Definitions 49 and 50. The length of the busy period is denoted by w and the critical instant is denoted by t_c.

For example in Figure 2.6b), the busy period starts at t_c. 2 jobs of task τ_{i1} and 3 jobs of task τ_{i3} are in the busy period because they are of higher priority than τ_{ab} (the analyzed task), itself also in the busy period. The length of the busy period is then $w = 2C_{i1} + 3C_{i3} + C_{ab} = 9$.

In the worst case scenario, the interference of transactions to a task τ_{ab} is maximum. The problem of determining the critical instant that leads to the worst case scenario is the problem of determining how all transactions are released according to a critical instant t_c. Otherwise said, we must determine how transactions are phased according to t_c.

For example in Figure 2.6a), transaction Γ_i is phased such that its first job that occurs after the critical instant t_c is released at $t = t_c + 1$. In [105] the authors showed the following theorem:

Theorem 12. *[105] The maximum interference from a transaction Γ_i to a task τ_{ab} (i.e. maximum contribution to a τ_{ab} busy period) occurs when the release of Γ_i is phased such that some task $\tau_{ik} \in hp_i$ is released at a critical instant t_c after having experienced its maximum release jitter J_{ik}.*

We say that τ_{ik} starts the τ_{ab} busy period and we created a scenario. After creating a scenario, without lose of generality, it is also assumed that the critical instant $t_c = 0$ [105]. Jobs of tasks in hp_i before t_c must experience enough jitter to be released at t_c, and jobs of tasks in hp_i after t_c must not experience jitter to be released as early as possible.

When τ_{ik} starts the τ_{ab} busy period, not only can several jobs of Γ_i interfere τ_{ab} but several jobs of τ_{ab} may need to be analyzed. The phasing of jobs of Γ_i or Γ_a can be determined. Figure 2.6b) shows parameters of the phasing of jobs of Γ_i.

A job number p is assigned to a job of Γ_i according to the job's release time. Jobs $p \leqslant 0$ are released before or at t_c and jobs $p > 0$ are released after t_c. For a particular task $\tau_{ij} \in \Gamma_i$, the first job $p = 1$ after t_c is released at φ_{ijk} [105]:

$$\varphi_{ijk} = T_i + O_{ij} - (O_{ik} + J_{ik}) \bmod T_i \tag{35}$$

The number of pending jobs of τ_{ij} at t_c is [105]:

$$n_{ijk} = \left\lfloor \frac{J_{ij} + \varphi_{ijk}}{T_i} \right\rfloor \tag{36}$$

Since the last pending job of τ_{ij} at t_c is numbered $p = 0$, the first pending job of τ_{ij} at t_c is numbered $p_{0,ijk}$ [105]:

$$p_{0,ijk} = 1 - \left\lfloor \frac{J_{ij} + \varphi_{ijk}}{T_i} \right\rfloor \tag{37}$$

The number of jobs of τ_{ij} after t_c that are released within a τ_{ab} busy period of length w is [105]:

$$\left\lceil \frac{w - \varphi_{ijk}}{T_i} \right\rceil \tag{38}$$

For example in Figure 2.6b), τ_{i3} starts the τ_{ab} busy period after having experienced its jitter J_{i3}. Γ_i is then phased such that task τ_{i3} is released at $\varphi_{i33} = 3$. For task τ_{i1} (resp. τ_{i2}), this value is $\varphi_{i13} = 1$ (resp. $\varphi_{i23} = 2$). We have 3 jobs of τ_{i3} in the τ_{ab} busy period: 2 jobs before t_c and one after t_c. The first pending job of τ_{i3} at t_c is thus numbered $p_{0,i33} = -1$. For task τ_{i1} (resp. τ_{21}), this value is $p_{0,i13} = 0$ (resp. $p_{0,i23} = 0$).

Similarly values representing the phasing of Γ_a, to which the analyzed task τ_{ab} belongs to, can also be computed.

Once a scenario is created, the goal is to compute interference of transactions Γ_i, and transaction Γ_a, to τ_{ab}. When computing interference, Γ_i and Γ_a contributes to the τ_{ab} busy period. The interference contributes to the WCRT of τ_{ab}.

To create the scenario that leads to the exact WCRT of a task, all combinations of tasks τ_{ik} in all transactions must be tested [143, 105]. This operation is exponential and intractable for large systems with many tasks [143, 105]. Thus tests for transactions usually compute an upper-bound to the WCRT of a task by considering each transaction contributes its maximum to the busy period of τ_{ab}, without knowledge of how other transactions interfere τ_{ab}.

2.4.3.3 *Execution Conflicts*

As stated before, an issue is that due to precedence dependencies and how task priorities are specified, not all higher priority tasks may be part of a same busy period of an analyzed task.

When analyzing a task τ_{ab}, not all tasks in hp_i are eligible to execute within the same τ_{ab} busy period. This is called an execution conflict. Figure 2.6 illustrates the consequences of execution conflicts.

Figure 2.6a) shows the execution of the three jobs of Γ_i alone. Two jobs are released before t_c, and one job is released after. Figures 2.6b) to 2.6d) show how Γ_i contributes to the τ_{ab} busy period, according to different tests, in the scenario where τ_{i3} starts the τ_{ab} busy period.

Figure 2.6b) shows the interference computed by WCDO [105]. The test does not consider execution conflicts so any jobs of $\tau_{ij} \in hp_i(\tau_{ab})$, that experiences enough jitter to be released at t_c, or that is released after t_c in the τ_{ab} busy period, can contribute to the τ_{ab} busy period. Therefore the two jobs of τ_{i1} and three jobs of τ_{i3} contribute to the τ_{ab} busy period.

Figure 2.6c) shows the interference computed by WCDOPS [106]. The test reduces pessimism by solving an execution conflict. Since tasks of Γ_i are related by precedence dependency, and $\tau_{i2} \in lp_i(\tau_{ab})$ must complete before the release of τ_{i3}, tasks τ_{i1} and τ_{i3} cannot be in the same τ_{ab} busy period. For the second job of Γ_i, τ_{i3} is chosen to contribute to the busy period because its WCET is longer than the WCET of τ_{i1}. For the third job of Γ_i released at t_c, only τ_{i1} can contribute to the busy period because τ_{i2} cannot execute in the busy period and thus τ_{i3} cannot be released before the end of the busy period.

Figure 2.6d) shows the interference computed by WCDOPS+ [118]. It also shows the real schedule of the scenario, where τ_{i3} experiences jitter to be released at t_c, and thus τ_{i2} completes at t_c. In Figure 2.6d), WCDOPS+ further reduces pessimism by solving another execution conflict. Since the first job of τ_{i3} starts the τ_{ab} busy period, it must be in the busy period. Once the busy period starts, the second job of τ_{i2} cannot be in the busy period. Neither can the second job of τ_{i3} since it is preceded by τ_{i2}. Thus for the second job of Γ_i, only task τ_{i1} can contribute to the busy period.

The next section presents the WCDOPS+ test, where the basic concepts, presented in this section, are all considered.

2.4.4 *The WCDOPS+ Test*

This section presents the schedulability test in [118] for tree-shaped transactions. In the following sections, first an overview of the RTA method of the test is given. Afterwards key steps of the method are exposed.

2.4.4.1 *Overview of the RTA Method*

In this section, an overview of the analysis is shown. Then the approach of the analysis is compared to the WCRT computation approach for the periodic task model. As we will see, there are similarities and differences.

Figure 2.7 show the general algorithm of WCDOPS+, for the analysis of a given task τ_{ab}, during an iteration of the algorithm. The approach is based on [143, 105].

Some tasks sets are first defined to solve execution conflicts **(Op0)**. The idea is to compute the WCRT of τ_{ab} for each scenario where a task τ_{ac} starts the τ_{ab} busy period **(Op1)**. Within a scenario, the WCRT of each job p_{ab} of τ_{ab} in the τ_{ab} busy period is computed. The length of the busy period w can be estimated [105]. To compute the WCRT of job p_{ab} of τ_{ab}, interference from transactions of the system is computed **(Op2)**. The interference includes those from transactions Γ_i and transaction Γ_a to which the analyzed task τ_{ab} belongs to. The WCRT of τ_{ab} is then the maximum of WCRTs of each job p_{ab} of each scenario **(Op3)**.

Once the WCRT of each task τ_{ab} is computed, jitters are updated. Convergence is then checked and if any values are modified, the test goes on to the next iteration.

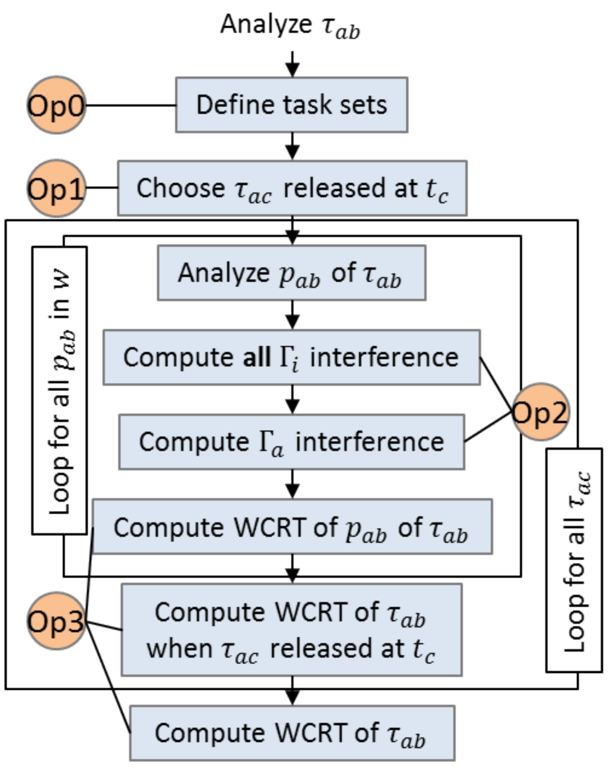

Analyze τ_{ab}
Op0
Define task sets
Op1
Choose τ_{ac} released at t_c
Loop for all p_{ab} in w
Analyze p_{ab} of τ_{ab}
Compute **all** Γ_i interference
Op2
Compute Γ_a interference
Compute WCRT of p_{ab} of τ_{ab}
Loop for all τ_{ac}
Op3
Compute WCRT of τ_{ab} when τ_{ac} released at t_c
Compute WCRT of τ_{ab}

Figure 2.7: WCDOPS+ Overview: Circles indicate key operations

The approach is similar to the RTA method for the periodic task model. Indeed, the WCRT of a periodic task is computed by first computing the length of its busy period, then the number of jobs in its busy period, then the response times of each job to get the maximum.

The approaches are similar but there are some major differences, due the basic concepts of RTA of transactions. First, there are several scenarios to consider. This was explained previously in Section 2.4.3.2. Second, the interference computation is different due to precedence dependencies. This was explained previously in Section 2.4.3.3 and will be fully exposed in later sections, since the interference computation is the bulk of the WCDOPS+ test. Third, jitters are updated after the WCRT computation and the whole algorithm iterates if jitters are modified. This was explained previously in Section 2.4.3.1.

All of these concepts, and others, are part of the key operations of the WCDOPS+ algorithm. These key operations are illustrated in Figure 2.7. In the following sections each key operation is explained. To illustrate the operations, the example of the tree-shaped transaction in Figure 2.8 will be referred to.

2.4.4.2 *Op0: Task Sets and Execution Conflicts*

To identify and solve execution conflicts, it is useful to group tasks of a transaction in sets according to their priorities and on which processor they are allocated on [106, 118]. This is done in **Op0**.

H SEGMENT AND H SECTION In [118], two kinds of sets are defined: H segments and H sections.

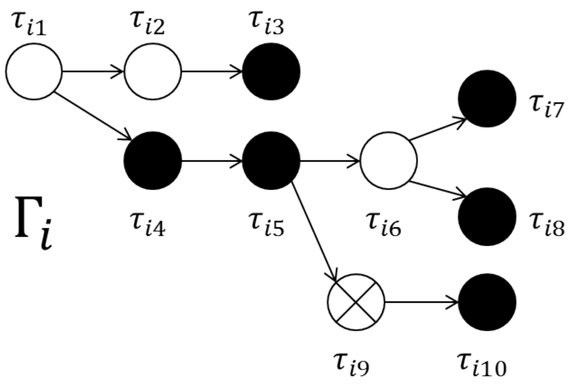


Figure 2.8: Tree-Shaped Transaction: Task τ_{ab} assumed under analysis; Tasks belong to Γ_i; Black circles are tasks in $hp_i(\tau_{ab})$; White circles are tasks in $lp_i(\tau_{ab})$; Crossed circles are tasks on another processor

An H segment is a set of tasks that must all execute within a same τ_{ab} busy period. Two tasks in $hp_i(\tau_{ab})$ belong to the same H segment if there is no other task that is not in hp_i that precedes one but not the other. Formally an H segment is defined as:

$$H^{seg}_{ij}(\tau_{ab}) = \{\tau_{im} \mid \tau_{im} \in hp_i(\tau_{ab}) \wedge (\neg\exists \tau_{il} \in \Gamma_{ij}\Delta\Gamma_{im} \mid \tau_{il} \notin hp_i(\tau_{ab})\} \tag{39}$$

where Γ_{ij} is defined as:

$$\Gamma_{ij} = \{\tau_{ik} \in \Gamma_i \mid \tau_{ik} \prec \tau_{ij}\} \cup \{\tau_{ij}\} \tag{40}$$

and $\Gamma_{ij}\Delta\Gamma_{im}$ is defined as:

$$\Gamma_{ij}\Delta\Gamma_{im} \equiv \Gamma_{ij} \cup \Gamma_{im} \setminus \Gamma_{ij} \cap \Gamma_{im} \tag{41}$$

To illustrate H segments, let us see some examples in Figure 2.8. We have $H^{seg}_{i3} = \{\tau_{i3}\}$, $H^{seg}_{i4} = H^{seg}_{i5} = \{\tau_{i4}, \tau_{i5}\}$, $H^{seg}_{i10} = \{\tau_{i10}\}$, and $H^{seg}_{i7} = H^{seg}_{i8} = \{\tau_{i7}, \tau_{i8}\}$.

An H section is a set of tasks that may execute in the same τ_{ab} busy period. Two tasks in $hp_i(\tau_{ab})$ belong to the same H section if there is no other task in $lp_i(\tau_{ab})$ that precedes one but not the other. Formally an H section is defined as:

$$H_{ij}(\tau_{ab}) = \{\tau_{im} \mid \tau_{im} \in hp_i(\tau_{ab}) \wedge (\neg\exists \tau_{il} \in \Gamma_{ij}\Delta\Gamma_{im} \mid \tau_{il} \in lp_i(\tau_{ab})\} \tag{42}$$

To illustrate H sections let us see some examples in Figure 2.8. We have $H_{i4} = H_{i5} = H_{i10} = \{\tau_{i4}, \tau_{i5}, \tau_{i10}\}$.

PRECEDENCE DEPENDENCY OF H SEGMENT A task τ_{ik} is said to precede an H segment H^{seg}_{ij}, denoted by $\tau_{ik} \prec H^{seg}_{ij}$, if it precedes all tasks in H^{seg}_{ij}. The immediate predecessor of an H segment is denoted by $pred(H^{seg}_{ij})$ and satisfies the following condition:

$$\tau_{ip} \prec H^{seg}_{ij} \wedge (\exists \tau_{im} \in H^{seg}_{ij} \mid pred(\tau_{im}) = \tau_{ip} \tag{43}$$

If τ_{i1} is in H^{seg}_{ij} then $pred(H^{seg}_{ij})$ is undefined. For example in Figure 2.8, we have $pred(H^{seg}_{i4}) = \tau_{i1}$.

The set of immediate successors of an H segment are tasks that do not belong to H_{ij}^{seg} but whose predecessors do:

$$succ(H_{ij}^{seg}) = \left\{\tau_{im} \mid \tau_{im} \notin H_{ij}^{seg} \wedge pred(\tau_{im}) \in H_{ij}^{seg}\right\} \tag{44}$$

For example in Figure 2.8, we have $succ(H_{i4}^{seg}) = \{\tau_{i6}, \tau_{i9}\}$.

An H segment H_{ij}^{seg} is said to precede a task τ_{im}, denoted by $H_{ij}^{seg} \prec \tau_{im}$, if:

$$\tau_{im} \notin H_{ij}^{seg} \wedge (pred(H_{ij}^{seg}) \prec \tau_{im} \vee \tau_{i1} \in H_{ij}^{seg}) \tag{45}$$

For example in Figure 2.8, H_{i4}^{seg} precedes τ_{i6}, τ_{i7}, τ_{i8}, τ_{i9}, τ_{i10}.

H SEGMENT OFFSET AND JITTER H segments may have an offset and jitter defined by the offset and jitter of their first task(s). The first task(s) of an H segment is (are) the one(s) with the smallest offset.

Let $XP_i(\tau_{ab})$ denote the set of tasks in Γ_i that come first in their respective H segment:

$$XP_i(\tau_{ab}) = \{\tau_{if} \in hp_i(\tau_{ab}) \mid pred(\tau_{if}) \notin hp_i(\tau_{ab})\} \tag{46}$$

For example in Figure 2.8, we have $XP_i = \{\tau_{i3}, \tau_{i4}, \tau_{i7}, \tau_{i8}, \tau_{i10}, \}$.

The offset of an H segment H_{ij}^{seg} is then:

$$O_{ij}^{seg}(\tau_{ab}) = O_{if} \mid \tau_{if} \in XP_i(\tau_{ab}) \wedge H_{if}^{seg}(\tau_{ab}) = H_{ij}^{seg}(\tau_{ab}) \tag{47}$$

For example in Figure 2.8, we have $O_{i7}^{seg} = O_{i7} = O_{i8}$.

The jitter $J_{ij}^{seg}(\tau_{ab})$ of an H segment H_{ij}^{seg} is defined in the same way. For example in Figure 2.8, we have $J_{i7}^{seg} = J_{i7} = J_{i8}$.

PHASING OF H SEGMENT It is also possible to define the phasing of an H segment. The phasing of an H segment is described by values that describe the phasing of the first task(s) of the H segment.

Job $p = 1$ of an H segment H_{ij}^{seg}, when τ_{ik} starts the busy period, is released at:

$$\varphi_{ijk}^{seg}(\tau_{ab}) = T_i + O_{ij}^{seg}(\tau_{ab}) - (O_{ik} + J_{ik}) \mod T_i \tag{48}$$

Otherwise said, job $p = 1$ of an H segment H_{ij}^{seg} is released at the time job $p = 1$ of its first task(s) is released. For example in Figure 2.8, we have $\varphi_{i4k}^{seg} = \varphi_{i4k}$.

The first pending job of an H segment H_{ij}^{seg} at t_c is numbered:

$$p_{0,ijk}^{seg}(\tau_{ab}) = 1 - \left\lfloor \frac{J_{ij}^{seg}(\tau_{ab}) + \varphi_{ijk}^{seg}(\tau_{ab})}{T_i} \right\rfloor \tag{49}$$

Otherwise said, the first pending job of an H segment H_{ij}^{seg} at t_c is the first pending job of its first task(s). For example in Figure 2.8, we have $p_{0,i4k}^{seg} = p_{0,i4k}$.

EXECUTION CONFLICTS OF H SEGMENT There are execution conflicts between tasks and an H segment is a set of tasks. Therefore the exists execution conflicts between H segments. They are in *blocking conflict* or *precedence conflict*.

H segments that have an immediate predecessor in lp_i are called blocking H segments. Only one of these from all jobs of all transactions, can execute within a τ_{ab} busy period. For example, H^{seg}_{i3}, H^{seg}_{i4}, and H^{seg}_{i7} are blocking H segments. Two tasks that belong to different blocking H segments are said to be in *blocking conflict*. For example, tasks τ_{i3}, τ_{i5}, τ_{i8} are in blocking conflict.

Let us consider τ_{ix} and τ_{iy} both in lp_i. They are said to be in *precedence conflict*, if they belong to different H sections and either $H^{seg}_{ix} \prec \tau_{iy}$ or $H^{seg}_{iy} \prec \tau_{ix}$. For example, τ_{i5} and τ_{i7} are in precedence conflict.

2.4.4.3 *Op1: Worst Case Scenario*

A scenario, where τ_{ik} starts the τ_{ab} busy period, is created so the phasing of Γ_i can be determined, so its interference can computed.

A scenario is created by choosing $\tau_{ik} \in \Gamma_i$ to start the τ_{ab} busy period at t_c. Tasks in Γ_i that may start the busy period are in a set $XP_i(\tau_{ab})$, which is the set of tasks that come first in their respective H segments. In the case of Γ_a, to which τ_{ab} belongs to, task $\tau_{ac} \in XP_a(\tau_{ab})$ starts the τ_{ab} busy period. This reduces the number of scenarios to create compared to a test like WCDO [106, 118].

For example in Figure 2.8, tasks in $XP_i = \{\tau_{i3}, \tau_{i4}, \tau_{i7}, \tau_{i8}, \tau_{i10}, \}$ are tasks τ_{ikk} that may start the τ_{ab} busy period at t_c.

For the analysis of τ_{ab}, each scenario created by $\tau_{ac} \in XP_a$ is combined with only one scenario created by a τ_{ik}, in each XP_i. Otherwise said, each τ_{ac} is only combined with only one τ_{ik} in each XP_i. For example, if there are 2 tasks in XP_a, and there are 3 other XP_i sets, then there are $2 \times 3 = 6$ combinations. Task τ_{ik} of XP_i is chosen if it leads to the greatest interference of Γ_i to τ_{ab}, in the scenario where τ_{ac} starts the τ_{ab} busy period. This τ_{ik} is determined in the interference computation operation (**Op2**).

The combinations of all scenarios, created by tasks τ_{ab} and τ_{ik}, are not checked because otherwise the operation is exponential. For example if there are 2 tasks in XP_a, and there are 3 other XP_i sets, with 2 tasks in each XP_i set, the number of combinations is $2^4 = 16$.

For tasks $\tau_{ik} XP_i$, the one that starts the one that gives the greatest interference of Γ_i, in the scenario where τ_{ac} starts the τ_{ab} busy period. is chosen to start the τ_{ab} busy period. As a reminder, each combination of tasks τ_{ac} and τ_{ik}

2.4.4.4 *Op2: Worst Case Interference*

Once a scenario is created, the interference from any transaction to jobs of τ_{ab} is computed. Let us consider the analysis of a job p_{ab} of τ_{ab}.

The test computes two kinds of interference for a transaction: blocking and non-blocking. Blocking interference is the interference of a transaction, with interference from one of its blocking H segment. Only one blocking interference from any job of any transaction in the system, can contribute to the τ_{ab} busy period. If a transaction's blocking interference is not chosen to contribute, then only its non-blocking interference contributes to the τ_{ab} busy period.

The following paragraphs show how to compute interference of jobs of a transaction Γ_i, and Γ_a, before or at t_c, then jobs after t_c. Afterwards it is shown how to compute the total interference, i.e. interference from all Γ_i and Γ_a.

It is assumed that task $\tau_{ik} \in XP_i$ of Γ_i, and task $\tau_{ac} \in XP_a$ of Γ_a, start the τ_{ab} busy period, of length w, at $t_c = 0$.

JOBS BEFORE OR AT t_c ($p \leqslant 0$) The blocking and non-blocking interferences are computed for the jobs p of Γ_i released before or at t_c. To compute both interferences the idea is to iterate through each pending job p of Γ_i before or at t_c that may interfere. The first pending job of Γ_i that may interfere is the first pending job of the H segment of the last task of Γ_i that belongs to hp_i.

The last task of Γ_i that belongs to hp_i is the task in hp_i with the greatest offset. This task is denoted by τ_{iN}. For example in Figure 2.8, assuming all tasks have a WCET of 1, τ_{iN} is τ_{i7} or τ_{i8} or τ_{i10}.

The first pending job of τ_{iN} is numbered $p^{seg}_{0,iNk}(\tau_{ab})$, which is computed with Equation 49. Therefore only jobs $p^{seg}_{0,iNk}(\tau_{ab}) \leqslant p \leqslant 0$ may interfere.

The blocking interference and the non-blocking interference of a particular job p of Γ_i are computed by exploring the tree representing the transaction. After exploring the whole tree, the combination of H segments, that gives the highest blocking and non-blocking interferences, is determined for job p.

Only some H segments may interfere together within the given scenario, where τ_{ik} and τ_{ac} start the τ_{ab} busy period. Indeed, some H segments are in blocking and precedence conflicts. Furthermore an H segment may also be in blocking or precedence conflict with the H segments of tasks τ_{ik}, τ_{ac}, and τ_{ab}.

After the blocking interference and non-blocking interference for each job p are computed, the blocking interferences are summed to get the interference of jobs $p \leqslant 0$ of Γ_i. The maximum of the non-blocking interferences is the non-blocking interference of jobs $p \leqslant 0$ of Γ_i.

For jobs $p \leqslant 0$ of Γ_a, the blocking and non-blocking interferences computation is similar to the Γ_i case. The difference relies in how some H segments may interfere together, since there are more execution conflicts in the case of Γ_a, due to the fact that τ_{ab} belongs to Γ_a and it may precede some other tasks [118].

The computation of interferences of jobs $p \leqslant 0$ of Γ_i and Γ_a is performed with three functions that are described in the next Section 2.4.4.5.

JOBS AFTER t_c $(p > 0)$ The interference of jobs after t_c is computed differently for Γ_i and Γ_a.

Interference of Γ_i Jobs $p > 0$ of τ_{ij} can only interfere τ_{ab} if τ_{ij} belongs to the first H section of Γ_i, and the first H segment is not a blocking segment. As such, the set of tasks that may interfere are in:

$$MP_i(\tau_{ab}) = \{\tau_{il} \in \Gamma_i \mid (\neg\exists \tau_{ix} \in lp_i(\tau_{ab}) \mid \tau_{ix} \prec \tau_{il})\} \tag{50}$$

In Figure 2.8, $MP_i = \emptyset$ because $\tau_{i1} \in lp_i$. Otherwise suppose that $\tau_{i1} \notin lp_i$. Graphically in the figure, this would mean τ_{i1} is a crossed task. In this case we have $MP_i = \{\tau_{i4}, \tau_{i5}, \tau_{i10}\}$.

A task τ_{ij} in MP_i contributes to the τ_{ab} busy period if its H segment H^{seg}_{ij} is released within the busy period. As such, the interference from jobs $p > 0$ of tasks τ_{ij} in MP_i is:

$$W_{ik}(\tau_{ab}, w)\mid_{p>0} = \sum_{\tau_{ij} \in MP_i(\tau_{ab})} \left\lceil \frac{w - \varphi^{seg}_{ijk}(\tau_{ab})}{T_i} \right\rceil_0 C_{ij} \tag{51}$$

where $\lceil x \rceil_0 = \max(\lceil x \rceil, 0)$.

In Figure 2.8, let us suppose again that $\tau_{i1} \notin lp_i$. Let us now suppose that each task has a WCET of 1 and the busy period w is long enough for jobs $0 < p \leqslant 3$ of tasks τ_{ij} to be released within the busy period. Only tasks τ_{i4}, τ_{i5}, and τ_{i10} can interfere because they belong to the first H section H_{i4}. We then have $W_{ik}(\tau_{ab}, w)\mid_{p>0} = 3C_{i4} + 3C_{i5} + 3C_{i10} = 9$.

Interference of Γ_a Similarly to Γ_i, jobs $p > 0$ of τ_{aj} can only interfere τ_{ab}, if they belong to a similarly defined MP_a set. Contrary to Γ_i, a job of a task in MP_a cannot interfere τ_{ab} if the task is preceded by τ_{ab} and the job is the analyzed job p_{ab} or later jobs. Therefore there are two sets of tasks that may interfere τ_{ab}:

$$MP_a^1(\tau_{ab}) = \{\tau_{al} \in MP_a(\tau_{ab}) \setminus \{\tau_{ab}\} \mid \neg(\tau_{ab} \prec \tau_{al})\} \tag{52}$$

$$MP_a^2(\tau_{ab}) = \{\tau_{al} \in MP_a(\tau_{ab}) \setminus \{\tau_{ab}\} \mid \tau_{ab} \prec \tau_{al}\} \tag{53}$$

Tasks in MP_a^1 are not preceded by τ_{ab} so the interference of their jobs $p > 0$ is:

$$W_{ac}^1(\tau_{ab}, w) = \sum_{\tau_{aj} \in MP_a^1(\tau_{ab})} \left\lceil \frac{w - \varphi_{ajc}^{seg}(\tau_{ab})}{T_a} \right\rceil_0 C_{aj} \tag{54}$$

Tasks in MP_a^2 are preceded by τ_{ab} so at most $p_{ab} - 1$ jobs of these tasks may interfere:

$$W_{ac}^2(\tau_{ab}, w, p_{ab}) = \sum_{\tau_{aj} \in MP_a^2(\tau_{ab})} \min\left(p_{ab} - 1, \left\lceil \frac{w - \varphi_{ajc}^{seg}(\tau_{ab})}{T_a} \right\rceil_0 \right) C_{aj} \tag{55}$$

In Figure 2.8, let us suppose that τ_{i5} is the analyzed task, i.e. $\tau_{ab} = \tau_{i5}$ and $\Gamma_a = \Gamma_i$. Let us also suppose that $\tau_{i1} \notin lp_i$. Graphically in the figure, this would mean τ_{i1} is a crossed task. In this case we have $MP_a^1 = \{\tau_{i4}\}$. We also have $MP_a^2 = \{\tau_{i10}\}$.

Suppose that job $p_{ab} = 3$ of τ_{i5} is under analysis. Then at most 2 jobs of τ_{i10} can interfere job 3 of τ_{i5}. There is no such restriction on the interference of jobs of τ_{i4} since it is not preceded by τ_{i5}.

Sets MP_a^1 and MP_a^2 both do not contain τ_{ab}. Thus when job p_{ab} of τ_{ab} is under analysis, the contribution of p_{ab} number of its jobs, to the busy period, must be added. For $p_{ab} < 0$, the contribution cannot be negative. Therefore the total interference from jobs $p > 0$ of Γ_a is:

$$W_{ac}(\tau_{ab}, w, p_{ab}) \mid_{p>0} = W_{ac}^1(\tau_{ab}, w) + \max(0, p_{ab} \cdot C_{ab} + W_{ac}^2(\tau_{ab}, w, p_{ab})) \tag{56}$$

Let us take the same example with the same assumptions as stated previously. Let us suppose that each task has a WCET of 1 and the busy period w is long enough for jobs $0 < p \leqslant 3$ of tasks τ_{aj} to be released in the τ_{i5} busy period. We have $W_{ac}(\tau_{i5}, w, 3) \mid_{p>0} = 3C_{i4} + \max(0, 3C_{i5} + 2C_{i10}) = 8$

TOTAL INTERFERENCE The total interference of Γ_i is composed of interference of jobs $p \leqslant 0$ and jobs $p > 0$. Let $W_{ik}(\tau_{ab}, w)$ denote the non-blocking interference and $WB_{ik}(\tau_{ab}, w)$ the blocking interference. Therefore we have $W_{ik}(\tau_{ab}, w) \leqslant WB_{ik}(\tau_{ab}, w)$. These values are computed as follows:

$$[W_{ik}(\tau_{ab}, w, \tau_{ac}), WB_{ik}(\tau_{ab}, w, \tau_{ac})] = [\mathit{transI\_NoB}, \mathit{transI\_B}] + [1,1].W_{ik}(\tau_{ab}, w) \mid_{p>0} \tag{57}$$

where [*transI_NoB*, *transI_B*] are the non-blocking and blocking interference of jobs $p \leqslant 0$ of Γ_i.

In Figure 2.8, let us suppose that task $\tau_{ik} = \tau_{i8}$ and $\tau_{ac} = \tau_{ab}$ start the busy period. Let us suppose that the busy period w is long enough for jobs $0 \leqslant p \leqslant 3$ of tasks τ_{ij} to be released within the busy period. Finally let us suppose that each task has a WCET of 1 except $C_{i8} = 5$. We then have $W_{i8}(\tau_{ab}, w, \tau_{ab}) = C_{i10} = 1$ and $WB_{i8}(\tau_{ab}, w, \tau_{ab}) = C_{i10} + C_{i7} + C_{i8} = 7$.

The result of the example means that for job $p = 0$, the blocking interference $WB_{i8}(\tau_{ab}, w, \tau_{ab})$ of Γ_i is the interference of H_{i8}^{seg} (blocking segment) and H_{i10}^{seg} (non-blocking H segment). The non-blocking interference $W_{i8}(\tau_{ab}, w, \tau_{ab})$ is simply the interference of H_{i10}^{seg} (non-blocking H segment).

For jobs $p = \{1, 2, 3\}$, the first H segment is a blocking H segment so no jobs $p > 0$ of Γ_i cannot interfere.

Once interference has been computed for each τ_{ik} that may start the τ_{ab} busy period, an upper-bound to the interference can be computed. Let $W_i^*(\tau_{ab}, w, \tau_{ac})$ (resp. $WB_i^*(\tau_{ab}, w, \tau_{ac})$) denote the upper-bound of the blocking (resp. non-blocking) interference of Γ_i. The upper-bound to the blocking interference is the greatest blocking interference computed for each scenario where τ_{ik} starts the τ_{ab} busy period. The upper-bound to the non-blocking interference is computed similarly.

$$W_i^*(\tau_{ab}, w, \tau_{ac}) = \max_{\tau_{ik} \in XP_i(\tau_{ab})} W_{ik}(\tau_{ab}, w, \tau_{ac}) \tag{58}$$

$$WB_i^*(\tau_{ab}, w, \tau_{ac}) = \max_{\tau_{ik} \in XP_i(\tau_{ab})} WB_{ik}(\tau_{ab}, w, \tau_{ac}) \tag{59}$$

Previously, in the example, the computed blocking and non-blocking interference, for $\tau_{ik} = \tau_{i7}$, are the upper-bounds: $W_i^*(\tau_{ab}, w, \tau_{ac}) = W_{i8}(\tau_{ab}, w, \tau_{ab}) = 1$ and $WB_i^*(\tau_{ab}, w, \tau_{ac}) = WB_{i8}(\tau_{ab}, w, \tau_{ab}) = 7$. Indeed, when τ_{ik} starts the τ_{ab} busy period, we get the greatest interference since $C_{i8} = 5$ and other tasks have a WCET of 1.

The difference between the blocking and non-blocking interference of Γ_i can be expressed as an interference increase to the blocking interference:

$$\Delta W_i^*(\tau_{ab}, w, \tau_{ac}) = WB_i^*(\tau_{ab}, w, \tau_{ac}) - W_i^*(\tau_{ab}, w, \tau_{ac}) \tag{60}$$

In the example, we have $\Delta W_i^*(\tau_{ab}, w, \tau_{ac}) = WB_{i8}(\tau_{ab}, w, \tau_{ab}) - W_{i8}(\tau_{ab}, w, \tau_{ab}) = 6$.

Similarly the non-blocking interference of Γ_a is denoted by $W_{ac}(\tau_{ab}, w, p_{ab})$, computed with corresponding equations for Γ_a. The blocking interference is denoted by $WB_{ac}(\tau_{ab}, w, p_{ab})$ and the interference increase is denoted by $\Delta WB_{ac}(\tau_{ab}, w, p_{ab})$. Upper-bounds are not computed for Γ_a since we iterate through each scenario where $\tau_{ac} \in XP_a$ starts the τ_{ab} busy period. The reason to not compute an upper-bound, an thus iterate through τ_{ac} tasks, is to reduce some pessimism [118].

2.4.4.5 *Op2 cont.: Worst Case Interference Algorithms*

The previous section showed how interference from Γ_i and Γ_a is computed. The equations for the computation of the interference of jobs $p > 0$ were given. This section describes the algorithms for the computation of the interference of jobs $p \leqslant 0$.

The interference of jobs $p \leqslant 0$ of Γ_i is computed with three functions:

– **(f1)** Compare/sum interference of each job $p \leqslant 0$ of Γ_i
– **(f2)** Compute interference of a particular job p of Γ_i
– **(f3)** Compute interference of a particular task of job p of Γ_i

The following three paragraphs describe each function. Then the final paragraph introduces the modifications necessary for Γ_a.

(f1) TRANSACTIONINTERFERENCE Interference from jobs of Γ_i before or at t_c is computed by the `TransactionInterference` function. The function takes as input parameters:

– τ_{ab}: task to analyze
– τ_{ik}: task that starts the τ_{ab} busy period
– w: length of the busy period
– τ_{ac}: task in Γ_a that starts the τ_{ab} busy period

The function returns a transaction's blocking and non-blocking interference:

– transI_NoB: non-blocking interference

– transI_B: blocking interference

The idea is to iterate through each pending job p of Γ_i, before or at t_c, that may interfere. The first pending job of Γ_i that may interfere is the first pending job of the H segment of the last task of Γ_i that belongs to hp_i. Algorithm 2.1 shows the `TransactionInterference` function.

Algorithm 2.1 TransactionInterference Function

```
 1: function TRANSACTIONINTERFERENCE($\tau_{ab}, \tau_{ik}, w, \tau_{ac}$)
 2:     Add ghost root task $\tau_{i0}$ as predecessor to tasks without any predecessor
 3:
 4:     for p in $p^{seg}_{0,iNk}(\tau_{ab})$..0 do
 5:         [jobI, jobDelta] ← BranchInterference($\tau_{ab}, \tau_{ik}, \tau_{i0}, w, p, \tau_{ac}$)
 6:         transI_NoB ← transI_NoB + jobI
 7:         transDelta ← max(transDelta, jobDelta)
 8:     end for
 9:
10:     transI_B ← transI_NoB + transDelta
11:     return [transI_NoB, transI_B]
12: end function
```

To simplify the algorithm, line 2 adds a ghost root task as predecessor of tasks without predecessors. A ghost root task is not formally defined in [118] so it may be defined as follows:

Definition 55 (Ghost Root Task). *A ghost task is allocated alone on a processor modeled only for the purpose of the analysis. It has a BCET and WCET of* 0*, an offset of* 0*, an infinity deadline, no jitter, a WCBT of* 0*, and its priority does not matter since it is allocated alone on the unique processor. In a transaction, a ghost root task is a ghost task that precedes all other tasks, and does not have any predecessor.*

(f2) BRANCHINTERFERENCE To compute the interference of a particular job $p \leqslant 0$ of Γ_i, since the transaction is tree-shaped, the tree is explored by a depth-first search algorithm. The algorithm is the `BranchInterference` function in Algorithm 2.2. The function takes as input parameters:

– τ_{ab}: task to analyze
– τ_{ik}: task in Γ_i that starts the τ_{ab} busy period
– τ_{iB}: task defining a branch (defined below)
– w: length of the busy period
– p: job of Γ_i to compute interference
– τ_{ac}: task in Γ_a that starts the τ_{ab} busy period

The function returns the interference of a branch without interference from any blocking H segment, and its interference increase if a blocking H segment of the branch is allowed to contribute to the τ_{ab} busy period:

– *branchI*: branch non-blocking interference
– *branchDelta*: branch interference increase

The tree is explored by branches. A branch is defined by a task denoted by τ_{iB}, and we have $\tau_{iB} \notin hp_i$. The branch contains τ_{iB} and all tasks preceded by τ_{iB}. For example in Figure 2.8, tasks $\tau_{i1}, \tau_{i2}, \tau_{i6}, \tau_{i9}$ define branches.

If τ_{iB} precedes an H segment, let τ_{im} be a task in the H segment A branch can have a number of sub-branches denoted by SB, which contain tasks in $succ(H^{seg}_{im})$ and immediate successors of τ_{iB} that are not in H^{seg}_{im}. For example in Figure 2.8, the set of sub-branches of branch τ_{i1} is $SB = \{\tau_{i6}, \tau_{i9}\}$.

The following paragraphs explain the algorithm of the function. During the description of the algorithm, it is illustrated by examples taken from the tree-shaped transaction in Figure 2.8. Let us assume that all tasks have a WCET of 1, except $C_{i8} = 5$. Let us assume also that all tasks can potentially be released in the τ_{ab} busy period. Values are computed for the τ_{i1} branch.

The `BranchInterference` function is called recursively. The general idea is to compute interference of an H section H_{im} preceded by τ_{iB}, denoted by *sectionI* (line 5), and compare/sum it with

Algorithm 2.2 BranchInterference Function

```
1: function BRANCHINTERFERENCE($\tau_{ab}, \tau_{ik}, \tau_{iB}, w, p, \tau_{ac}$)
2:     SB ← $succ(\tau_{iB})$
3:     if $\exists \tau_{im} \in SB \mid \tau_{im} \in hp_i(\tau_{ab})$ then
4:         $S \leftarrow \{\tau_{il} \in H_{im}(\tau_{ab}) \mid \tau_{iB} \prec \tau_{il}\}$
5:         sectionI ← $\sum_{\tau_{ij} \in S}$ TaskInterference($\tau_{ab}, \tau_{ik}, \tau_{ij}, w, p, \tau_{ac}$)
6:         $SB \leftarrow \{SB \cup succ(H^{seg}_{im}(\tau_{ab}))\} \setminus \{succ(\tau_{iB}) \cap H^{seg}_{im}(\tau_{ab})\}$
7:     end if
8:
9:     for $\tau_{iS} \in SB$ do
10:        [bI, bD] ← BranchInterference($\tau_{ab}, \tau_{ik}, \tau_{iS}, w, p, \tau_{ac}$)
11:        subBranchesI ← subBranchesI + bI
12:        subBDelta ← max(subBDelta, bD)
13:    end for
14:
15:    if $\tau_{iB} \in lp_i(\tau_{ab})$ then
16:        branchI ← subBranchesI
17:        branchDelta ← max(sectionI - subBranchesI, subBDelta)
18:    else
19:        branchI ← max(sectionI, subBranchesI)
20:        branchDelta ← max(subBranchesI + subBDelta - branchI, 0)
21:    end if
22:
23:    return [branchI, branchDelta]
24: end function
```

interference from sub-branches of the τ_{iB} branch in SB, denoted by *subBranchesI* and *subBDelta*. This will compute correct values of *branchI* and *branchDelta* by eliminating blocking and precedence conflicts.

For the τ_{i1} branch, *sectionI* is the sum of WCETs of tasks in $H_{i4} = \{\tau_{i4}, \tau_{i5}, \tau_{i10}\}$:

$$sectionI = C_{i4} + C_{i5} + C_{i10} = 3 \tag{61}$$

Value *subBranchesI* (line 11) is the sum of *branchI* (denoted by *bI*) computed for sub-branches of the τ_{iB} branch. Value *subBDelta* (line 12) is the maximum *branchDelta* (denoted by *bD*) computed for sub-branches. These values are returned by calls of the `BranchInterference` function to tasks in SB.

For the τ_{i1} branch, the sub-branches are τ_{i2}, τ_{i6}, τ_{i9}. Tasks τ_{i2} and τ_{i6} are in lp_i so only τ_{i9} returns a non-blocking branch interference that is added to *subBranchesI*:

$$subBranchesI = C_{i10} = 1 \tag{62}$$

For the same reason, only branches τ_{i2} and τ_{i6} return a blocking interference to be compared for the computation of *subBDelta* of branch τ_{i1}:

$$subBDelta = \max(C_{i3}, (C_{i7} + C_{i8})) = \max(1, 1 + 5) = 6 \tag{63}$$

To compute the values of *branchI* and *branchDelta* for the τ_{iB} branch, it must be determined if the H segment H^{seg}_{im} preceded by τ_{iB} is a blocking one, i.e. if τ_{iB} is in lp_i (line 15).

If H^{seg}_{im} is a blocking segment, the non-blocking interference is simply the interference of the sub-branches (line 16).

Notice that the interference of H_{im}^{seg} is *sectionI* - *subBranchesI*. This value is compared to the interference increase *subBDelta* of sub-branches, to get the maximum value to assign to *branchDelta* (line 17).

If H_{im}^{seg} is not a blocking segment, then the interference of H_{im} (sectionI) is compared to interference of sub-branches to eliminate precedence conflicts (line 19). The value of *branchDelta* is then computed accordingly (line 20).

For the τ_{i1} branch, task $\tau_{i1} \in lp_i$. Therefore its *branchI* and *branchDelta* values are:

$$branchI = subBranchesI = 1 \tag{64}$$

$$branchDelta = \max(sectionI - subBranchsI, subBDelta) = \max(3-1, 6) = 6 \tag{65}$$

The *branchI* and *branchDelta* values of branch τ_{i1} mean that the transaction Γ_i has a non-blocking interference of 1 and a blocking interference of 6.

This result is expected. Indeed only τ_{i10} is in a non-blocking H segment. All other tasks are in blocking segments and we have:

$$(C_{i7} + C_{i8} = 6) > (C_{i4} + C_{i5} = 1) > (C_{i3} = 1) \tag{66}$$

Therefore H_{i7}^{seg} gives the largest blocking interference, which is the blocking interference of transaction Γ_i.

(f3) TASKINTERFERENCE When computing interference of a particular job $p \leqslant 0$ of Γ_i, each task's job's interference is computed by the `TaskInterference` function. The function takes as input parameters:
- τ_{ab}: task to analyze
- τ_{ik}: task in Γ_i that starts the τ_{ab} busy period
- τ_{ij}: a task to compute the interference
- w: length of the busy period
- p: job of τ_{ij} to compute interference
- τ_{ac}: task in Γ_a that starts the τ_{ab} busy period

The function returns the task's WCET if it can interfere, otherwise 0:
- *taskI*: interference of the task τ_{ij}

A job p of a task τ_{ij} can interfere if it is a job later than $p_{0,ijk}^{seg}$ and if it is released in $[0, w)$:

$$p \geqslant p_{0,ijk}^{seg} \wedge w > \varphi_{ijk}^{seg} + (p-1)T_i \tag{67}$$

A job of a task can only interfere if it passes a number of *reduction rules* that eliminate furthermore execution conflicts due to some jobs of tasks that must be in the τ_{ab} busy period: jobs of τ_{ik}, τ_{ac}, and τ_{ab} itself.

For example in Figure 2.8, if $\tau_{ik} = \tau_{i7}$ starts the τ_{ab} busy period, then each task of H_{i4}^{seg} has nil interference because H_{i4}^{seg} and H_{i7}^{seg} are in blocking conflict and because they are also in precedence conflict. Either conflict is sufficient to reduce the interference of one of the two H segments.

The complete `TaskInterference` function is in Algorithm 2.3. The reduction rules are shown in the function.

Note that the parameter τ_{ac}, that was passed through the different function until now, is only used in the 4th reduction rule. This parameter does not exist when $\Gamma_i = \Gamma_a$ so neither does the the 4th reduction rule when computing the interference of jobs of tasks in Γ_a.

MODIFICATIONS FOR INTERFERENCE OF Γ_a For jobs $p \leqslant 0$ of Γ_a the interference computation is similar to the Γ_i case. The only difference is the modification of a reduction rule and the addition of two new reduction rules in the `TaskInterference` function. These rules are in [118].

Algorithm 2.3 TaskInterference Function

```
1: function TaskInterference($\tau_{ab}, \tau_{ik}, \tau_{ij}, w, p, \tau_{ac}$)
2:     if $p \geqslant p^{seg}_{0,ijk} \wedge w > \varphi^{seg}_{ijk} + (p-1)T_i$ then
3:         taskI ← $C_{ij}$
4:     end if
5:     if $p \geqslant p^{seg}_{0,ikk} \wedge H^{seg}_{ik} \prec \tau_{ij} \wedge H_{ik} \neq H_{ij}$
6:      or $pred(H^{seg}_{ik}) \in lp_i \wedge pred(H^{seg}_{ij}) \in lp_i \wedge (H^{seg}_{ik} \neq H^{seg}_{ij} \vee p \neq p^{seg}_{0,ikk})$
7:      or $pred(H^{seg}_{ab}) \in lp_i \wedge pred(H^{seg}_{ij}) \in lp_i$
8:      or $pred(H^{seg}_{ac}) \in lp_i \wedge pred(H^{seg}_{ij}) \in lp_i$ then
9:         taskI ← 0
10:    end if                                                    ▷ Reduction rules 1 to 4
11:    return taskI
12: end function
```

Furthermore the job p_{ab} of τ_{ab}, currently under analysis, is an input parameter for all the functions. Like the τ_{ac} parameter for Γ_i, the p_{ab} parameter for Γ_a is only used for reduction rules.

2.4.4.6 *Op3: Worst Case Response Time*

Op3 consists in computing the WCRT of τ_{ab}. The following paragraphs give the details of the WCRT computation, which is divided into:

- Compute the length of the τ_{ab} busy period in a scenario
- Compute the number of jobs of τ_{ab} that occur in the busy period in a scenario
- Compute the response time of each job p_{ab} in τ_{ab} busy period of each scenario, and take the maximum of the response times as the WCRT of τ_{ab}

LENGTH OF BUSY PERIOD To estimate the number of jobs to analyze, the length of the busy period may be estimated. When τ_{ac} starts the τ_{ab} busy period, the length of the τ_{ab} busy period is denoted by L_{abc} and computed as follows:

$$L_{abc} = B_{ab} + W'_{ac} + \sum_{\forall i \neq a} W^*_i(\tau_{ab}, L, \tau_{ac}) + \max(WB'_{ac} - W'_{ac}, \Delta W^*_i(\tau_{ab}, L, \tau_{ac})) \tag{68}$$

where $W'_{ac} = W_{ac}(\tau_{ab}, w, p_{ab})$ and $W'_{ac} = WB_{ac}(\tau_{ab}, w, p_{ab})$.

NUMBER OF JOBS IN BUSY PERIOD Jobs of τ_{ab} numbered between $p^{seg}_{0,abc}(\tau_{ab})$ and $p^{seg}_{L,abc}(\tau_{ab})$ need to be analyzed. The lowest job number $p^{seg}_{0,abc}(\tau_{ab})$ is computed with Equation 49 applied to τ_{ac}. The highest job number $p^{seg}_{L,abc}(\tau_{ab})$ is computed as follows:

$$p^{seg}_{L,abc}(\tau_{ab}) = \left\lceil \frac{L_{abc} - \varphi^{seg}_{0,abc}(\tau_{ab})}{T_a} \right\rceil_0 \tag{69}$$

If τ_{ab} is not in MP_a, then $p^{seg}_{L,abc}(\tau_{ab}) = 0$.

WCRT OF TASK τ_{ab} First let $\Delta WB^*_{ac}(\tau_{ab}, w, p_{ab})$ denote the maximum interference increase among all transactions in the system:

$$\Delta WB^*_{ac}(\tau_{ab}, w, p_{ab}, \tau_{ac}) = \max(\Delta WB_{ac}(\tau_{ab}, w, p_{ab}), \Delta W^*_i(\tau_{ab}, w, \tau_{ac})) \tag{70}$$

The response time of job p_{ab} of τ_{ab}, when τ_{ac} starts the busy period, is derived from its completion time $w_{abc}(p_{ab})$ which is the sum of: its WCBT (B_{ab}); the non-blocking interference from Γ_a (W_{ac}); the sum of non-blocking interference upper-bounds from other transactions Γ_i; and the maximum interference increase ($\Delta WB^*_{ac}(\tau_{ab}, w, p_{ab}, \tau_{ac})$). The completion time is computed as follows:

$$w_{abc}(p_{ab}) = B_{ab} + W_{ac} + \sum_{\forall i \neq a} W^*_i + \max(\Delta W_{ac}, \Delta W^*_i) \tag{71}$$

Equation 71 is solved by iteration. Job p_{ab} of τ_{ab} is released at $\varphi_{abc} + (p_{ab} - 1)T_a$, so its response time, denoted by $R^w_{abc}(p_{ab})$, is computed as follows:

$$R^w_{abc}(p_{ab}) = w_{abc}(p_{ab}) - (\varphi_{abc} + (p_{ab} - 1)T_a) + O_{ab} \tag{72}$$

Equation 72 is applied to each job of τ_{ab} that may be in the busy period.

The WCRT of τ_{ab} (denoted by R^w_{ab}) is then the maximum $R^w_{abc}(p_{ab})$ for all τ_{ac} that start the τ_{ab} busy period:

$$R^w_{ab} = \max_{\tau_{ac} \in XP_a(\tau_{ab})} \left\{ \max_{p_{ab} = p^{seg}_{0,abc}(\tau_{ab})..p^{seg}_{L,abc}(\tau_{ab})} R^w_{abc}(p_{ab}) \right\} \tag{73}$$

The WCRT R^w_{ab} given by Equation 73 is compared to global deadline ($O_{ij} + d_{ij}$) of a task to determine if it is schedulable. The analysis is performed for all tasks τ_{ab} in the system to determine if the system is schedulable. The assessment of schedulability is performed at each iteration where response times are computed. After an iteration jitters are updated, if necessary, and then R^w_{ab} are re-computed if necessary.

The transaction and periodic task models both express task parameters. The next section presents a task model that expresses parameters of jobs of a task.

2.5 MULTIFRAME MODELS: ILLUSTRATION WITH GMF

The GMF task model exposed in [14] is a generalization of the multiframe task model proposed in [100]. Multiframe allows to model tasks that don't have the same execution time from one job to the other, due to some input data of the task. Such behaviors occur in the multimedia domain, which motivated the multiframe task models.

It is proven in [100] that if this behavior is not considered, in some cases the fundamental periodic task model [85] assesses that the tasks are not feasible, although they are in reality. When analyzed with the multiframe task model, the tasks are assessed feasible.

GMF generalizes the multiframe task model by relaxing some constraints on the task parameters: a task not only can have a different execution time from one job to the other, but it can also have a different deadline and minimum separation time to its next release time. GMF models the individual job parameters of a sporadic task.

In the following sections, the definition of the GMF task model is first given. Afterwards two tests are presented: a test for preemptive DP scheduling based on processor demand, and a test for FP scheduling based on response time.

2.5.1 *Definition*

The definition of a GMF task, and its parameters, is taken from [14]. A GMF task G_i is an independent task, with an ordered vector composed of N_i frames F^j_i, with $1 \leqslant j \leqslant N_i$. Each frame is a job of G_i, and has the following parameters:

– E^j_i is the WCET of F^j_i.

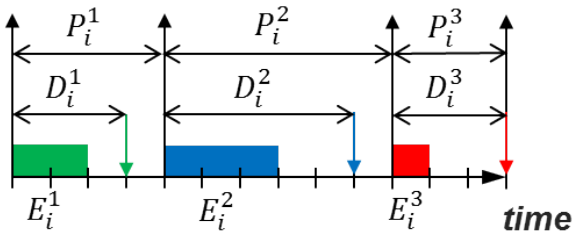


Figure 2.9: GMF Example: Up arrows are frame releases; Down arrows are frame deadlines

– D_i^j is the relative deadline of F_i^j.
– P_i^j is the min-separation of F_i^j, defined as the minimum time separating the release of F_i^j and the release of F_i^{j+1}.

A set of GMF tasks are assumed to be independent and all tasks are allocated on a uniprocessor. Under FP scheduling, a GMF task G_i has a fixed priority.

Frames are released cyclically [14]: frames are released in the order defined by the vector and any released frame $F_i^{j+k\times N_i}$ has parameters equal to frame F_i^j, for $k > 0$. Later this constraint is relaxed in [140], which proposes the non-cyclic GMF task model.

The sum of P_i^j is the GMF period of G_i, denoted by P_i [14]:

$$P_i = \sum_{i=1}^{N_i} P_i^j \tag{74}$$

Figure 2.9 shows an example of a GMF task G_i with three frames and their E_i^j, D_i^j, and P_i^j parameters illustrated on the timeline. For example, the first frame F_i^1 has parameters $E_i^1 = 2$, $D_i^1 = 3$, and $D_i^1 = 4$. These parameters are different from those of the second frame F_i^2, which are $E_i^1 = 3$, $D_i^1 = 5$, and $D_i^1 = 6$.

Two properties of GMF tasks are given below. Historically, these properties were given to ease the analysis. The properties hold if the deadlines of frames are constrained.

Property 2 (Localized Monotonic Absolute Deadline [14]). *A GMF task G_i is said to respect the localized Monotonic Absolute Deadline (l-MAD) property if any of its frame F_i^j has a deadline less than the deadline of the next frame F_i^{j+1}:*

$$\forall j \in [1; N_i], D_i^j \leqslant P_i^j + D_i^{j \bmod N_i + 1} \tag{75}$$

Property 3 (Frame Separation [138]). *A GMF task G_i is said to respect the Frame Separation property if any of its frame F_i^j has a deadline less than the release of the next frame F_i^{j+1}:*

$$\forall j \in [1; N_i], D_i^j \leqslant P_i^j \tag{76}$$

Property 3 is more restrictive than Property 2 [138]. Indeed a GMF task respecting the *Frame Separation* property also respects the *l-MAD* property [138].

In the following sections, first a feasibility test for GMF tasks is presented, for preemptive scheduling. Then a schedulability test is exposed, for preemptive FP scheduling of GMF tasks. Since GMF tasks are assumed to run on a uniprocessor, these tests are restricted to such systems.

2.5.2 *Processor Demand Feasibility Test*

A processor demand bound based feasibility test for GMF tasks is proposed in [14]. Let us assume that a set of GMF tasks with the *l-MAD* property is analyzed. To determine the feasibility of the GMF tasks, the approach in [14] is first to transform the processor demands by the GMF tasks to sporadic tasks. Then the feasibility of the sporadic tasks is assessed. The following paragraphs gives some details on this approach.

In [14] the $\mathrm{dbf}(G_i, t)$ function computes the maximum processor demand by frames of G_i released in $[0; t]$. The maximum processor demand by frames of G_i is computed by considering that any of its frames may start the critical instant by being released at time $t = 0$.

To compute $\mathrm{dbf}(G_i, t)$ for any t, in [14] the authors prove that only values of $t \in [0; P_i]$ need to be considered. The authors also prove that only values of t equal to all possible frame deadlines, contained within $[0; P_i]$, need to be considered. All possible frame deadlines within $[0; P_i]$ are obtained by considering that any frame can be released at time 0.

This determines the values of t for the computation of values of $\mathrm{dbf}(G_i, t)$. A list of couples (w, t) is then computed. Parameter t is a value to consider in $[0; P_i]$. Parameter w is the workload in the interval $[0; t]$.

As shown in [14], with this list it is possible to determine the maximum processor demand of G_i within any interval $[0; t']$, with $t' \leqslant P_i$. The maximum processor demand of a G_i is the largest w among all (w, t) pairs, where t' is less than t. Thus the list can also be used to compute $\mathrm{dbf}(G_i, t')$ for any $t' \leqslant P_i$.

After building the list of (w, t) for each G_i, each (w, t) is transformed to a list of sporadic tasks. Let us suppose that the list for G_i is $< (w_1, t_1), (w_2, t_2), ..., (w_m, t_m) >$. The transformation of G_i gives a set of sporadic tasks, with parameters (C_j, D_j, T_j) defined in Section 2.3:

$$\{(w_0 = 0, t_0 = 0, P_i), (w_1 - w_0, t_1 - t_0, P_i),$$
$$(w_2 - w_1, t_2 - t_1, P_i), ..., (w_m - w_{m-1}, t_m - t_{m-1}, P_i)\}$$

The problem of assessing feasibility of a set of GMF tasks then becomes the problem of assessing the feasibility of the sporadic tasks that are results of the transformation. If the set of sporadic tasks is feasible, then the set of GMF tasks is feasible [14]. As a reminder, feasibility tests for sporadic tasks are given in Section 2.3.

2.5.3 *Response Time Schedulability Test*

A response time based schedulability test for GMF tasks scheduled by FP scheduling is proposed in [138]. This test assumes that GMF tasks respect the *Frame Separation* property (Property 3).

Let us call $W_{ik}(t)$ the workload of a task G_i in the time interval $[0; t)$, when F_i^k is released at time 0. The computation of $W_{ik}(t)$ consists in summing the WCETs of frames that are released within $[0; t)$, starting with the release of F_i^k at 0:

$$\forall t > 0, W_{ik}(t) = \sum_{h=k}^{k+j-1} C_i^{(h-1) \bmod N_i + 1}, j = \min\left(j : \sum_{h=k}^{k+j-1} T_i^{(h-1) \bmod N_i + 1} \geqslant t\right) \quad (77)$$

To assess the schedulability of a GMF task, the schedulability of each of its frame is assessed through their response time. The WCRT of F_a^b occurs when F_a^b is released at the same time as a frame of each G_i of higher priority [138]. To compute the exact WCRT of F_a^b, each combination of frames, belonging to higher priority tasks G_i, must thus be tested. This gives a test of exponential time complexity.

A pseudo-polynomial complexity test can be derived by considering only the maximum interference that F_a^b can experience from frames of a higher priority task G_i. The maximum interference of G_i to F_a^b, in a time interval of length t, is computed with a function denoted by $M_i(t)$:

$$\forall t > 0, M_i(t) = \max_{1 \leqslant k \leqslant N_i} W_{ik}(t) \quad (78)$$

The WCRT of F_a^b is then computed recursively:

$$\begin{aligned} & R_a^{b,(0)} = C_a^b \\ & \forall n > 0, R_a^{b,(n)} = C_a^b + \sum_{i \in hp_a} M_i(R_a^{b,(n-1)}) \end{aligned} \tag{79}$$

where hp_a is the set of indexes of GMF tasks G_i of higher priority than G_a.

Once the WCRT of a frame is computed, the WCRT is compared to its deadline to assess the schedulability of the frame. This assessment is done for all frames of all tasks to assess schedulability of the tasks set.

To leverage some constraints on the modeling of jobs of a task, the next section presents a more generalized task model for the modeling of jobs.

2.6 DAG TASK MODELS: ILLUSTRATION WITH SPORADIC DAG TASK

The GMF task model allows to model different jobs of a sporadic task, with different parameters. The jobs have to execute in a single sequential order defined by the vector of frames of a GMF task.

To relax the constraint of a unique sequence of ordered jobs, it is proposed in [10] to model a task as a DAG. A DAG is a graph where for any vertex, there is no sequence of edges that will lead back to the vertex.

In the following section the sporadic DAG task model is first defined. This model is one of the more generalized models among those that represent a task as a DAG. Afterwards the relations between DAG task models is discussed. Finally feasibility and schedulability tests for the sporadic DAG task model, on a global multiprocessor system, are presented.

2.6.1 *Definition*

This section presents the definition of a sporadic DAG task given in [12, 24]. A sporadic DAG task is a tuple (G_i, D_i, T_i):

– G_i is a DAG specified as $G_i = (V_i, E_i)$, where V_i is a set of vertices and E_i a set of directed edges. Each $v \in V_i$ is a job. Each job v is characterized by e_v, the WCET of the job. Each edge (v_i, v_j) represents a precedence dependency between jobs: v_i must complete execution before v_j can execute. A job becomes eligible to execute once all of its predecessor jobs have completed execution.

– T_i is the period. When a DAG task is released, a *dag-job* is released. A dag-job is a sequence of precedence-related jobs represented by the vertices: when a dag-job is released at at time t, all jobs $v \in V_i$ are released at time-instant t. The period denotes the minimum separation time that must elapse between the release of successive dag-jobs: if a dag-job is released at t, then the next dag-job may not be released before $t + T_i$.

– D_i is a relative deadline. If a dag-job is released at t then all jobs that are released at t must complete execution by $t + D_i$

Some notations are defined in [12, 24] and used in the analysis:

– $vol(G_i)$ is the sum of WCETs of jobs of a sporadic DAG task G_i. It is computed as: $vol(G_i) = \sum\limits_{v \in V_i} e_v$.

– $len(G_i)$ is the length of the longest chain in G_i. A chain is a sequence of vertices $v_1 ... v_k$ such that $\forall 1 \leqslant j < k, (v_j, v_{j+1}) \in E_i$. The length of the chain is the sum of the WCETs of the vertices: $\sum\limits_{j=1}^{k} e_j$.

The DAG tasks execute on a global multiprocessor system, with m identical processors, scheduled by a preemptive scheduling policy. The speed of a processor is denoted by s, which is a fraction of its frequency. The processor is said to be of speed-s. For example a processor of speed-1 at 400 Mhz processes at a frequency of 200 Mhz at speed-0.5.

The sporadic DAG task is one of the most generalized task models that represent a task as a DAG but it is not the first. Let us see how it relates to other DAG task models.

2.6.2 *Generalizations Between DAG Task Models*

The representation of a sporadic task as a DAG dates back to the RRT task model proposed in [10]. The RRT uses the same notations as the sporadic DAG task model. The RRT model was later generalized by the non-cyclic RRT model proposed in [9]. A non-cyclic RRT does not repeat its jobs in a cycle.

Both RRT and non-cyclic RRT are applicable to tasks allocated on a uniprocessor. In [12] the RRT model is generalized as the sporadic DAG task model that was defined in the previous section. Contrary to RRT, the sporadic DAG task model is applicable to a global multiprocessor system.

The acyclic constraint of a DAG is relaxed in [136] by using a directed graph to represent a task. A directed graph is a graph where, contrary to a DAG, there may exits cycles. The resulting model is called DRT but although it is more expressive than the models based on DAGs, the DRT model is applicable to a uniprocessor system, contrary to the DAG task models.

In the next sections, feasibility and schedulability tests for sporadic DAG tasks, executing on a global multiprocessor system, are exposed.

2.6.3 *Speedup Bound Feasibility Tests*

A necessary and sufficient feasibility test for a set of sporadic DAG task on a global multiprocessor system, with m identical processors, is intractable (NP-hard) [24]. For this reason, the authors propose to develop approximate tests. The concept of speedup bound is used.

Definition 56 (Scheduling Policy Speedup Bound [24])**.** *A scheduling policy is said to have a speedup bound* $b \geqslant 1$ *if any tasks set, that is feasible on* m *speed-1 processors, is schedulable by the policy on* m *speed-*b *processors.*

Definition 57 (Schedulability Test Speedup Bound [24])**.** *A schedulability test has speedup bound* $b \geqslant 1$ *if the following holds: any tasks set that is feasible on* m *speed-*1 *processors is assessed by the test to be schedulable on* m *speed-*b *processors.*

Note that a schedulability test of speedup bound b also assesses that the tasks set is schedulable even if the tasks set is not feasible on m speed-1 processors.

The speedup bound of a test is a metric for quantifying the quality of the approximation of the test [12]. In [24], the speedup bounds for EDF and DM are given:

Theorem 13 (EDF Speedup Bound)**.** *Any DAG tasks set that is feasible on* m *speed-*1 *processors is schedulable by EDF on* m *speed-*b *processors where* $b = 2 - \frac{1}{m}$.

Theorem 14 (DM Speedup Bound)**.** *Any DAG tasks set that is feasible on* m *speed-*1 *processors is schedulable by DM on* m *speed-*b *processors where* $b = 3 - \frac{1}{m}$.

These speedup bounds are used to assess schedulability of tasks set scheduled by EDF (resp. DM), on m processors of speed $2 - \frac{1}{m}$ (resp. $3 - \frac{1}{m}$), using a pseudo-polynomial schedulability test proposed in [24]. If the test determines non-schedulability, then the tasks set is not feasible on m speed-1 processors.

2.6.4 *Schedulability Tests*

In [24] the authors also give some simple sufficient schedulability tests for m speed-1 processors. These tests run in polynomial time.

Theorem 15. *[24] A sporadic DAG tasks set of* n *tasks is schedulable by EDF on* m *speed-*1 *processors if:*

$$\forall k \in [1;n], \mathrm{len}(G_k) \leqslant D_k/3 \tag{80}$$

$$\forall k \in [1;n], \sum_{\forall i | T_i \leqslant D_k} (\mathrm{vol}(G_i)/T_i) + \sum_{\forall i | T_i > D_k} (\mathrm{vol}(G_i)/D_k) \leqslant (m + 1/2)/3 \tag{81}$$

Theorem 16. *[24] A sporadic DAG tasks set of* n *tasks is schedulable by DM on* m *speed-1 processors if:*

$$\forall k \in [1;n], \mathrm{len}(G_k) \leqslant D_k/5 \tag{82}$$

$$\forall k \in [1;n], \sum_{\forall i | T_i \leqslant D_k} (\mathrm{vol}(G_i)/T_i) + \sum_{\forall i | T_i > 2D_k} (\mathrm{vol}(G_i)/4D_k) \leqslant (m + 1/4)/5 \tag{83}$$

Theorem 17. *[24] A sporadic DAG tasks set of* n *tasks, all respecting* $D_i \leqslant T_i$*, is schedulable by DM on* m *speed-*1 *processors if:*

$$\forall k \in [1;n], \mathrm{len}(G_k) \leqslant D_k/4 \tag{84}$$

$$\forall k \in [1;n], \sum_{\forall i | T_i \leqslant D_k} (\mathrm{vol}(G_i)/T_i) + \sum_{\forall i | T_i > 2D_k} (\mathrm{vol}(G_i)/D_k) \leqslant (m + 1/3)/4 \tag{85}$$

2.7 CONCLUSION

In this Chapter scheduling analysis was presented. This thesis focuses on scheduling analysis methods based on feasibility and schedulability tests. Tests use several methods that compute either the processor utilization by tasks, the response time of tasks, the processor demand by tasks, or the processor speedup bound so the tasks are schedulable.

The tests are associated with task models and several common task models of the literature were put into relation by generalization. The task models that were presented are the fundamental periodic and sporadic task models, transactions, multiframe task models, and DAG task models.

The general idea that came out from this study is that task models are generalizations of each other and the more a task model is generalized, the more it is expressive, but the more its analysis is difficult. Some task models were illustrated and their tests were exposed.

In the next chapter, a system called software radio protocol is introduced. Its development is also exposed. The next chapter thus defines the context of the work of this thesis. A software radio protocol is a typical RTES with time constraints to verify during its development. Scheduling analysis can be applied to verify the time constraints. The applicability of scheduling analysis on a software radio protocol is studied in the next chapter.

Chapter 3

SCHEDULING ANALYSIS OF SOFTWARE RADIO PROTOCOL

Chapter 1 exposed some generalities on a RTES and its development methods. Verification of time constraints is an important part of the development of such a system and the verification can be done with scheduling analysis, presented in Chapter 2. The objective of the thesis is to perform scheduling analysis of software radio protocol, and to integrate the analysis into the development cycle of such a system. The goal of this chapter is thus to define the assumptions and context of the work, expose some problems that must be leveraged to achieve the objective, and give an overview of the solution proposed in this thesis.

In the rest of this chapter, the software radio protocol's radio domain will first be presented. Afterwards the system's architecture, and its development, will be described. This defines the assumptions made for scheduling analysis contributions proposed by this thesis. The problems of applying scheduling analysis to a software radio protocol is then be stated, before the solution to solve these problems are introduced.

3.1 INTRODUCTION TO RADIOS

Radios are now part of our daily life. The roots of radios go as far back as 1873, when James Clerk Maxwell put on paper the equations of the propagation of electromagnetic waves in free space. It then took more than 10 years, in 1888, before Heinrich Rudolf Hertz was able to conduct an experiment to transmit electromagnetic waves over the air. But it is then only about 20 years later that the term "radio" was generally adopted.

The basic concept of a radio is that when some digital data are to be transmitted, it is converted into electrical signal which is electrical energy from a source producing alternate current at a desired frequency. The electrical energy goes to a transmitter which converts it into electromagnetic wave, and impresses the signal on the wave by modulating it. Modulation consists in varying the properties of the electromagnetic wave. For example the modulation may consist in simply alternating the wave's on/off state, or in altering its amplitude, frequency, or phase. Once the electromagnetic wave is impressed with a signal, it is sent over the air by a resonant antenna. The wave propagates in the air and the process is reversed at the receiver's end to rebuild the digital signal.

In the following sections the concepts of a radio network is first presented. Then we will see how radio protocols are designed and how they enable the communication between users in the network. Afterwards a particular functionality of a radio protocol is exposed. Finally we will see how typical application constraints in communication systems translate to time constraints of the RTES domain.

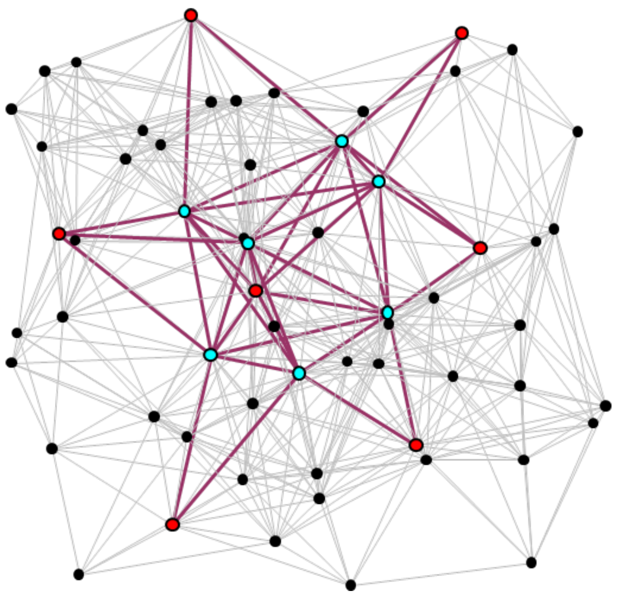


Figure 3.1: Wireless Ad-hoc Network: Circles are nodes; A line between two nodes indicates a link

3.1.1 *Radio Network*

Today radios are not only used for communication between two points, but are part of networks of several radio stations.

A network allows several users to exchange data between them. A user in the network is called node and when one node can reach another, there is a link between them and they are neighboring nodes. The nodes communicate over a shared communication medium. A medium in communication refers to the means of delivering and receiving data. In the case of radios, the medium is the air. The medium shared between radio stations is thus wireless, for example a frequency shared between stations.

One kind of network composed of several radio stations is a Mobile ad-hoc wireless NETwork (MANET), illustrated in Figure 3.1. Radios developed by Thales are used in this kind of network.

A MANET [55] is a kind of wireless ad-hoc network. In a wireless network, the number of neighboring radio stations that one can reach is limited in distance.

An ad-hoc network is a network where radio stations communicate without relying on an existing infrastructure, i.e. without relying on pre-existing nodes that have specific roles in the network. Each station in the ad-hoc network is free to associate itself with any other station. Thus each station of a wireless ad-hoc network participates in the exchange of data in the network. For example a station may not be the recipient of some data it receives. It then forwards the data to some other stations, until the recipient is reached.

The mobile characteristic of a MANET means that in this kind of wireless ad-hoc network, the stations are not geographically statically positioned. They may move and link up with different stations. For this reason, the network may be reconfigured and radio stations must then update their links.

To enable communication between the radio stations part of the network, a radio protocol stack must be defined.

3.1.2 *Radio Protocol Stack*

A radio protocol is a set of rules to respect so communication can be established between different users on a same network. A radio protocol defines the syntax and semantics of the messages

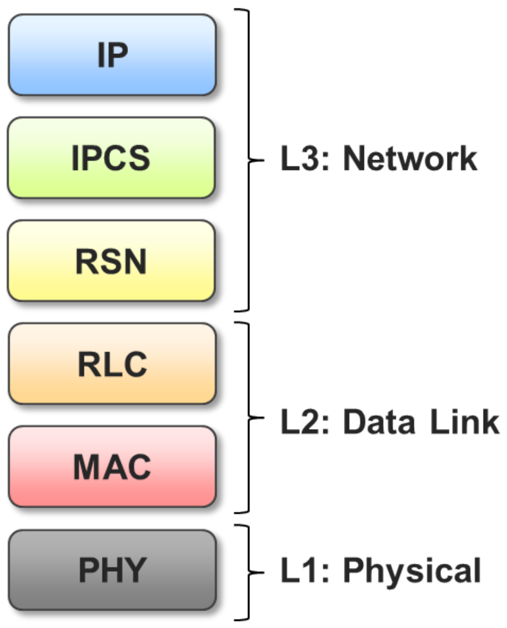


Figure 3.2: Protocol Stack

exchanged between users, but also how communication is synchronized between all users. Several protocols can cooperate.

The design of a protocol is done on the basis of the concept of layering. The Open Systems Interconnection (OSI) model [152] for communication systems, proposed in 1970, standardizes the functions of a communication system by dividing them into abstract layers. Each layer represents one or several protocols, and it interfaces with a layer above and below it, representing themselves other protocols. A layer serves the layer above and is served by the layer below. The set of layers, organized from top to bottom, is called the protocol stack. Abstract layers of the OSI model are:

– L1, Physical: Modulation and conversion of digital data into physical signal to be sent over the communication medium, establishment of connection between two radio stations.
– L2, Data Link: Provides a reliable link between two neighboring nodes of a network, detects errors in the physical layer, controls medium access.
– L3, Network: Routing of messages to be sent, management of addresses

The protocol stacks developed at Thales follow loosely the OSI model. An example of a radio protocol stack designed at Thales is shown in Figure 3.2. The PHYsical (PHY) layer is implemented as part of the L1 abstract layer in the OSI model. The Media Access Controller (MAC), and Radio Link Control (RLC) are implemented as part of the L2 abstract layer. Finally the Radio Sub-Network (RSN), Internet Packet Convergence Sub-layer (IPCS) layers are implemented as part of the L3 abstract layer. The IPCS layer, of the radio protocol stack, interfaces with the IP stack of the system above it.

The functionalities of layers of a radio protocol stack may be impacted by the method used by the radio to access the communication medium. The next section presents a typical method used at Thales.

3.1.3 *Impact of TDMA on Radio Protocol Stack*

Time Division Multiple Access (TDMA) [30] is a channel access method, based on time-division multiplexing. It allows several radio stations to transmit over a same communication medium. In TDMA, time is divided into several time slots, called TDMA slots. At each slot, each radio station in the network either transmits or receives. The sequence of slots is represented as a TDMA frame. Figure 3.3 shows a typical frame.

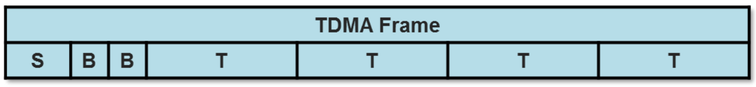


Figure 3.3: TDMA Frame

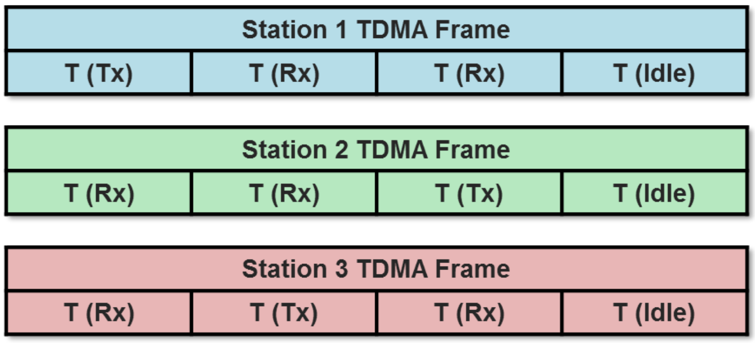


Figure 3.4: TDMA Frame Modes

Slots may be of different types. For example in Figure 3.3, the TDMA frame has three types of slot: *Service* (*S*) for synchronization between stations; *Beacon* (*B*) for observation/signaling the network; *Traffic* (*T*) for effective data transmission/reception.

Slots of different types do not necessarily have the same characteristics. One of the characteristics of a slot is its duration. Therefore slots of different types do not necessarily have the same duration. For example in Figure 3.3, a *B* slot duration is shorter than a *S* and *T* slot duration. The TDMA frame duration is the sum of durations of its slots.

A TDMA configuration defines the combination of slots, of different types, in a TDMA frame. A TDMA frame is repeated after it finishes. An instance of a frame is a cycle.

Slots can either be in transmission (*Tx*), reception (*Rx*), or *Idle* mode. In a *Tx* slot, a radio station can thus transmit data. When a radio station can transmit in a slot, we say that the slot is allocated to the station. When there are more stations than slots in a TDMA frame, some slots in the next cycle of the TDMA frame can be allocated to stations that did not transmit in the previous cycle.

To illustrate transmission, reception, and idling, consider the four *T* slots in Figure 3.3. If three stations are to communicate, Figure 3.4 shows a possible allocation of *T* slots. The first slot of radio station 1 is in *Tx* mode (station 1 transmits), so the first slots of the other radio stations are in *Rx* mode (they receive). The same logic is respected for the second and third slots. In the fourth slot, since there is no transmission or reception to be done, the slot is in *Idle* mode for all stations.

When TDMA is used to access the communication medium, it impacts the functionalities of the radio protocol stack. The allocation of TDMA slots is done in the RSN layer. RLC segments IP packets, provided by the IPCS layer. RLC segments them into smaller data packets, which are small enough in size to be sent in a TDMA slot. The MAC layers fetches these data packets regularly, before the start of a *Tx* slot, and delivers them to the PHY layer so they are sent over the air in the slot. In a *Rx* slot, the MAC layer receives some data packets to deliver to the RLC layer that will assemble the data packets into an exploitable IP packet. Therefore TDMA has an impact on the activities in the layers of the radio protocol stack.

The TDMA method is a particular case of the timed-token protocol for real-time communications [94]. Indeed the same approach of dividing time into time slots, and allocating slots to entities, is used in both methods. They also both focus on network scheduling, instead of task scheduling, but the next section shows how the network scheduling has an impact on the task scheduling.

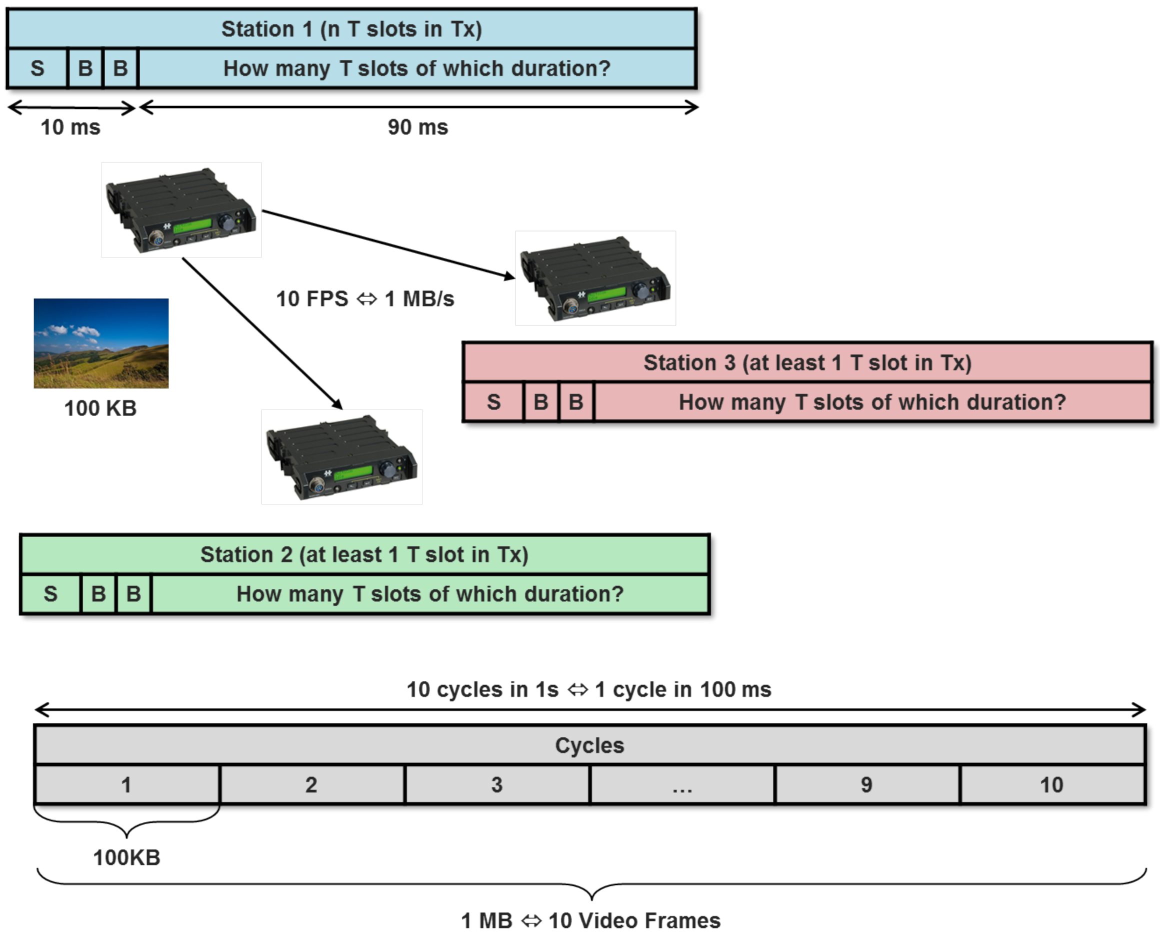

Station 1 (n T slots in Tx)
S
B
B
How many T slots of which duration?
10 ms
90 ms
10 FPS ⇔ 1 MB/s
100 KB
Station 3 (at least 1 T slot in Tx)
S
B
B
How many T slots of which duration?
Station 2 (at least 1 T slot in Tx)
S
B
B
How many T slots of which duration?
10 cycles in 1s ⇔ 1 cycle in 100 ms
Cycles
1
2
3
...
9
10
100KB
1 MB ⇔ 10 Video Frames

Figure 3.5: Radio Network with QoS Constraints

3.1.4 *From Application Constraints and Radio Protocol Configuration to Task Constraints*

The configuration of a radio, like the TDMA configuration, is impacted by the user application constraints (e.g. video streaming quality). In the case of TDMA, the characteristics of TDMA slots, like their duration, are impacted by application constraints.

Consider the example of a radio network in Figure 3.5 that illustrates how constraints, apparently not related to time, will result in different time constraints on the system. The figure shows a network where *station* 1 streams a video to two other stations. The requirement is to stream at least 10 Frames Per Second (FPS). TDMA is implemented in the radio protocols. The goal is to determine the allocation of *T* slots, the duration of *T* slots, and how many *T* slots are in a TDMA frame.

Let us assume a video frame is 100 KB and thus a minimum of 1 MB must be streamed in 1 second. It is supposed that the RLC layer can only produce data packets of a maximum size of 5 KB. Therefore 5 KB can be transmitted in a *T* slot. Let us assume that the duration of a *S* slot is 6 ms and the duration of a *B* slot is 2 ms. The sum of durations of 1 *S* and 2 *B* slots is then 10 ms. Assuming that synchronization (in a *S* slot) is necessary every 100 ms, the duration of a TDMA frame is 100 ms. Finally it is assumed that at least 1 *T* slot is allocated to station 2, and at least 1 *T* slot is allocated to station 3, in the TDMA frame.

1000 KB must be streamed in 10 cycles of the TDMA frame. Therefore 100 KB must be streamed in one cycle. 5 KB are transmitted in 1 *T* slot so there must be 20 *T* slots in a cycle allocated to the transmission of station 1. There are therefore 22 *T* slots in a TDMA frame (at least 2 *T* slots are allocated for the other stations in the frame).

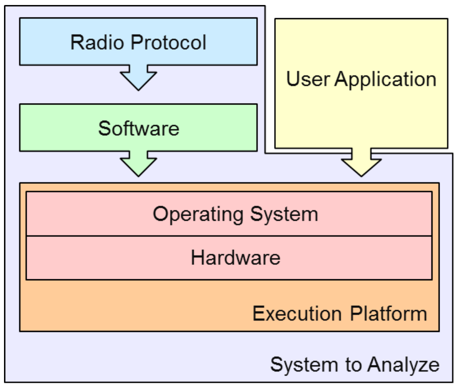


Figure 3.6: Execution Environment of a Software Radio Protocol

Since the TDMA frame has a duration of 100 ms, and T and B slots take 10 ms, there are 90 ms allocated to T slots. The duration of a T slot is then $\lfloor 90/22 \rfloor = 4$ ms.

In conclusion, a TDMA frame with 22 T slots, 20 T slots allocated to station 1, and a T slot duration of maximum 4 ms, will guarantee the application constraint of streaming 10 FPS.

Data packets to send in T slots are fetched by software tasks in MAC. In the example, the tasks in the MAC layer of *station* 1, which fetch the data packets to send in a T slot, have a deadline of 4 ms.

This example shows how a typical application constraint is translated into a typical time constraint of tasks scheduling. In the next section some characteristics of radio protocols developed at Thales will be described. This will define the requirements for the scheduling analysis of such a system.

3.2 CONTEXT OF WORK

The previous section showed how a radio protocol is designed. This section presents one possible implementation by Thales, called a software radio protocol [99]. A software radio protocol defines the system analyzed throughout this thesis.

The assumptions on the system are first described through the description of its software and execution platform. The architecture of the system is then compared two typical architecture designs. The characteristics to consider for scheduling analysis are then summarized. Finally we will see how the system is developed.

3.2.1 *Software and Execution Platform Architecture*

Traditionally functionalities of a radio protocol are implemented as dedicated hardware (e.g. Application-Specific Integrated Circuit (ASIC), Field-Programmable Gate Array (FPGA)). There is no concurrency to access these computing resources. Scheduling analysis is thus not necessary.

In the case of systems developed at Thales, the majority of the functionalities of the protocol is implemented as software running on a General Purpose Processor (GPP). These systems are thus called software radio protocols.

Figure 3.6 shows how a software radio protocol is integrated into a whole system. The system to analyze is composed of the *radio protocol* implemented by a *software* executing on an *execution platform*, composed of an *OS* and *hardware*. It is to be noted that the *execution platform* is shared with the *user application* of the whole system, but the *software* of the *radio protocol* is of higher priority than the user application.

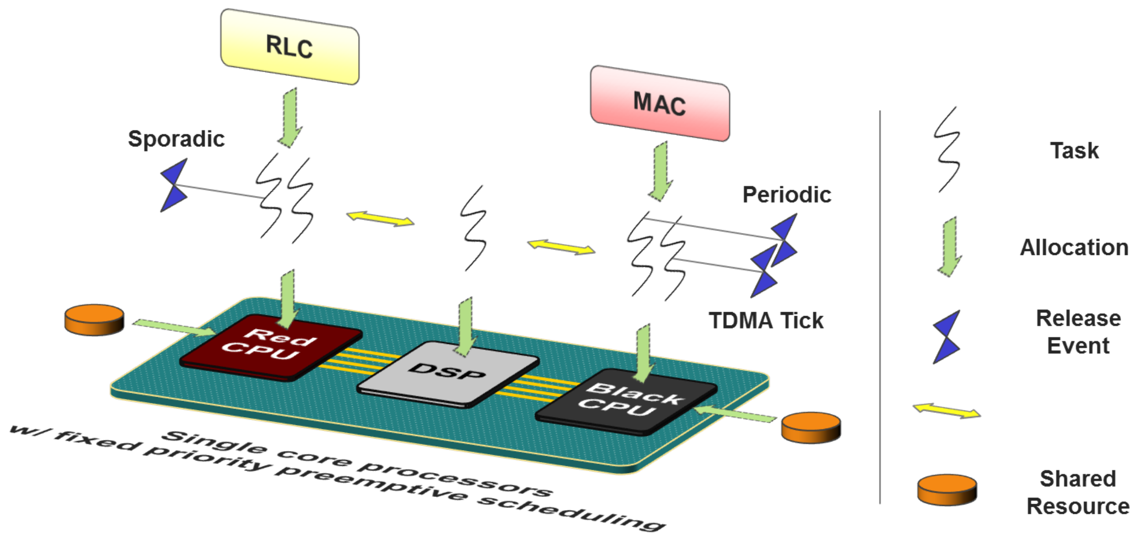


Figure 3.7: Architecture for Scheduling Analysis

The software entities are implemented by some entities in the execution platform: OS and hardware. Figure 3.7 shows an example of these entities. As a reminder, their definitions are given in Chapter 1.

Figure 3.7, shows that the layers of the protocol are implemented by *tasks* allocated on *processors*. The system is a partitioned multiprocessor system. Indeed tasks are scheduled by partitioned scheduling, i.e. a task is allocated on a processor, and it does not migrate. The main reason for choosing this kind of multiprocessor architecture is the security offered by the partitioning: tasks and data are logically and physically separated [39].

Let us see an example of partitioning for security. The *Red CPU* (left most in Figure 3.7) contains encrypted data since it is accessible by the user. The *Black CPU* (right most in Figure 3.7) contains clear data so it should not be accessible (like a black box). Between the two, there is processor for encryption/deciphering of data passed between the two other processors. This middle processor can be one dedicated to signal processing, e.g. *DSP* in Figure 3.7.

Tasks are scheduled by a preemptive FP scheduling policy. The choice of a FP policy is made by engineers at Thales Communications & Security. This policy is well-known by the engineers at Thales, it is less complex to implement [29], and it is available in OS part of real software radio protocol systems.

They may have precedence dependency (e.g. communication through semaphores signaling [28]). They may also use shared resources. Shared resources are assumed local, i.e. two tasks can use a shared resource only if they are allocated on the same processor. Shared resources are protected by a resource access protocol that prevents unbounded priority inversion for uniprocessor [127]. Such protocols can typically be found on execution platforms with the VxWorks [8] OS. This OS is present in certain products developed by Thales.

Some tasks may be released sporadically. For example tasks implementing the IPCS layer are typically released upon arrival of IP packets, which depends on the user application.

Other tasks are released at pre-defined times. This is the case for tasks implementing the MAC layer. In this thesis, activities in MAC are divided in time due to the implementation of TDMA. The releases of a task can follow a periodic pattern or a pattern defined by TDMA slots. Events called TDMA ticks indicate the start of a slot and thus the release of some tasks. Tasks that are released at the start of a slot may also have execution times and deadlines that are constrained by the slot.

In the rest of this thesis, it is assumed that a software radio protocol is based on the TDMA channel access method. The software architecture of the system is then conform to some architectures found in the literature.

3.2.2 *Type of Software Architecture of a Software Radio Protocol*

Two software architectures for RTES have been studied extensively in the literature: event-triggered [70] and time-triggered [71]. In this section the design of a software radio protocol is compared to these two architectures.

3.2.2.1 *Event-Triggered*

An event-triggered system is one where activities are released by *significant events* [70]. A significant event is a change of state in the entities of the system, handled by the computing entities. From an implementation point of view, the signaling of significant events is realized by interrupt mechanisms.

The events may be predictable or *by chance* [70]. For example an entity may be released by a sporadic event or it may wait for the completion of some entity that precedes it, but for which the execution time is not constant. Sporadic and aperiodic tasks are part of an event-triggered system, as well as tasks with precedence dependency implemented with a signaling mechanism.

Because of the unpredictable nature of certain events, online scheduling is often necessary for these kind of systems [70].

3.2.2.2 *Time-Triggered*

In a time-triggered system, instead of waiting for events to occur, entities disseminate their states periodically at fixed times [70]. The periodic sequence of time instants, at which an entity is observed, is called its *observation grid* [70].

As such, the state of an entity is polled by other entities, independently of the activities that occur in the given entity. With this strategy, since the observation grid is the same throughout the whole execution of the system, it is defined and validated offline.

In the case of tasks scheduling, an offline schedule is often defined [70]. During execution, tasks are then released at different time instants, following the offline schedule. Generally the absolute deadline of a task is equal to the next instant when the data it produces will be polled by some other task(s). The tasks set is assessed schedulable offline.

3.2.2.3 *Comparison to Software Radio Protocol*

A software radio protocol is both a time-triggered and an event-triggered system.

Indeed, some tasks released by TDMA ticks indicating the start of a slot. Thus certain tasks are released at pre-defined times, which is conform to the time-triggered architecture.

On the other hand, there are also tasks released by sporadic events, for example upon arrival of IP packets. Due to precedence dependency, a task can also be released by an event produced by a predecessor task.

3.2.3 *Summary of Characteristics for Scheduling Analysis*

From the description of its architecture in Section 3.2.1, the characteristics of a software radio protocol, useful for scheduling analysis, can be summarized in Table 3.1.

In this thesis, it is expected that the scheduling analysis method, proposed for software radio protocols, covers all of the characteristics in Table 3.1.

Table 3.1: Characteristics for Scheduling Analysis

Name	Description
TDMA release	Tasks may be released at pre-defined times, corresponding to start of TDMA slots, according to the TDMA configuration
Periodic and sporadic release	Tasks may be released by periodic and sporadic events.
Arbitrary deadline	Deadlines are arbitrarily defined.
Arbitrary priority	In case of FP scheduling, priorities are arbitrarily defined.
Precedence dependency	Two tasks may have a precedence dependency.
Local Shared resource	Tasks may use local shared resources and thus have critical sections. Shared resources are protected by an access protocol that prevents deadlocks and unbounded priority inversion time.
Individual job parameter	Task execution time and deadline may be constrained by the nature of the release event (e.g. TDMA slot), and may be different from one job to the other. Precedence dependency and shared resource critical sections may also be different from one job to the other.
Preemptive FP policy on uniprocessor	On a uniprocessor, tasks are scheduled according to a preemptive FP scheduling policy.
Partitioned multiprocessor	A task is allocated on a processor and does not migrate. Tasks on different processors may communicate.

3.2.4 *Software Radio Protocol Development*

The development of a software radio protocol at Thales follows the V-model development cycle presented in Section 1.5. There is a will at Thales to integrate scheduling analysis in the verification branch, before coding, at the high level and detailed design steps.

This way the early verification of the architecture can ensure that no non-functional architecture mistakes are made before coding and locate the source of mistakes in the coding step instead of the design steps.

At Thales, the software of a radio protocol is designed with re-usability in mind. Thus the software architecture is divided into several components, with interfaces and services to provide or require. Data structures are also modeled in the architecture.

The MyCCM framework is used to model the software architecture, and then generate code of software components from the software architecture model. Modelers used at Thales are based on Eclipse.

3.3 PROBLEM STATEMENT

Now that the system assumptions of a software radio protocol are presented, the applicability of scheduling analysis is discussed. Integration of scheduling analysis into the development cycle, through an automatic process, is also addressed.

The following sections first discuss issues faced when applying task models to a software radio protocol. Then the availability of tools, where the task models are implemented, is shown. Afterwards the applicability of ADLs will be discussed for software radio protocols. Finally all of the issues presented will be summarized into three problem statements in the last section.

3.3.1 *Applicability of Task Models*

As a reminder, the characteristics of a software radio protocol, that are necessary to consider for scheduling analysis, are listed In Table 3.1.

Let us see which task models, presented in Chapter 2, are applicable to these characteristics. A task model is said applicable to a characteristic, if it is possible to model the characteristic, and if there exists a feasibility or schedulability test for the task model that considers the characteristic.

Table 3.2 shows the applicability of task models to each characteristic. Their applicability is discussed in the following paragraphs. Either the non-applicability of some task models is justified, or the applicability of all task models is shown.

TDMA RELEASE Initial tests in [85, 64] for the fundamental periodic and sporadic task models assume a synchronous system, where tasks are all released at a unique critical instant.

Some works [108, 38] focus on asynchronous periodic and sporadic tasks. The asynchronous release of tasks is not the TDMA release. Tasks are not synchronous but they are released at pre-defined times.

The transaction model, that generalizes the fundamental periodic and sporadic task models, also focus on asynchronous releases of tasks. In [143], Tindell analyzes tasks that are asynchronous, with the transaction model. He takes the example of a network with a bus and messages scheduled by TDMA. His goal was not to analyze the effect that TDMA has on the scheduling of tasks within a single system (e.g. job execution time and release time constrained by the TDMA slot), but end-to-end response times of message transiting in the whole network. For example he computes the response time from the sending of a message by a task, to the transition of the message on the bus, to the reception by another task.

Table 3.2: Applicability of Task Models to Software Radio Protocol: Abbreviations are P/S = Periodic/Sporadic, TR = Transactions, GMF = Multiframe Task Models, DAG = DAG Task Models; N = No, Y = Yes

	P/S	TR	GMF	DAG
TDMA release	N	N	Y	Y
Periodic and sporadic release	Y	Y	Y	Y
Arbitrary deadline	N	Y	N	Y
Arbitrary priority	Y	Y	Y	Y
Precedence dependency	N	Y	N	N
Local shared resource	Y	Y	N	N
Individual job parameter	N	N	Y	Y
Preemptive FP policy on uniprocessor	Y	Y	Y	Y
Partitioned multiprocessor	N	Y	N	N

Some tests for transactions in [106, 118] also consider a release pattern where a task is immediately released by its predecessor, instead of pre-defined times.

Finally some works have been done on systems that are both event-triggered and time-triggered, like a software radio protocol. In [147], the authors work on an automotive system respecting the design of both architectures. Their work does not focus on task scheduling, but network scheduling. Indeed, their objective is to optimize the TDMA slot duration, when a bus is accessed by several processors through the TDMA channel access method.

PERIODIC AND SPORADIC RELEASE All task models support tasks released by a periodic or sporadic event [85, 143, 14, 12].

ARBITRARY DEADLINE Tests for the periodic and sporadic task models assume $D_i \leqslant T_i$ [85, 64]. Thus the deadline is constrained.

Tests [14, 138] for GMF assume the *l-MAD* property (Property 2) or the *Frame Separation* property (Property 3). These properties constrain the deadline of frames and thus tasks.

ARBITRARY PRIORITY There exists at least one test for each task model that supports this characteristic [64, 105, 138, 10]. On the other hand, it is to be noted that tests for sporadic DAG tasks, on a global multiprocessor, assume DM scheduling [12, 24].

PRECEDENCE DEPENDENCY Analysis of the periodic and sporadic task models do not fully handle precedence dependencies, until they are first generalized by the transaction model, and then dynamic offsets are proposed in [105].

Some works [34, 5, 143, 50] propose a method to schedule precedence dependent periodic or sporadic tasks. These methods either constrain the fixed priorities of tasks, or their deadlines, or they are proposed for EDF. Therefore they are not applicable due to the arbitrary priorities, arbitrary deadlines, and FP scheduling policy of a software radio protocol.

Some works [51] also focus on periodic tasks with precedence dependencies that are not specified for every job, but every $2, 3, ..., n$ jobs. These works are not applicable to a software radio protocol, because precedence dependencies, among task jobs, do not follow this kind of pattern.

The multiframe and DAG task models assume independent tasks [14, 24], therefore precedence dependencies are not supported.

LOCAL SHARED RESOURCE Like precedence dependencies, shared resources are not supported by the classical multiframe task models and DAG task models, since they assume independent

Table 3.3: Methods of Tests in Tools: Method abbreviations are WCBT = worst case blocking time, CPU = processor utilization, RTA = response time analysis, DBF = demand bound function, SPD = speedup bound; N = No, Y = Yes

	WCBT	CPU	RTA	DBF	SPD
Rapid-RMA [61]	Y	Y	N	N	N
Cheddar [129]	Y	Y	Y	N	N
MAST [57]	Y	Y	Y	N	N
SymTA/S [59]	Y	Y	Y	N	N
Rubus-ICE [101]	Y	Y	Y	N	N

tasks [14, 24]. In [47] the authors propose an optimal shared resource access protocol for GMF tasks but the protocol is not implemented in the OS of Thales products.

INDIVIDUAL JOB PARAMETER The modeling of jobs with different parameters is only supported by the multiframe and DAG task models [14, 10, 24], among those present in Table 3.2.

PREEMPTIVE FP POLICY ON UNIPROCESSOR All task models have tests for the preemptive FP scheduling policy [85, 143, 14, 24].

PARTITIONED MULTIPROCESSOR The periodic, sporadic task models, and the multiframe task models have tests for a uniprocessor system [85, 14]. The DAG task models has tests for uniprocessor systems [10], or global multiprocessor systems [24].

In conclusion, the results of Table 3.2 show that none of the task models presented in Chapter 2 support all characteristics of the software radio protocol to analyze.

3.3.2 *Availability of Analysis Methods Implemented in Tools*

The availability of tools that implement the selected task model, or their analysis methods, is also to be considered. Table 3.3 shows some methods implemented in some scheduling analysis tools. A system is described in the modeling language of a tool, before the scheduling analysis methods can be applied. The tools are described in the following paragraphs.

RAPID-RMA [61] is a set of modeling and scheduling analysis tools. The architecture is modeled with components. Design scenarios are then modeled for scheduling analysis. A design scenario indicates the interactions between the components. Rapid-RMA uses tests for RM scheduling to determine schedulability but it also provides a simulator.

CHEDDAR [129] is a scheduling analysis tool with feasibility, schedulability tests, and a simulator. In Cheddar, the architecture of the system is modeled with the tool's own ADL called Cheddar-ADL. Cheddar implements tests [85, 64] for the periodic and sporadic task models. Although the transaction model is not implemented, Cheddar offers the possibility to express the offset, and jitter parameters. Furthermore the test in [143] is implemented but without using the concept of transaction.

MAST [57] is a modeling and analysis suite for real-time applications. In the MAST, an architecture is modeled with a modeling language based on events. Events are sent between tasks that have precedence dependency. Tasks are allocated on processors and they may use shared resources. MAST then transforms the event-based architecture model to transactions for scheduling analysis.

Schedulability tests for transactions can then be applied. The tests in [143, 105, 105] are implemented.

SYMTA/S [59] is a scheduling analysis tool originally dedicated to the automotive industry. As such, SymTA/S has entities based on those found in automotive systems. The architecture is modeled as components allocated on buses and processors. Components have ports through which they receive and send event streams. SymTA/S thus uses an event stream propagation model for scheduling analysis. Although it does not implement the transaction model, the same approach of jitter propagation is used in the RTA method implemented in the tool.

RUBUS-ICE [101] is a tool suite for model-driven development of real-time systems, with modelers, code generators and analysis methods. The architecture is modeled in the Rubus Component Model (RCM) language. In RCM, software functions are modeled as components that communicate through a producer-consumer scheme. Time parameters are extracted from the component-based model and scheduling analysis methods can then be applied. RTA schedulability tests have been implemented as plug-ins in Rubus-ICE.

As shown by Table 3.3, most of these tools implement tests based on RTA or processor utilization. None of the tools actually implement a test, or equivalent, for the multiframe and DAG task models because these tests are based on demand bound or speedup bound.

Furthermore some tools implement RTA methods for the release pattern of the linear transaction model. As we will see in this thesis, these methods are not sufficient for the work of this thesis.

3.3.3 *Applicability of ADLs*

Chapter 1 exposed some generic and domain-specific ADLs to describe a system's architecture. This section shows how these ADLs have been exploited for scheduling analysis.

MARTE There exists a number of tools that support directly MARTE for scheduling analysis. MARTE is implemented in IBM's RSA modeler. The modeler has a plug-in for Rapid-RMA [61]. Thales Research & Technology have also experimented by transforming RSA models to Cheddar for scheduling analysis [88]. Since Rapid-RMA and Cheddar implement feasibility tests for periodic tasks scheduled by RM or DM, the architectural entities of MARTE exploited by the tool are tasks modeled with specific stereotypes of MARTE.

MARTE is also implemented in the Papyrus [104] modeler, a open-source UML modeler part of the Eclipse Modeling Framework (EMF). There exists a plug-in for Eclipse that transforms a MARTE model made with Papyrus, to a model in MAST for scheduling analysis with the tool [96]. The transformation exploits UML activities with MARTE stereotypes applied. The UML activity actions are allocated on tasks, modeled as components with MARTE stereotypes applied on them.

AADL There exists several works on scheduling analysis of a system described with AADL. Some of these works are based scheduling analysis of AADL models with model checking tools. The AADL model is transformed to some model used by the model checking analysis method of a tool. The output models of the transformation are either petri nets [119] (Tina tool [20]), synchronous language models (Polychrony tool [87]), timed automata (UPPAAL tool [18]), or real-time process algebra [132] (VERSA tool [37]).

Other works are based on scheduling analysis of AADL models with schedulability and feasibility analysis tools. In [46], a system modeled in AADL is analyzed with Cheddar. The AADL model is created in STOOD, developed by Ellidiss Software, and then analyzed with AADL Inspector which includes the analysis core of Cheddar.

EAST-ADL Some works have been done to offer the capability of performing scheduling analysis on EAST-ADL models. EAST-ADL has been extended with MARTE in [1] for scheduling analysis with the MAST tool. In [115, 48] EAST-ADL models are transformed to timed automatas for time constraints verification with UPPAAL, the model checking tool. As a reminder EAST-ADL is an ADL for the AUTOSAR standard. In [69], the authors analyze the schedulability of a system conform to AUTOSAR, with the SymTA/S [59] tool.

MYCCM The MyCCM framework is developed by Thales. Ongoing projects aim at integrating a scheduling analysis solution. This will either be done by developing a proprietary scheduling analysis tool of Thales, or by extending MyCCM models and transforming them to the model of an existing scheduling analysis tool.

In conclusion, MARTE, AADL, and EAST-ADL, through AUTOSAR, have been exploited for scheduling analysis with the feasibility and schedulability test approach. These ADLs are not specific to a software radio protocol and they are not used to describe its architecture at Thales Communications & Security. MARTE is generic to the domain of RTES but it lacks modeling guidelines and clearly defined semantics [62, 114] which handicaps its adoption by engineers.

3.3.4 *Summary of Problem Statement*

In the previous three sections, three problems were determined for scheduling analysis of software radio protocols. These problems are summarized as follows:

– Applicability of task models: No task model is applicable to all characteristics of a software radio protocol, and it is not obvious to choose one for extension.
– Availability of analysis methods implemented in tools: Some existing task models and their analysis methods are not implemented in scheduling analysis tools. Some of these task models that are not implemented, can be chosen for extension in this thesis.
– Applicability of ADLs: There is not a domain-specific ADL for software radio protocols, that has been tested for scheduling analysis. The generic modeling language MARTE also lacks modeling guidelines and clearly defined semantics.

3.4 OVERVIEW OF THE SOLUTION

The objective of this thesis is to perform scheduling analysis of a software radio protocol. To achieve the objective, solutions are proposed for the problems stated in the last section. The solutions are illustrated in Figure 3.8.

The general solution consists in first proposing a new task model called *Dependent General Multiframe (DGMF)*, adapted for characteristics of a software radio protocol. This solves the problem of *applicability of task models*. The new task model is then implemented in a *scheduling analysis tool*. This solves the problem of *availability of analysis methods implemented in tools*. Finally an experimental *architecture model*, in UML MARTE, is proposed to perform automatically scheduling analysis by *transforming* it to DGMF tasks. This solves the problem of *applicability of ADLs* for scheduling analysis of a software radio protocol.

The solutions can be implemented with some existing tools. Some examples of implementation are: the DGMF task model and its analysis method can be implemented in the *Cheddar scheduling analysis tool*; the UML MARTE architecture model can be made with the *Papyrus modeler* of Eclipse; the transformation of the architecture model to the task model can be implemented as an *Eclipse plug-in*.

The next three sections describe in detail the proposed solution for each problem. First the solution to the applicability of task models problem is described. The the solution to the availability of

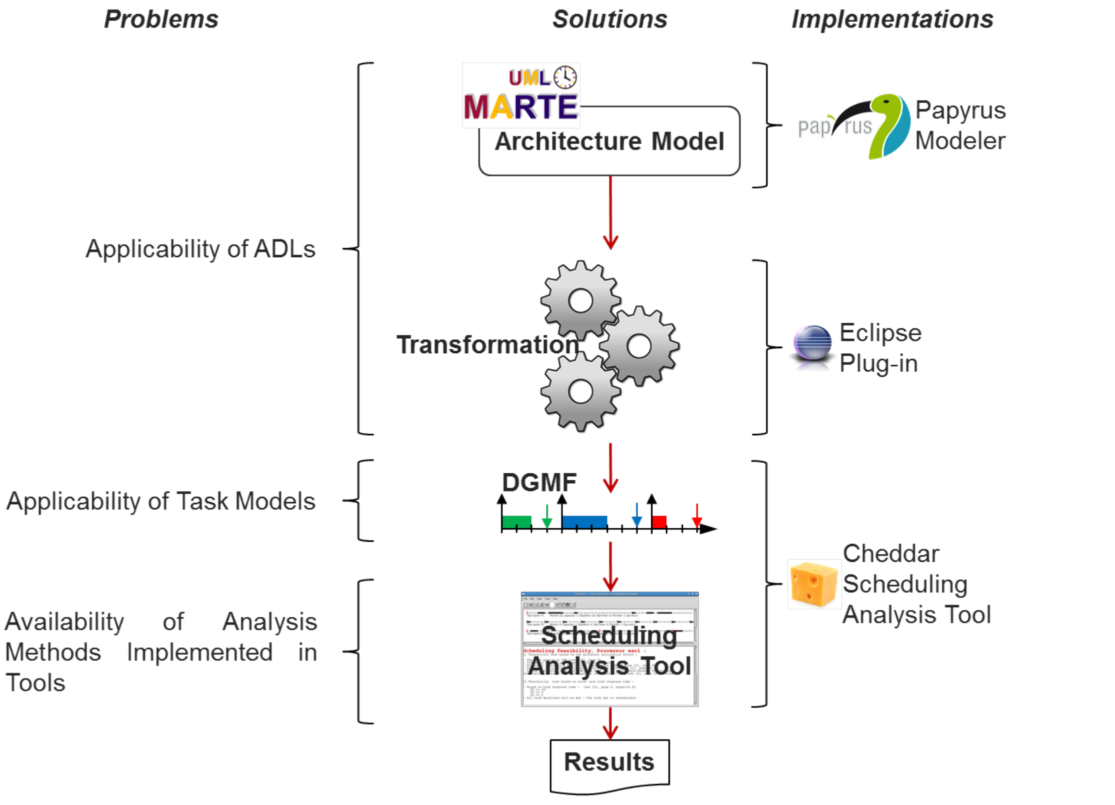


Figure 3.8: Overview of the Solution: Solutions in the middle related to problems on the left, implemented by tools on the right

analysis methods implemented in tools problem is presented. Finally the solution to the applicability of ADLs problem is shown.

3.4.1 *Applicability of Task Models Problem*

First a task model needs to be proposed for software radio protocols. The solution is to extend a task model among those in Table 3.2. The choice of the task model to extend must consider its expressiveness, analysis difficulty, and the impact of characteristics of the software radio protocol on the analysis results.

The task model to extend is chosen through an experiment presented in this thesis. The experiment applies the fundamental periodic task model for schedulability analysis of a simplified software radio protocol.

Through this experiment the characteristic of the system, that has the most impact on pessimism of response times for schedulability analysis, is determined. This characteristic is individual job parameters.

The multiframe task models, and the DAG task models, are expressive enough to model individual job parameters of a task.

The GMF task model is motivated by behaviors of tasks in the multimedia domain. Jobs of a task have an execution time, deadline, and minimum separation time to the next job release, that depend on the type of video image being processed by the task. This kind of behavior also occurs in a software radio protocol. Thus the GMF task model is expressive enough for the aspect of individual job parameters. On the other hand, GMF does not handle task dependencies, and it is not applicable to a partitioned multiprocessor system.

This thesis thus proposes the DGMF task model to extend GMF. DGMF extends GMF with task dependencies. DGMF is also applicable to a partitioned multiprocessor system.

The DGMF analysis method consists in first transforming DGMF tasks to transactions. This approach is chosen because transactions support task dependencies, and they are applicable to a partitioned multiprocessor system. An initial independent GMF tasks to transaction is also proposed in [116]. After the transformation, existing schedulability tests for transactions can be exploited, or extended if necessary.

As we will see, the transformation faces the issue of the difference in semantic between the two models. Indeed, transactions represent tasks related by collectively performed functionalities and timing attributes [143], not individual jobs of a task. GMF, on the other hand, express the individual job parameters through frames.

Due to the difference in semantic, schedulability tests for transactions must be also adapted or results are pessimistic. Therefore this thesis proposes a schedulability test for transactions that may be the result of DGMF transformation. As we will see, these transactions are tree-shaped and they have tasks that are not necessarily released immediately by their predecessor. The test is called WCDOPS+NIM and it extends the WCDOPS+ test in [118].

Since DGMF is a new task model, the model and its analysis method must be implemented in a scheduling analysis tool.

3.4.2 *Availability of Analysis Methods Implemented in Tools Problem*

The GMF task model is not implemented in a scheduling analysis tool. Therefore in this thesis, task models GMF, DGMF, transaction, transformation of DGMF to transaction, and transaction schedulability tests, are all implemented in Cheddar. This thesis shows how the tool's framework is extended for this purpose. Once the tool is extended, its model must be produced from some architecture model of a software radio protocol described with an ADL.

3.4.3 *Applicability of ADLs Problem*

MyCCM is used at Thales to describe the architecture of a software radio protocol but the framework has not been tested for scheduling analysis with tools. An ADL among MARTE, AADL, and EAST-ADL must thus be chosen to describe the system architecture.

For EAST-ADL, schedulability analysis is supported by SymTA/S for which an open-source version is not yet available. Therefore the choice of an ADL comes down to MARTE and AADL. Table 3.4 compares UML MARTE to AADL according to some criterias.

Considering the criterias in Table 3.4, the MARTE profile for UML is chosen for the modeling of a software radio protocol, for the following reasons:

– MARTE is not domain-specific.
– MARTE benefits from the UML extension mechanisms and extension for scheduling analysis has no impact on the design model that may be used for code generation.
– MARTE can trace back analysis results to the entities of the architecture model, which is important for iterative engineering work.
– MARTE can be transformed to a MyCCM model used at Thales Communications & Security for code generation.

Since one of the main issues with MARTE is the lack of modeling guidelines and clearly defined semantics, this thesis proposes a model called Service Model and its mapping to MARTE concepts. The model is made with the open-source Papyrus implementation of MARTE.

The Service Model contains architectural entities as well as behavior information. The behavior information are the behavior of services of a software radio protocol. The model is transformed, via an Eclipse plug-in, to a Cheddar model for scheduling analysis.

Table 3.4: Architecture Description Language Comparison

Criteria	Generic ADL: UML MARTE	Domain-Specific ADL: AADL
Semantics: Do entities have clearly defined semantics?	AADL was originally dedicated to the avionic domain so its entities have clearly defined semantics since they were limited to this domain originally.	MARTE is a UML profile for generic RTES. A same attribute of a stereotype can have different signification, according to the domain of the modeled RTES.
Modeling guidelines: Are there clearly defined modeling guideline?	AADL was dedicated to the avionic domain originally. Therefore modeling guidelines exist for engineers of this domain. Guidelines are eased by the clearly defined semantics of AADL.	MARTE lacks modeling guidelines [114, 62], which is an issue when the semantics of its entities can have different significations, according to the domain being modeled.
Extensiveness: Is the ADL extensible if necessary?	AADL can be extended with annexes and property-sets of the modeling language. But the extension may have an impact on the design models. In case of extensions for scheduling analysis, the design model may then depend on the scheduling analysis tool [151].	MARTE being an UML profile, benefits from the extension mechanisms of UML. It can thus be extended either through the extension of its sub-profiles, or the completion by other profiles. For example in [2] the MARTE profile is extended for power management of RTES. Extending MARTE for analysis has no impact on design, due to the separation of design and analysis models in the profile.
Traceability: Is there a way to trace analysis results back to their entities in the architecture model?	Currently there is no solution to trace back analysis results but the feature may be added as an extension of AADL.	The stereotypes of MARTE are designed to trace back analysis results to the entities of the model.
Applicability: Is it applicable to software radio protocols?	AADL was originally dedicated to the avionic domain.	MARTE has been experimented on the modeling of software radio protocols in [60]. The work in [60] is based on early works of this thesis, presented in later chapters.
Integration at Thales Communications & Security: Is a model described with the ADL easily integrated into the Thales development cycle?	AADL is currently not used at Thales Communications & Security.	There exists a MARTE to MyCCM transformation plug-in for Eclipse, developed by Thales.
Tool support: Is the ADL supported by a large number of modelers? By modelers used at Thales?	A common way to use AADL with Eclipse is through the OSATE plug-in.	MARTE being an UML profile, benefits for the large eco-system of UML modelers based on Eclipse, including several tools (like RSA) used at Thales.

3.5 CONCLUSION

In this chapter the radio domain was first presented. Radios communicate in a network which may be for example a MANET. In order to communicate, the radios must respect a radio protocol that defines the rules of communication. Traditionally radio protocols are implemented as hardware but at Thales Communications & Security, they are implemented as software. The architecture of a software radio protocol was presented, focusing on radio protocols that use a TDMA channel access method. We saw how this kind of system is a RTES that needs scheduling analysis.

To perform scheduling analysis, some characteristics of the system were described and summarized in Table 3.1, thus defining the assumptions on the system to analyze. Scheduling analysis is also to be integrated into the development cycle of a software radio protocol. The development cycle at Thales follows the V-model. Software development also exploits models for code generation, through the MyCCM framework.

Several issues were observed when studying the applicability of scheduling analysis to a software radio protocol. First, no task model is applicable to all characteristics of a software radio protocol, and it is not obvious to choose one for extension. Second, some existing task models and their analysis methods are not implemented in scheduling analysis tools. Some of these task models, not implemented, can be chosen for extension in this thesis. Third, there is not a domain-specific ADL for software radio protocols, that has been tested for scheduling analysis. The generic modeling language MARTE also lacks modeling guidelines and clearly defined semantics.

The approach to solve these problems is an analysis method that consists in first modeling a software radio protocol in UML MARTE, in a model proposed by this thesis called Service Model. The Service Model contains architectural entities but also behavior information. The UML MARTE model is transformed to a task model adapted for software radio protocols. The task model is called DGMF. It extends the GMF task model with task dependencies and partitioned multiprocessor scheduling. The analysis method of DGMF is implemented in Cheddar, a scheduling analysis tool.

Several technical contributions are necessary to obtain the final scheduling analysis method for a software radio protocol. These contributions are presented throughout the chapters of this thesis. The next chapter focuses on the experiment that evaluates the viability of scheduling analysis on a software radio protocol.

Part II

CONTRIBUTIONS

Chapter 4

EXPERIMENT ON SCHEDULING ANALYSIS OF SOFTWARE RADIO PROTOCOL

The previous chapters showed that scheduling analysis needs to be applied to software radio protocols. Scheduling analysis must also be applied automatically through transformation of architecture models and usage of scheduling analysis tools.

Chapter 3 showed that there is not an applicable task model for scheduling analysis of a software radio protocol. There is also an issue of applicability of ADLs for automatic scheduling analysis of such systems.

This chapter presents an experiment where the periodic task model is applied to a simplified software radio protocol. To apply the task model for scheduling analysis, an automatic scheduling analysis approach is used.

The periodic task model is thus to be applied by producing automatically a tasks set from an architecture model, described with basic entities of UML MARTE. The tasks set is analyzed by the Cheddar tool in which the periodic task model is already implemented.

Experimental results will confirm that the periodic model is not suited for a software radio protocol implementing TDMA. The observations of the experiment will then help choose a task model to extend, among those compared in Section 3.3.1.

The following sections first present the case-study of the experiment. Then the case-study is modeled in UML MARTE and transformed to a Cheddar-ADL model so the analysis of the case-study can be performed. Finally the analysis results are discussed.

4.1 CASE-STUDY OF THE EXPERIMENT

The case-study for the experiment is a software radio protocol called Highly Dynamic Radio Network (HDRN), developed by Thales as a demonstrator. HDRN is integrated below the IP stack of a communication system. The radio protocol is based on TDMA. It is a simplified software radio protocol that does not represent the majority of products developed at Thales. It was chosen for the experiment because its simplified architecture makes the periodic task model applicable.

The protocol stack is designed according to Figure 4.1. The wrapper code of the software is generated through a MyCCM model.

Under the experimental conditions, the system behavior is the following:

1. IP packets arrive from the IP stack and are segmented into RLC packets before being stored in the RLC layer.
2. RLC packets are segmented into MAC packets before being stored in the MAC.
3. A TDMA tick from a synchronization source indicates the start of a new slot.
4. The current slot type and the next one are checked according to the TDMA configuration.

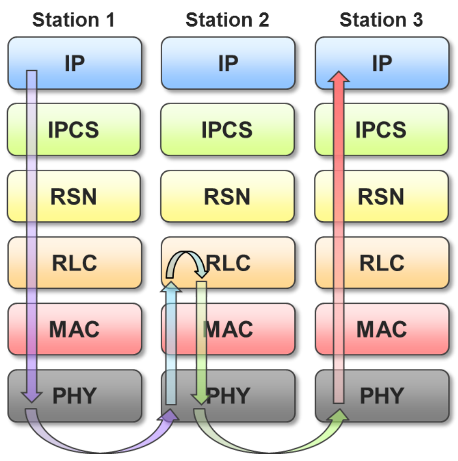


Figure 4.1: Communicating HDRN Layers: Arrows show the communication

5. Packets stored in MAC are passed to PHY for transmission in the next transmission slot.
6. Received packets are assembled and transferred all the way from PHY to IPCS.
7. The system is prepared for the next slot.

The protocol is implemented by five tasks and 6 shared resources. These entities are illustrated in Figure 4.2. The fives tasks are: *IPPacketSendingTask*, *RLCPDUSendingTask*, *CommunicationManagementTask*[1], *TickObserverTask*[2], and *DwellReceiverTask*. The shared resources are: *PDUsToSendFifo*, *Queue*, *TDMAStructure*, *NextSlot*, *SlotProcessQueue*, *RxDataBuffer*. Shared resources *PDUsToSendFifo*, *Queue*, *SlotProcessQueue*, and *RxDataBuffer* are resources where data are stored. Shared resources *TDMAStructure* and *NextSlot* are resources that indicate the state of the system.

The behavior of the tasks is the following:

– IPPacketSendingTask: This task is released by arriving IP packets from the IP stack. It puts them into the *PDUsToSendFifo*.
– RLCPDUSendingTask: This task is released by the *IPPacketSendingTask*, each time there are packets in the *PDUsToSendFifo*. It segments the packet and stores the segmented packets in the *Queue*.
– CommunicationManagementTask: This task is released by a TDMA tick. It processes the current slot and then checks the *TDMAStructure* for the TDMA configuration. Afterwards it updates *NextSlot* with the next slot identifier, and wakes the *TickObserverTask* task. On reception it stores received packets in the *RxDataBuffer*.
– TickObserverTask: This task is released by the *CommunicationManagementTask*. It reads *NextSlot* and configures PHY for the next slot. It also transmits the data to be send over air on the next slot (stored originally in the *Queue*) to the *SlotProcessQueue*. The *TDMAStructure* is checked for slot consistency. These operations have to be done before the next slot.
– DwellReceiverTask: This task is released by the *CommunicationManagementTask*, each time there are data in the *RxDataBuffer*. It processes the data and sends it all the way to the IP stack. The *TDMAStructure* is checked for slot consistency.

The HDRN tasks are implemented by POSIX threads [28] with properties SCHED_ FIFO and PTHREAD_SCOPE_SYSTEM. The properties ensure that the tasks are scheduled with a preemptive FP scheduler, and that they can only be preempted by other SCHED_FIFO tasks and OS tasks.

The tasks are allocated on a same core of a quad-core processor. The core is dedicated to the tasks of HDRN. Only OS kernel tasks still persist on the core.

1. May be abbreviated as "CommMgtTask" or "CommunicationMgtTask".
2. May be abbreviated as "TickObs".

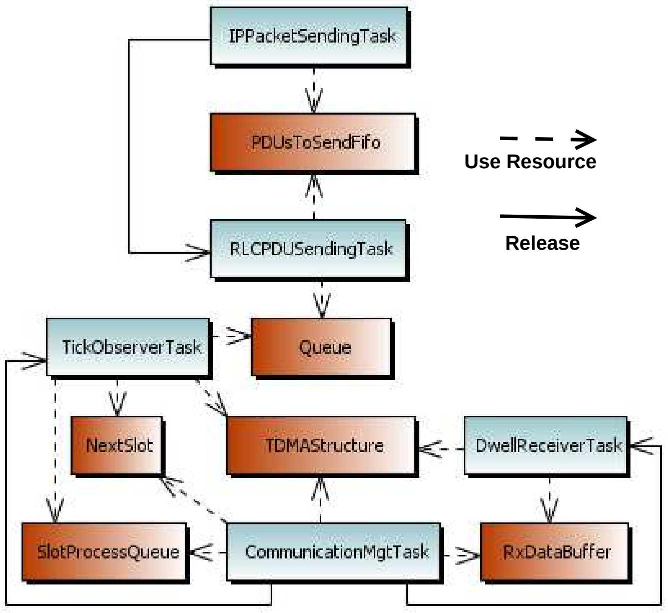


Figure 4.2: HDRN Tasks and Shared Resources: Plain arrows are precedence dependencies; Dashed arrows are shared resource usages

The scheduling parameters of the tasks are shown in Table 4.1. Table 4.2 shows how tasks use shared resources. The shared resources are protected by the default POSIX mutex [28], i.e. there is no protection against unbounded priority inversion.

The priorities of tasks of HDRN were assigned by software engineers, based on experience and knowledge from the development of other software radio protocols. The scheduling parameters in Table 4.1 show that the priority assignment is done such that a predecessor task has a higher priority than a successor task. This priority assignment was initially proposed in [5]. With this priority assignment, the tasks can be analyzed with the periodic task model, even when they have precedence dependencies.

Time values in the tables come from real executions of instrumented code. Each instrumentation point is used to measure a specific time value. A time value is then the maximum of values measured at its instrumentation point. Some time values were measured more than 7500 times to get the maximum. The times values are thus maximum values observed during a real execution. The accuracy of the measurements, like the WCET, is not the purpose of this thesis.

The setup of the execution is shown in Figure 4.3. Three PCs are connected through a central hub. *PC-1* and *PC-2* each run a HDRN protocol while *PC-3* (called the "Communication Server") plays the role of the synchronization source. It broadcasts ticks to *PC-1* and *PC-2*. *PC-3* also manages data routing. This means *PC-1* and *PC-2* send their packets to *PC-3* which redirects the packets to the correct station. For the experiment the a video is streamed between *PC-1* and *PC-2*.

In order to apply the periodic task model analysis on the case-study, it must be modeled. The next section presents the modeling approach.

4.2 APPLYING THE PERIODIC TASK MODEL

To apply automatically the analysis of the periodic task model on HDRN, the approach is to first model the system architecture in UML MARTE. The architecture model is then transformed to the periodic task model implemented in Cheddar. The analysis can then be performed.

The UML MARTE model is based on existing MARTE models that can be transformed to MyCCM models for development. The existing MARTE models are extended so they contain enough infor-

Table 4.1: HDRN Task Parameters: Priorities in highest priority first order

Task	Parameter	Value
IPPacketSendingTask	Release	Sporadic
	Min. inter-arrival	6000 µs
	Priority	20
	WCET	207 µs
	Deadline	None
RLCPDUSendingTask	Release	Sporadic
	Min. inter-arrival	6000 µs
	Priority	10
	WCET	1725 µs
	Deadline	None
TickObserverTask	Release	Periodic
	Period	5000 µs
	Priority	10
	WCET	773 µs
	Deadline	5000 µs
DwellReceiverTask	Release	Sporadic
	Inter-arrival	10000 µs
	Priority	10
	WCET	917 µs
	Deadline	10000 µs
CommunicationManagementTask	Release	Periodic
	Period	5000 µs
	Priority	40
	WCET	1115 µs
	Deadline	5000 µs

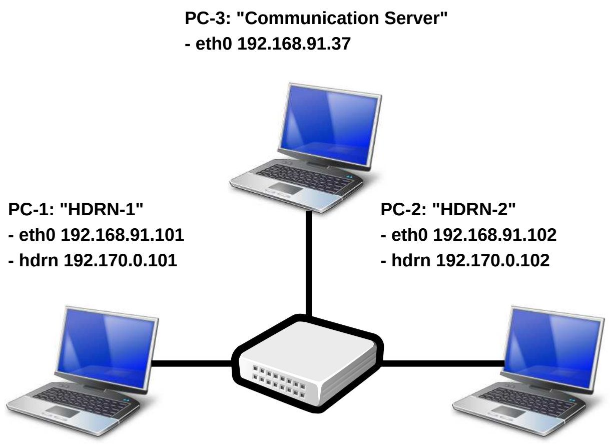


Figure 4.3: Experiment Setup

Table 4.2: HDRN Shared Resources: Start and end times are those of critical sections

Resource	Used by	Start (μs)	End (μs)
PDUsToSendFifo	IPPacketSendingTask	30	34
	RLCPDUSendingTask	1	3
Queue	RLCPDUSendingTask	127	1624
	TickObserverTask	16	142
TDMAStructure	DwellReceiverTask	65	67
	TickObserverTask	4	6
	CommMgtTask	109	111
RxDataBuffer	DwellReceiverTask	1	27
	CommMgtTask	93	107
NextSlot	TickObserverTask	0	772
	CommMgtTask	329	330
SlotProcessQueue	TickObserverTask	84	90
	CommMgtTask	0	325
	CommMgtTask	708	758

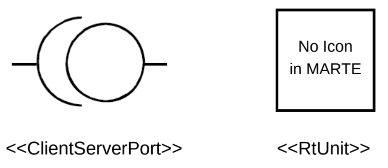


Figure 4.4: Applied GCM and HLAM Stereotypes

mation to be transformed into periodic task models for scheduling anlaysis. The extensions are done with respect to the original models.

The following sections first present a high level software architecture model. This model is the original MARTE model. Then a software and execution platform architecture model is exposed. Afterwards the associations between the different entities are shown. The software and execution platform architecture model, and the associations, represent the extension of the original model. Finally the transformation to Cheddar is exposed.

4.2.1 *High Level Software Architecture Model*

The Generic Component Model (GCM) and High Level Application Model (HLAM) sub-profiles of MARTE are used to model some high level software entities.

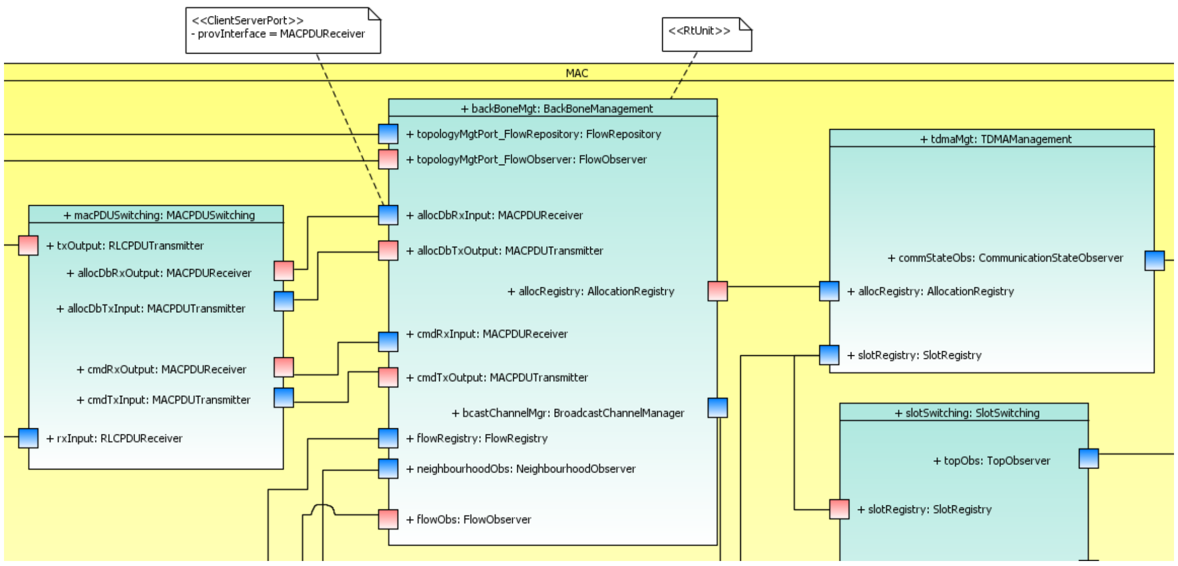
The GCM sub-profile fulfills designer needs to model abstract and generic components. HLAM is a sub-profile that provides designers with facilities to represent high level entities such as set of services. Figure 4.4 illustrates the stereotypes in the GCM and HLAM sub-profiles that are used.

Table 4.3 maps the different high level software architecture entities, of a software radio protocol, to their stereotyped UML meta-classes in MARTE. As a reminder, the entities are those of an architecture model and they are defined in Section 1.7.

As an example, Figure 4.5 gives a partial view of the high level software architecture of HDRN's MAC layer. For example *BackBoneManagement* is a component stereotyped <<RtUnit>>, and *allocDbRxInput* is a port stereotyped <<ClientServerPort>> with *provInterface* attribute tagging the *MACPDUReceiver* interface class.

Table 4.3: High Level Software Architecture Model in UML MARTE

Entity	**UML MARTE**
Component	**Component** stereotyped <<RtUnit>> from HLAM
Interface	**Port** stereotyped <<ClientServerPort>> from GCM with *provInterface* or *reqInterface* attribute of <<ClientServerPort>> tagging the **interface** that contains services provided or required by the component.
Service	**Operation** of the **interface**
Connector	**Connector** from UML



<<ClientServerPort>>
- provInterface = MACPDUReceiver
<<RtUnit>>
MAC
+ backBoneMgt: BackBoneManagement
+ topologyMgtPort_FlowRepository: FlowRepository
+ topologyMgtPort_FlowObserver: FlowObserver
+ allocDbRxInput: MACPDUReceiver
+ allocDbTxOutput: MACPDUTransmitter
+ allocRegistry: AllocationRegistry
+ cmdRxInput: MACPDUReceiver
+ cmdTxOutput: MACPDUTransmitter
+ bcastChannelMgr: BroadcastChannelManager
+ flowRegistry: FlowRegistry
+ neighbourhoodObs: NeighbourhoodObserver
+ flowObs: FlowObserver
+ macPDUSwitching: MACPDUSwitching
+ txOutput: RLCPDUTransmitter
+ allocDbRxOutput: MACPDUReceiver
+ allocDbTxInput: MACPDUTransmitter
+ cmdRxOutput: MACPDUReceiver
+ cmdTxInput: MACPDUTransmitter
+ rxInput: RLCPDUReceiver
+ tdmaMgt: TDMAManagement
+ commStateObs: CommunicationStateObserver
+ allocRegistry: AllocationRegistry
+ slotRegistry: SlotRegistry
+ slotSwitching: SlotSwitching
+ topObs: TopObserver
+ slotRegistry: SlotRegistry

Figure 4.5: MAC Layer Model: Rectangles are components; Squares are interfaces with services; Lines are connectors; For readability, not all stereotypes are visible and stereotypes are shown by UML comments

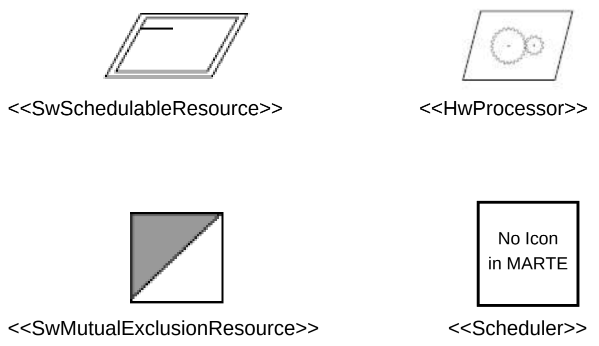


Figure 4.6: Applied GRM, SRM, and HRM Stereotypes

Table 4.4: Execution Platform Model in UML MARTE

Entity	UML MARTE
Task	**Component** stereotyped <<SwSchedulableResource>> from SRM. Task parameters are specified by the attributes of the stereotype.
Memory Partition	**Component** stereotyped <<MemoryPartition>> from SRM
Shared Resource	**Component** stereotyped <<SwMutualExclusionResource>> from SRM. Shared resource access protocol are specified by the attributes of the stereotype.
Scheduler	**Component** stereotyped <<Scheduler>> from GRM. Scheduling parameters are specified by the attributes of the stereotype.
Processor	**Component** stereotyped <<HwProcessor>> from HRM

4.2.2 *Software and Execution Platform Architecture Model*

The General Resource Model (GRM), Software Resource Model (SRM), and Hardware Resource Model (HRM) sub-profiles of MARTE are used to model some refined software and execution platform entities.

The GRM sub-profile provides the foundations needed for a more refined modeling of both software (SRM) and hardware (HRM). The SRM sub-profile offers support for modeling of software entities such as tasks, shared resources and memory partitions. The HRM sub-profile provides several stereotypes to model the platform hardware through three different views: a high-level architectural view, a specialized view, and a detailed physical view. The model of this thesis stays at a high-level architectural view, which contains enough information for the analysis methods of the periodic task model. Figure 4.6 illustrates the GRM, SRM, and HRM stereotypes used in the execution platform model.

Table 4.4 maps the different software and execution platform architecture entities, of a software radio protocol, to their stereotyped UML meta-classes in MARTE. As a reminder, the entities are presented in Section 3.2.1.

As an example, Figure 4.7 shows some software and execution platform entities of HDRN. For example, *CommunicationMgtTask* is a task stereotyped <<SwSchedulableResource>>, *MainProcess* is a memory partition stereotyped <<MemoryPartition>>, *SlotProcessQueue* is a shared resource stereotyped <<SwMutualExclusionResource>>, *HPFScheduler* is a scheduler stereotyped <<Scheduler>>, and *CPU* is a processor stereotyped <<HwProcessor>>.

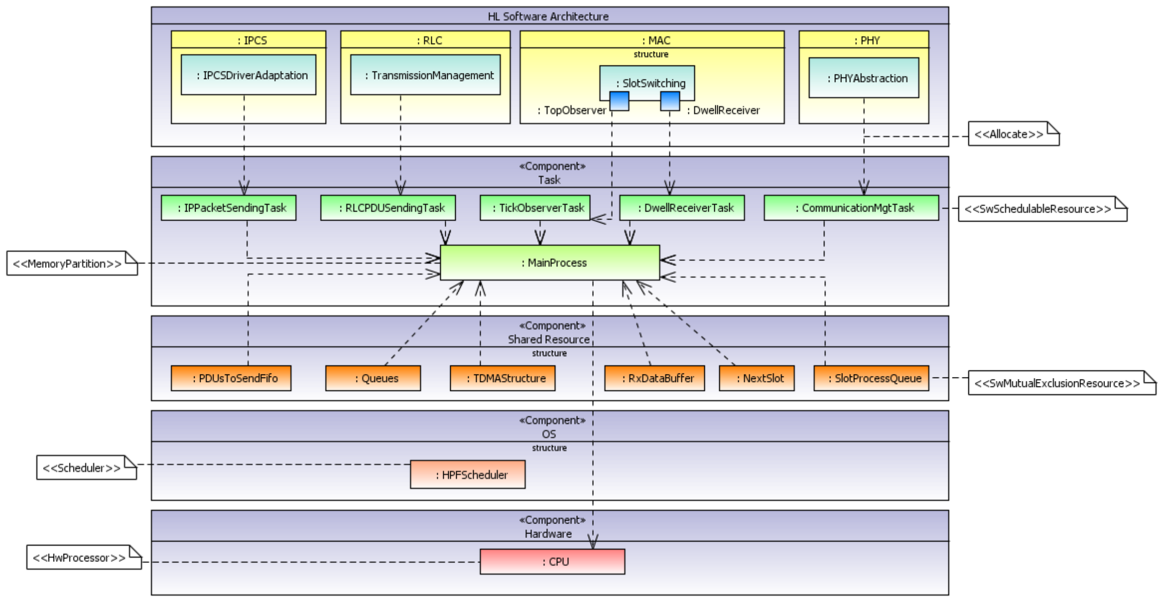


Figure 4.7: Spatial Distribution View: Dashed arrows are allocations; For readability, not all stereotypes are visible and stereotypes are shown by UML comments

4.2.3 *Associations Between Entities*

UML associations, stereotyped with MARTE, are used to represent the relationships between the entities. The associations are modeled in different views.

4.2.3.1 *Spatial Distribution View*

The spatial distribution view shows how entities are allocated. Through the allocation mechanisms, the original MARTE model can be extended without any modifications to it.

An allocation is a abstraction association stereotyped <<Allocate>> from MARTE. The abstraction has a client and a supplier. The client is the entity to be allocated. The supplier is the the entity on which the client is allocated on.

The different possible allocations are listed in Table 4.5. The spatial distribution view associates high level software architecture components and interfaces (ports and operations) to tasks, tasks to memory partitions, shared resources to memory partitions, and memory partitions to processors.

As an example, Figure 4.7 shows a spatial distribution view of HDRN. For example the *PHYAbstraction* high level software component is allocated onto *CommunicationMgtTask*; the *DwellReceiver* interface is allocated onto *DwellReceiverTask*; the *DwellReceiverTask* is allocated onto *MainProcess*; the *SlotProcessQueue* is allocated onto *MainProcess*; the *MainProcess* is allocated onto *CPU*.

4.2.3.2 *Shared Resource Usage View*

The shared resource usage view shows how tasks use shared resources. A resource usage is a usage association stereotyped <<ResourceUsage>> from the MARTE.

The usage has a client and a supplier. The client is the resource user, the supplier is the resource. Through the <<ResourceUsage>> stereotype, it is possible to specify the critical section. A constraint, stereotyped <<TimedConstraint>> from the MARTE, is used to specify the critical section's length. The constraint constrains the usage.

As an example, Figure 4.8 shows the shared resource usages by tasks of HDRN. For example *TickObserverTask* uses *NextSlot*.

Table 4.5: Allocations

Client	Supplier	Description
Component	Task	The high level software component contains a task released by events.
Interface	Task	The services provided through the interface of a high level software component, are handled by a task released by calls to any of the services.
Service	Task	A specific service is handled by a task released by a call to the service.
Task	Memory Partition	The memory partition contains the task.
Shared Resource	Memory Partition	The memory partition contains the shared resource.
Memory Partition	Processor	The tasks of the memory partition are allocated on the processor.

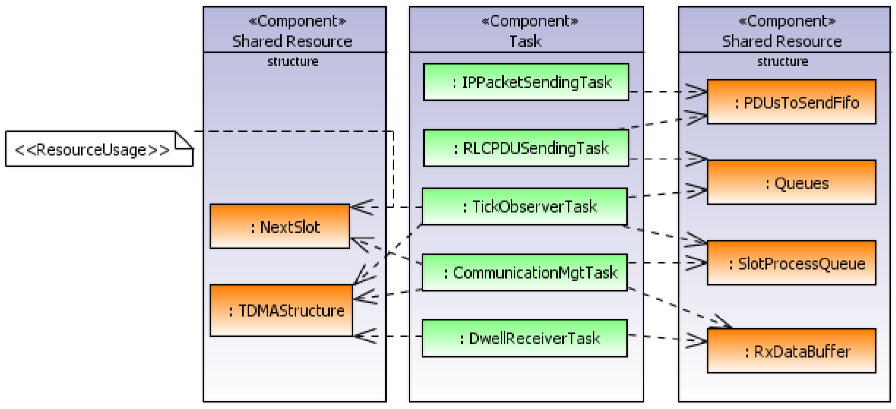


Figure 4.8: Shared Resource Usage View: Dashed arrows are resource usages; For readability, not all stereotypes are visible and stereotypes are shown by UML comments

Table 4.6: Computed WCRTs of HDRN Tasks

Task	**WCRT (µs)**
IPPacketSendingTask	1322
RLCPDUSendingTask	3047
CommMgtTask	1115
TickObserverTask	4737
DwellReceiverTask	2239

4.2.4 *Transformation to Cheddar*

The proposed architecture model contains enough information for scheduling analysis. To proceed with the analysis, the Cheddar real-time scheduling analysis tool is used. The periodic task model and its analysis methods are implemented in the tool. Thus the model in MARTE is transformed into a Cheddar-ADL model.

The transformation implementation is an update of the first MARTE to Cheddar transformation [88], developed by Thales Research & Technology, for IBM's RSA modeler. This update was necessary to respect the current Cheddar meta-model and the MARTE model proposed in this thesis.

The MARTE to Cheddar transformation is implemented as an Eclipse plug-in. It supports models conforming to the Eclipse implementation of UML. Such models can be created with the Papyrus modeler which supports MARTE.

4.3 DISCUSSION ON THE ANALYSIS RESULTS

HDRN was modeled in MARTE by using the task parameters in Table 4.1. After HDRN was modeled and transformed into a Cheddar-ADL model, the system is was analyzed with the schedulability test in [64] for the periodic task model.

Each of the following sections discusses an analysis result: a software design mistake, a missed deadline, an unbounded priority inversion, pessimistic WCRTs, and some characteristics of a general software radio protocol that couldn't be considered by the periodic task model.

4.3.1 *Software Design Mistake*

The WCRTs of tasks, computed by the test in [64], are shown in Table 4.6. To verify these values, Figure 4.9 shows distribution of the measured response times of tasks.

Let us now focus on task *TickObserverTask* and *CommunicationManagementTask* to illustrate a software design mistake. Comparison of the results in Table 4.6 and Figure 4.9 shows that the measured response time of *TickObserverTask* exceeds its computed WCRT.

This inconsistency is due to a software design mistake. Indeed, the implementation of HDRN is conform to its specification. The modeling, with the periodic task model, is also conform to the specification. The computed WCRTs should then not be less than those that are measured. Otherwise there is an inconsistent behavior during execution, which is not predicted by the specification. There is then a software design mistake.

The inconsistent behavior is a task still locking a shared resource when it is inactive. The following paragraphs describe the software design mistake, which will then be illustrated with a figure.

The *CommunicationManagementTask* is inactive when it waits for a message from the communication server (*PC-3*). During this time the other tasks with lower priority can run. But the *CommunicationManagementTask* still locks the *SlotProcessQueue* shared resource when it is inactive. This blocks in turn the *TickObserverTask* when it asks for access the resource.

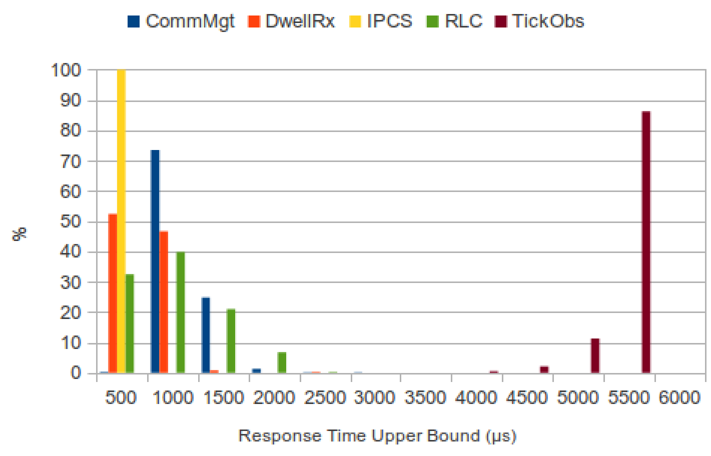


Figure 4.9: Measured Response Times of HDRN Tasks: Each category represents an interval: 500 is (0; 500], 1000 is (500; 1000], etc.; Each bar represents the percentage of the corresponding task's measured response time in the category; Abbreviations are: CommMgt = CommunicationManagementTask, DwellRx = DwellReceierTask, IPCS = IpPacketSendingTask, RLC = RLCPDUSendingTask, TickObs = TickObserverTask

Since the time, during which the *CommunicationManagementTask* is inactive (equal more or less to the *CommunicationManagementTask*'s 5000 µs period), is much longer than the execution time of tasks, the *TickObserverTask* takes in average 5000 µs to complete.

As *TickObserverTask* cannot complete during the time the *CommunicationManagementTask* is inactive, this latter task is blocked when it wants to access the *NextSlot* shared resource in its next job. *TickObserverTask* thus impacts the response time of *CommunicationManagementTask*. This behavior is not desired, nor predicted in the software specification documents.

To verify the described design mistake, the original model of HDRN is modified. A task should not lock a shared resource when it is inactive as this behavior does not respect the constraints of a test like the one in [64]. Therefore some modifications to the model are necessary, to represent this behavior. The software design mistake, as well as the necessary modifications, are shown in Figure 4.10.

The modification consists in first setting the lowest priority to the *TickObserverTask*. This is not far from reality since:

– *TickObserverTask* already has the lowest priority, along with *DwellReceiverTask* and *RLCPDUSendingTask*.
– *DwellReceiverTask* should contribute to the response time of *TickObserverTask* in the analysis. Indeed, *TickObserverTask* is more crucial for the functionality of the system.
– For the same reason *RLCPDUSendingTask* should contribute to the response time of *TickObserverTask* in the analysis.

The second step consists in adding a new task, called *SleepTask*, that is of higher priority than *TickObserverTask* but lower than the other tasks.

The *SleepTask* preempts the *TickObserverTask* when it wants to access the *SlotProcessQueue*. The preemption lasts until the *CommunicationManagementTask* is released again. This preemption represents the behavior where *SlotProcessQueue* is still locked during the time the *CommunicationManagementTask* is inactive.

The execution time of *SleepTask* is thus computed with the following equation:

$$C_{SleepTask} = T_{CommMgtTask} - r_{SleepTask} \tag{86}$$

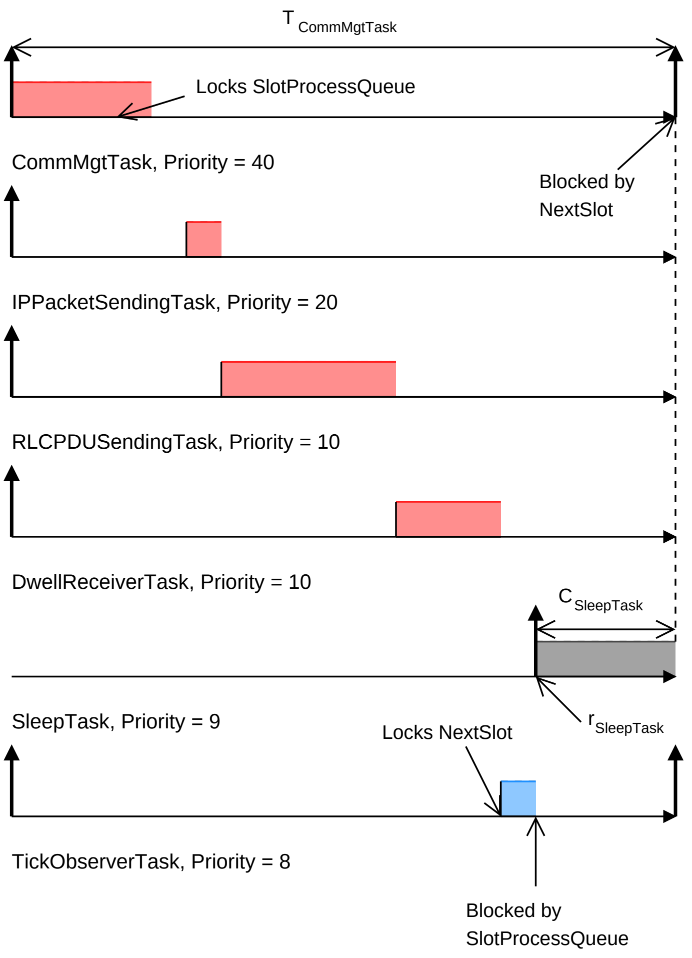

T CommMgtTask
Locks SlotProcessQueue
CommMgtTask, Priority = 40
Blocked by NextSlot
IPPacketSendingTask, Priority = 20
RLCPDUSendingTask, Priority = 10
DwellReceiverTask, Priority = 10
C SleepTask
SleepTask, Priority = 9
Locks NextSlot
r SleepTask
TickObserverTask, Priority = 8
Blocked by SlotProcessQueue

Figure 4.10: Software Design Mistake: Up arrows are task releases; Boxes are task executions; SleepTask only used for analysis; Sizes in figure are not proportional to time values

Table 4.7: Simulated WCRTs of HDRN Tasks

Task	WCRT (µs)
IPPacketSendingTask	1322
RLCPDUSendingTask	3964
CommMgtTask	3736
TickObserverTask	7619
DwellReceiverTask	2131

where C_i is the execution time of task i, T_i is the period of task i, and r_i the first release time of task i.

The first release time of *SleepTask*, $r_{SleepTask}$, is computed with the folowing equation:

$$r_{SleepTask} = \left(\sum_{j \in hp_{SleepTask}} C_j \right) + S(TickObserverTask, SlotProcessQueue) \tag{87}$$

where hp_i is the set of tasks with higher priority than task i, and $S(i, R)$ is the time task i starts using shared resource R in its execution time.

With the data in table 4.1 and 4.2, we have $C_{SleepTask} = 952$ µs and $r_{SleepTask} = 4048$ µs.

With this new model, it is not possible to apply the schedulability test in [64] because of the different release times. Thus a scheduling simulation is done in a time interval of 2 slots, i.e. 10000 µs. The simulation is done with the Cheddar simulator engine.

Note that the modification only works for computing response times in a time interval of 2 slots. *TickObserverTask* will start later in its second release, having been delayed by *CommunicationManagementTask*, but *SleepTask*'s release time and capacity cannot be modified. Table 4.7 shows the WCRTs given by the scheduling simulation.

By comparing table 4.7 with the measured response times distribution in Figure 4.9, we see that the results given by simulation, with the modified model, are consistent. This validates the assumption on the software design mistake.

In conclusion the periodic task model can be applied to the specification of HDRN but the implementation of HDRN has an inconsistent behavior not predicted by the specification. This inconsistent behavior is due to a software design mistake, which was only detected after the periodic task model was applied, and inconsistencies in analysis results were observed.

4.3.2 *Missed Deadline*

In the scheduling simulation in Cheddar, there is at least one case where the *TickObserverTask* misses its 5000 µs deadline. This has also been observed in the application's execution, where the task's maximum observed response time is 5387 µs, the average being 5058 µs.

In reality the system still works because as long as the *CommunicationManagementTask*'s response time does not exceed 5000 µs, the system can operate. Furthermore not all tasks actually run at their WCET, i.e. lateness in certain tasks may be compensated by tasks running less than their WCET in the next slot.

4.3.3 *Unbounded Priority Inversion*

Another interesting result comes from the Cheddar priority inversion detection tool. Cheddar detects that *CommunicationManagementTask* has an unbounded priority inversion from 5329 to 7950.

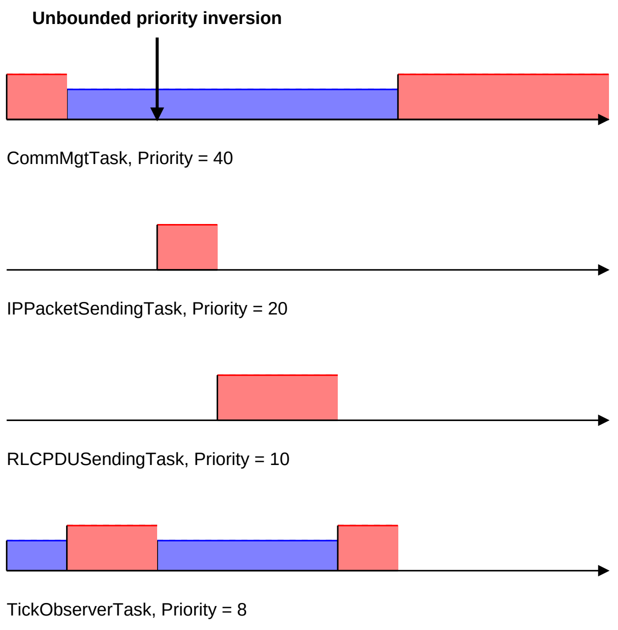

Unbounded priority inversion
CommMgtTask, Priority = 40
IPPacketSendingTask, Priority = 20
RLCPDUSendingTask, Priority = 10
TickObserverTask, Priority = 8

Figure 4.11: Unbounded Priority Inversion: A high rectangle means the task is executing; A low rectangle that it is preempted

This issue is illustrated in Figure 4.11. In the figure, *IPPacketSendingTask* and *RLCPDUSendingTask* execute when *CommunicationManagementTask* is blocked, but they do not share a resource. The execution traces show that this unbounded priority inversion does happen.

To verify the unbounded priority inversion, its time interval, from 5329 to 7950, is discussed. 5329 is the time at which *CommunicationManagementTask* asks for access to *NextSlot*. The priority inversion lasts $7950 - 5329 = 2621$ which corresponds to:

$$\begin{aligned} & C_{IPPacketSendingTask} \\ & + C_{RLCPDUSendingTask} \\ & + C_{TickObserverTask} \\ & - S(TickObserverTask, SlotProcessQueue) \\ & = 2621 = 7950 - 5329 \end{aligned} \tag{88}$$

As a reminder, in the modified task model *TickObserverTask* has the lowest priority, this is why it is preempted by *RLCPDUSendingTask* too.

The unbounded priority inversion is not surprising as the shared resources are not protected by some specific access protocol (e.g. PCP, PIP). The described blocking should not occur according to specification documents, but it does because there is a software design mistake. The choice of the execution platform thus needs to consider such shared resource access protocols.

4.3.4 *Pessimistic WCRTs*

Table 4.7 and Figure 4.9 show response times for the system with the software design mistake. Again let us focus on the two tasks impacted by the software design mistake.

In Table 4.7, the WCRT of *CommunicationManagementTask* (resp. *TickObserverTask*) is 3736 (resp. 7619). In Figure 4.9, the observed response time of *CommunicationManagementTask* (resp. *TickObserverTask*) is in majority less than 1500 (resp. less than 5500).

The computed WCRT of *CommunicationManagementTask* (resp. *TickObserverTask*) is 2.49 (resp. 1.38) times higher than the upper-bound of the response time observed for the majority of the *Communica-*

tionManagementTask (resp. *TickObserverTask*) executions. Thus the WCRT computed by the analysis is greater than in reality. This is mostly due to the fact that tasks do not execute at their WCET most of the time.

Let us now assume that the software design mistake is fixed. The test in [64] can then be used. This test gives pessimistic results.

Indeed in a slot in *Rx* mode, tasks *DwellReceiverTask*, *TickObserverTask*, and *CommunicationManagementTask* are released. *CommunicationManagementTask* executes during its WCET only when it is released in a *Tx* slot.

In a *Tx* slot *DwellReceiverTask* is not released so the WCET of *CommunicationManagementTask* should not contribute to the WCRT of *DwellReceiverTask*. Only the execution time of *CommunicationManagementTask* in a *Rx* slot (less than its WCET) should contribute to the WCRT of *DwellReceiverTask*. Again there is pessimism because tasks do not execute always at their WCET.

Therefore to get less pessimistic WCRTs, the individual job parameters of tasks must be considered, in the case of a software radio protocol.

4.3.5 *Limits of the Periodic Task Model*

The HDRN case-study is a simplified software radio protocol. Thus it does not represent a general software radio protocol. Therefore some characteristics of a real software radio protocol at Thales, were not considered for the experiment:

– TDMA release: This characteristic is not considered because tasks of HDRN are released in periodic intervals. Indeed, slots have the same duration.
– Arbitrary deadline: This characteristic is not considered because tasks of HDRN have a deadline shorter or equal to their period.
– Precedence Dependency: To model a precedence dependency of synchronous tasks, the priority assignment proposed in [5] is used for tasks of HDRN with precedence dependency. This means priorities cannot be arbitrary.
– Shared Resource: A blocking time can be bounded if a specific access protocol is used [127]. The blocking time is omitted since HDRN is supposed to be developed such that shared resource blocking should never occur unless deadlines are missed. As shown by the experiment, there are blocking times due to the software design mistake.
– Individual job parameter: This characteristic is not considered since the WCETs are considered by the periodic task model analysis.
– Partitioned multiprocessor: This characteristic is not considered because HDRN does not have security issues and thus execute on a uniprocessor.

The majority of the characteristics of a software radio protocol are thus not considered by the periodic task model. This confirms that it is not adapted for the system to analyze.

4.3.6 *Summary of the Analysis Results*

To conclude the evaluation of the scheduling analysis results of HDRN, the experiment showed that scheduling analysis, whether through schedulability tests or scheduling simulation, can help detect design mistakes in a software radio protocol. Indeed, by applying scheduling analysis, a software design mistake, missed deadline, and unbounded priority inversion were detected.

On the other hand, the schedulability results are pessimistic. The WCRTs are overestimated since the periodic task model is not adapted to a software radio protocol. This pessimism is mostly due to the variable task parameters from one job to another.

Finally the periodic task model could be applied to HDRN because it is a simplified software radio protocol. It does not have all characteristics of a general software radio protocol developed at Thales. Therefore a new task model needs to be proposed for the general case.

4.4 CONCLUSION

In this chapter an experiment was done by applying the fundamental periodic task model on a simplified software radio protocol. The case-study is the HDRN developed at Thales. The approach was to first model HDRN in UML MARTE. Then the MARTE model is transformed to the periodic task model implemented in Cheddar. This approach respects the development cycle of a software radio protocol at Thales, since the proposed MARTE model can be transformed to a MyCCM model for code generation.

The evaluation of the scheduling analysis results, given by the schedulability test in [64], showed that design mistakes of the architecture can be detected.

On the other hand, pessimism in WCRT computation limits the usefulness of the periodic task model for a system like a software radio protocol that is based on TDMA. Furthermore it was shown that the periodic task model is not applicable to a general software radio protocol since the HDRN case-study is a simplified system that does not represent the general case.

Finally the experiment showed that the individual job parameters of tasks have the greatest impact on the pessimism of computed WCRTs. Based on this observation, in the next chapter a new task model is proposed.

PUBLICATIONS The experiment described in this chapter is published in [82]. The high level software architecture model in UML MARTE, described in this chapter, is published in [78, 113, 27]. The execution platform model in UML MARTE is published in [77, 113, 26].

Chapter 5

A TASK MODEL FOR SOFTWARE RADIO PROTOCOLS

The experiment in the last chapter showed that individual job parameters of tasks have an impact on the pessimism of WCRTs computed to assess schedulability of a software radio protocol. In this chapter, the GMF task model is extended to propose a task model called Dependent General Multiframe (DGMF).

The following sections first define the DGMF task model. Then its analysis method is proposed. A summary of the analysis method is presented at the end. Afterwards the implementation in Cheddar is shown. Finally some experimental results and evaluation are exposed.

5.1 DEPENDENT GENERAL MULTIFRAME

Jobs with individual parameters is frequent in the multimedia domain. For example a video decoder task of a multimedia system has a sequence of images to decode. The parameters of a job of the task are constrained by the type of the image to decode.

The described task behavior also occurs in software radio protocols. Indeed a job of a task, released by TDMA slots, may also have parameters constrained by the slot. The job parameters are thus constrained by the sequence of slots of the TDMA frame.

A task model motivated by the described behavior is the GMF task model. As a reminder, a GMF task is an ordered vector of frames representing its jobs. Each frame can have a different execution time, deadline, and minimum separation time from the frame's release to the next frame's release.

Among task models that propose to model individual job parameters, GMF is sufficient for the modeling of a sequence of task jobs, constrained by a sequence of slots. On the other hand, as shown in Table 3.2, GMF is limited to uniprocessor systems, without task dependencies, and with constrained deadlines. The GMF task model is thus extended to propose a task model called DGMF.

The DGMF task model extends the GMF model with task dependencies. It is also applicable to partitioned multiprocessor systems. The following sections first define the DGMF task model and its properties. Then an example is shown. Finally the applicability of GMF analysis method on DGMF is discussed.

5.1.1 *DGMF Definitions and Properties*

A DGMF task G_i is a vector composed of N_i frames F_i^j, with $1 \leqslant j \leqslant N_i$. Each frame is a job of the same task G_i. Frames have some parameters inherited from GMF:

– E_i^j is the WCET of F_i^j.
– D_i^j is the relative deadline of F_i^j.
– P_i^j is min-separation of F_i^j, defined as the minimum time separating the release of F_i^j and the release of F_i^{j+1}.

A DGMF task also has a GMF period [14] inherited from the original task model. For DGMF task G_i with N_i frames, the GMF period P_i of G_i is:

$$P_i = \sum_{j=1}^{N_i} P_i^j \tag{89}$$

Frames also have some parameters and notations specific to the DGMF task model:

- $[U]_i^j$ is a set of (R, S, B) tuples denoting shared resource critical sections. F_i^j asks for access to resource R after it has run S time units of its execution time, and then locks the resource during the next S time units of its execution time.
- $[F_p^q]_i^j$ is a set of predecessor frames, i.e. frames from any other DGMF task that must complete execution before F_i^j can be released. A predecessor frame is denoted by F_p^q. Frame F_p^q can only be in $[F_p^q]_i^j$ if G_i and G_p have the same GMF period. When a frame F_x^y precedes F_i^j, the precedence dependency is denoted $F_x^y \rightarrow F_i^j$. We have $\forall F_p^q \in [F_p^q]_i^j, F_p^q \rightarrow F_i^j$. But F_p^q are not the only frames that precede F_i^j. For example, any frame that precedes a $F_p^q \in [F_p^q]_i^j$, also precedes F_i^j. For $j > 1$, we have $F_i^{j-1} \rightarrow F_i^j$.
- $\mathtt{proc}(F_i^j)$ is the processor on which F_i^j is allocated on. Critical section (R, S, B) can be in $[U]_i^j$ and $[U]_x^y$, with $F_i^j \neq F_x^y$, only if $\mathtt{proc}(F_i^j) = \mathtt{proc}(F_x^y)$.
- $\mathtt{prio}(F_i^j)$ is the fixed priority of F_i^j (for FP scheduling).
- r_i^j is the first release time of F_i^j. For the first frame F_i^1, parameter r_i^1 is arbitrary. For the next frames, we have $r_i^j = r_i^1 + \sum_{h=1}^{j-1} P_i^h$.

Frames are released cyclically [14]. Furthermore, the first frame to be released by a DGMF task G_i is always the first frame in its vector, denoted by F_i^1. Parameter r_i denotes the release time of G_i and we have $r_i = r_i^1$.

A DGMF tasks set may have the *Unique Predecessor* property:

Property 4 (Unique Predecessor). *Let F_i^j be a frame of a task G_i, in a DGMF tasks set. Let $[F_p^q]_i^j$ be the set of predecessor frames of F_i^j. F_i^{j-1} is the previous frame of F_i^j in the vector of G_i, with $j > 1$. The set of predecessor frames of F_i^j is the set $[F_p^q]_i^j$ and F_i^{j-1} (with $j > 1$).*

A DGMF tasks set is said to respect the Unique Predecessor property if, for all frames F_i^j, there is at most one frame F_x^y, in a reduced set of predecessors of F_i^j, with a global deadline (i.e. $r_x^y + D_x^y$) greater than or equal to the release time of F_i^j. The reduced set of predecessors, of F_i^j, are predecessors that do not precede another predecessor of F_i^j.

To formally define the Unique Predecessor property, the set of predecessor frames of F_i^j is denoted by $\mathtt{pred}(F_i^j)$ such that:

$$\mathtt{pred}(F_i^j) = \begin{cases} [F_p^q]_i^j \cup \{F_i^{j-1}\} & \textit{with } j > 1 \\ [F_p^q]_i^j & \textit{otherwise.} \end{cases} \tag{90}$$

Formally the Unique Predecessor property is then defined as follows:

$$\exists_{\leqslant 1} F_x^y \in \mathtt{pred}(F_i^j)', r_x^y + d_x^y \geqslant \max(\max_{F_l^h \in \mathtt{pred}(F_i^j)} (r_l^h + E_l^h), r_i^j) \tag{91}$$

where $\exists_{\leqslant 1}$ means "there exists at most one", and the set $\mathtt{pred}(F_i^j)'$ is defined as:

$$\mathtt{pred}(F_i^j)' = \mathtt{pred}(F_i^j) \setminus \{F_x^y \in \mathtt{pred}(F_i^j) \mid \exists F_k^l \in \mathtt{pred}(F_i^j), F_x^y \rightarrow F_k^l\} \tag{92}$$

A DGMF tasks set may also have the *Cycle Separation* property:

Table 5.1: DGMF Task Set

	E_i^j	D_i^j	P_i^j	$[U]_i^j$	$[F_p^q]_i^j$	$prio(F_i^j)$	$proc(F_i^j)$
$G_1; r_1 = 0$							
F_1^1	1	4	1		F_2^1	1	cpu1
F_1^2	1	3	1			1	cpu2
F_1^3	1	2	6			1	cpu1
F_1^4	1	4	4		F_2^2	1	cpu1
F_1^5	4	8	8	(R, 1, 3)	F_2^3	1	cpu1
$G_2; r_2 = 0$							
F_2^1	1	4	8			2	cpu1
F_2^2	1	4	4			2	cpu1
F_2^3	1	4	4			2	cpu1
F_2^4	2	4	4	(R, 0, 1)		2	cpu1
$G_3; r_3 = 4$							
F_3^1	1	2	2		F_4^1	1	cpu1
F_3^2	1	2	18		F_4^2	1	cpu1
$G_4; r_4 = 4$							
F_4^1	1	2	2			2	cpu1
F_4^2	1	2	18			2	cpu1

Property 5 (Cycle Separation). *A DGMF task G_i is said to respect the Cycle Separation property if:*

$$D_i^{N_i} <= r_i + P_i \tag{93}$$

The *Unique Predecessor* and *Cycle Separation* properties simplify the analysis method of DGMF so both properties are assumed for a DGMF tasks set. Later in this chapter, experiments will show that they have no impact on the ability to model a real software radio protocol, developed at Thales, with DGMF.

To illustrate the task model defined in this section, the next section shows an example with some DGMF tasks, the parameters of their frames, and a possible schedule of the tasks set.

5.1.2 *DGMF Example*

Consider the DGMF tasks set in Table 5.1, modeling tasks constrained by a TDMA frame. Frames of G_1 and G_3 have a priority of 1. Frames of G_2 and G_4 have a priority of 2. All frames are allocated on *cpu1* except F_1^2, which is allocated on *cpu2*. Frames F_1^5 and F_2^4 use a shared resource R.

Figure 5.1 shows an example of a schedule produced by the tasks set, over 20 time units. In the figure, the TDMA frame has 1 *S* slot, 2 *B* slots, and 3 *T* slots. Task G_2 is released at *S* and *T* slots. G_2 releases G_1 upon completion. G_4 is released at *B* slots. G_4 releases G_3 upon completion. Release time parameter r_i allows us to specify at which slot a task is released for the first time. For example task G_4 is released at time 4, which is the start time of the first *B* slot. Notice that precedence dependencies are respected and F_4^2 is blocked by F_1^5 during 1 time unit, due to a shared resource.

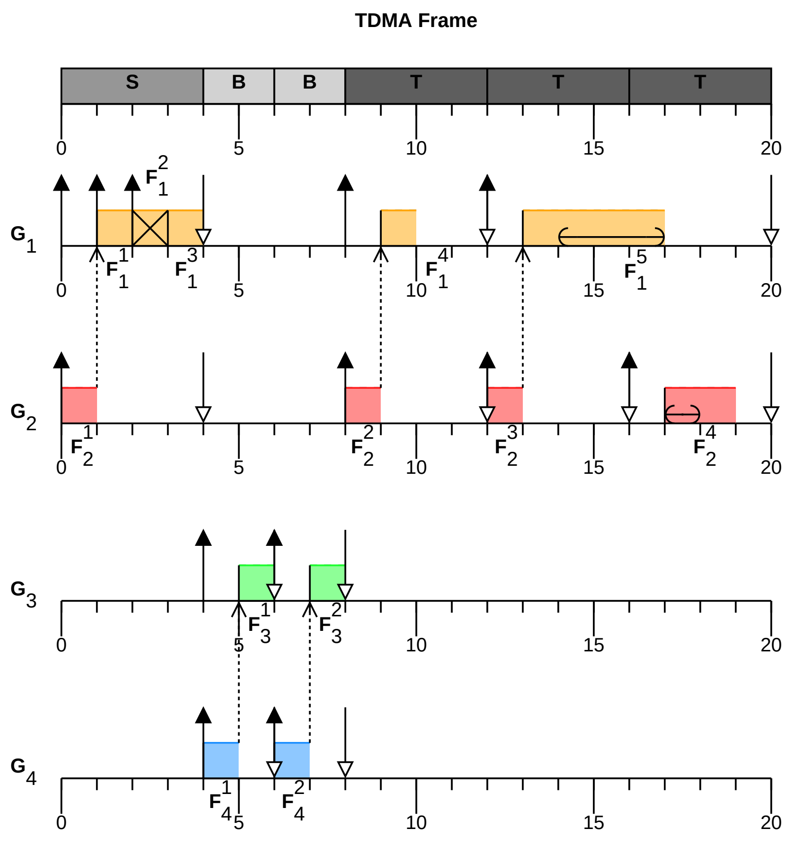


Figure 5.1: Example of Schedule of DGMF Tasks: Up arrows are frame releases; Down arrows are frame relative deadlines; Dashed arrows are precedence dependencies; Curved arrows are shared resource critical sections where R is used; Crossed frame executes on a different processor

Table 5.2: DGMF Example for GMF Test

	E_i^j	D_i^j	P_i^j	$[U]_i^j$	$[F_p^q]_i^j$	$prio(F_i^j)$	$proc(F_i^j)$
$G_1; r_1 = 0$							
F_1^1	3	5	5		F_2^1	2	cpu1
F_1^2	4	4	4			2	cpu2
$G_2; r_2 = 0$							
F_2^1	2	5	9			1	cpu1

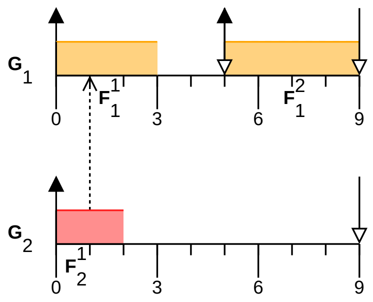


Figure 5.2: Illustration of DGMF Example for GMF Test: Up arrows are frame releases; Down arrows are frame relative deadlines; Dashed arrows are precedence dependencies

5.1.3 *Applicability of GMF Analysis Methods on DGMF*

Feasibility and schedulability tests exist for GMF tasks. In this section two tests are reminded and their applicability to DGMF is discussed.

In [14] a demand-bound feasibility test for GMF tasks is proposed. The test is restricted to independent tasks running on a uniprocessor system under a preemptive EDF scheduling policy.

In [138] a response time based schedulability test for GMF tasks is proposed. This test assumes that tasks are independent and run on a uniprocessor system with a preemptive FP scheduling policy. Furthermore the authors also assume that Property 3 (*Frame Separation*) holds: the relative deadline of a frame is less than or equal to its min-separation.

Obviously these two tests cannot be applied to DGMF tasks for the following reasons:

- DGMF tasks have task dependencies
- DGMF frames can be allocated on different processors.

The following paragraphs present an example to illustrate the task dependency issue. The test in [138] is applied to DGMF tasks with precedence dependencies.

Consider two DGMF tasks, G_1 and G_2. The parameters of these tasks are shown in Table 5.2 and illustrated in Figure 5.2. This kind of tasks set, where a predecessor task has a lesser priority than the successor task, can be found in input-output driver software.

If the test in [138] is applied, then the WCRT of G_1 is 4, and the WCRT of G_2 is 6. Both of these values are incorrect.

The WCRT of G_1 is underestimated: it should be $E_2^1 + E_1^1 = 2 + 3 = 5$ since F_2^1 precedes F_1^1. The WCRT of G_2 is overestimated: it should be $E_2^1 = 2$ since F_2^1 precedes F_1^1, which itself precedes F_1^2 because F_1^2 is released after F_1^1. Thus F_1^2 can never preempt F_1^1, unless the former misses its deadline.

Through this example, we see that a simple precedence dependency between two frames results in both under and overestimated WCRTs. Generally speaking, in the original tests for GMF, frames can be released simultaneously as long as they belong to different GMF tasks. This no longer holds true when precedence dependencies exist between frames.

The next section proposes a scheduling analysis method for DGMF by exploiting the transaction model.

5.2 DGMF SCHEDULING ANALYSIS USING TRANSACTIONS

In the previous section it was shown that GMF analysis methods cannot be applied to DGMF tasks because of task dependencies and the partitioned multiprocessor nature of the system. Two choices are then available to solve this issue:

– Extend GMF schedulability tests
– Transform to another model where schedulability tests exist

The second approach is chosen: DGMF tasks scheduling analysis will be performed by transforming them to transactions.The transaction model is chosen for the following reasons:

– Task dependencies (precedence and shared resource) are supported.
– Partitioned multiprocessor systems are considered.
– A independent GMF to transaction transformation is proposed in [116] for uniprocessor systems.

The transformation faces the issue of the difference in semantic between the two models. Indeed, transactions represent tasks related by collectively performed functionalities and timing attributes [143], not individual jobs of a task. The multiframe task models, on the other hand, express the individual job parameters.

Furthermore a transaction is a group of tasks with an offset, and release jitter. The basic entity is a task with some parameters. In the multiframe task models, the basic entity is a frame, a job of a task with some parameters.

In the next sections, the transaction model is first reminded. Then the transformation algorithm in [116] is extended for DGMF tasks. Afterwards the DGMF tasks set in Section 5.1.2 is transformed to a transactions set. Finally schedulability tests for transactions are discussed to choose one suitable for transactions resulting from DGMF transformation.

5.2.1 *Transaction Reminder and New Definitions*

As a reminder, a transaction is denoted by Γ_i and its tasks are denoted by τ_{ij}. The periodic event that releases a transaction occurs every T_i. The release time of the first job of Γ_i is denoted by r_i. Each task has parameters:

– C_{ij} is the WCET of τ_{ij}.
– O_{ij} is the offset of τ_{ij}. Value $r_{ij} = r_i + O_{ij}$ is the absolute release time of the first job of τ_{ij}.
– d_{ij} is the relative deadline of τ_{ij}. Value $O_{ij} + d_{ij}$ is the global deadline of τ_{ij}. Value $r_i + O_{ij} + d_{ij}$ is the absolute deadline of the first job of τ_{ij}.
– J_{ij} is the maximum jitter of τ_{ij}.
– B_{ij} is the WCBT of τ_{ij}.
– $\mathtt{prio}(\tau_{ij})$ is the fixed priority of τ_{ij}.
– $\mathtt{proc}(\tau_{ij})$ is the processor on which τ_{ij} is allocated on.
– R^w_{ij} is the global WCRT of τ_{ij}.
– R^b_{ij} is the global BCRT of τ_{ij}.

As a reminder, a critical section is denoted by (τ_{ij}, R, S, B). A precedence dependency between two tasks is denoted by $\tau_{ip} \prec \tau_{ij}$. Later sections will focus on tree-shaped transactions. In a tree-shaped transaction a task τ_{ij} has at most one immediate predecessor (denoted $\mathtt{pred}(\tau_{ij})$), but may have several immediate successors (denoted $\mathtt{succ}(\tau_{ij})$).

The next section exposes the DGMF to transaction transformation algorithm.

5.2.2 *DGMF To Transaction*

The DGMF to transaction transformation aims at expressing parameters and dependencies in the DGMF task model as ones in the transaction model. The transformation has three major steps:

– **Step** 1: Transform independent DGMF to transaction, i.e. consider DGMF tasks independent and transform them to transactions.
– **Step** 2: Express shared resource critical sections, i.e. model critical sections in the resulting transaction set and compute WCBTs.
– **Step** 3: Express precedence dependencies, i.e. model precedence dependencies in the transaction model.

In the following sections, each step is explained in detail.

5.2.2.1 *Independent DGMF to Transaction*

Step 1 consists in transforming each DGMF task to a transaction by considering DGMF tasks as independent. Algorithm 5.1 shows the original algorithm proposed in [116] that is extended for DGMF tasks.

The idea behind the algorithm is to transform frames F_i^j of a DGMF task G_i into tasks τ_{ij} of a transaction Γ_i. Parameters in the transaction model, like WCET (C_{ij}), relative deadline (d_{ij}) and priority ($\mathrm{prio}(\tau_{ij})$), are computed from parameters E_i^j, D_i^j and $\mathrm{prio}(F_i^j)$ from the DGMF model.

To transform the min-separation of a frame in the DGMF model, offsets (O_{ij}) are used in the transaction model. The offset of a task τ_{ij} is computed by summing the P_i^h of frames F_i^h preceding F_i^j in the vector of G_i.

In the extension of the transformation, the release time r_i of a DGMF task G_i is transformed into the release time r_i of a transaction Γ_i.

Algorithm 5.1 Independent DGMF to Transaction

```
1:  for each DGMF task $G_i$ do
2:      Create transaction $\Gamma_i$
3:
4:      $T_i \leftarrow \sum_{j=1}^{N_i} P_i^j$
5:      $\Gamma_i.r_i \leftarrow G_i.r_i$
6:
7:      for each $F_i^j$ in $G_i$ do
8:          Create task $\tau_{ij}$ in $\Gamma_i$
9:
10:         $C_{ij} \leftarrow E_i^j$
11:         $d_{ij} \leftarrow D_i^j$
12:         $J_{ij} \leftarrow 0$
13:         $B_{ij} \leftarrow 0$
14:         $\mathrm{prio}(\tau_{ij}) \leftarrow \mathrm{prio}(F_i^j)$
15:         $\mathrm{proc}(\tau_{ij}) \leftarrow \mathrm{proc}(F_i^j)$
16:         if $j = 1$ then
17:             $O_{ij} \leftarrow 0$
18:         else
19:             $O_{ij} \leftarrow \sum_{h=1}^{j-1} P_i^h$
20:         end if
21:     end for
22: end for
```

5.2.2.2 *Express Shared Resource Critical Sections*

In **Step** 2, the goal is to express critical sections of tasks τ_{ij} and then compute their B_{ij}. **Step** 2 is thus divided into 2 sub-steps:

- **Step** 2.**A**: Express critical sections
- **Step** 2.**B**: Compute worst case blocking times

The following paragraphs focus on these two sub-steps.

2.A, EXPRESS CRITICAL SECTIONS In **Step** 2.**A**, if a critical section is defined for a frame, then the task, corresponding to the frame after transformation, must also have the critical section. If there is a critical section (R, S, B) in $[U]_i^j$, then a critical section (τ_{ij}, R, S, B) must be specified in the transaction set resulting from **Step** 1.

2.B, COMPUTE WORST CASE BLOCKING TIMES In **Step** 2.**B**, the goal is to compute B_{ij} of a task τ_{ij}, assuming B_{ij} of a task is bounded [127, 117]. B_{ij} is computed by considering all shared resource R accessed by τ_{ij}. Then all other tasks τ_{xy} that may access R are considered too. Since some other tasks τ_{xy} actually represent jobs of a same DGMF task, not all tasks τ_{xy} accessing R must be considered for the computation of B_{ij}. Otherwise the WCBT is pessimistic.

Consider a task τ_{xy} that share a resource R with task τ_{ij}. Let $(\tau_{xy}, R, S, B_{max})$ denote the longest critical section of τ_{xy}. Let $SG(\tau_{xy}, \tau_{ij})$ denote the function that returns true if τ_{xy} and τ_{ij} result from frames that are part of a same DGMF task. Let $\beta_{ij,R}$ be the set of tasks considered for the computation of B_{ij} for a given R. For a given R, set $\beta_{ij,R}$ contains a task τ_{xy} if:

$$proc(\tau_{xy}) = proc(\tau_{ij}) \wedge \tag{94}$$
$$\neg SG(\tau_{xy}, \tau_{ij}) \wedge \tag{95}$$
$$\neg\exists(\tau_{kl}, R, S', B') \mid (SG(\tau_{kl}, \tau_{ij}) \wedge B' > B_{max}) \tag{96}$$

The three conditions have the following meaning:

1. Condition 1 (Equation 94) means that τ_{xy} must be on the same processor as τ_{ij}.
2. Condition 2 (Equation 95) means that both tasks must not come from frames that are part of a same DGMF task.
3. Finally for condition 3 (Equation 96), suppose that τ_{xy} results from a frame in a DGMF task G_x. For a given R, τ_{xy} is only considered if it has the longest critical section of R, among all tasks (with critical sections of R) that result from frames that are part of G_x.

*Proof of **Step** 2.**B**.* Frames represent jobs of a same DGMF task. Tasks resulting from frames of a same DGMF cannot block each other since jobs of a same DGMF task do cannot block each other. Furthermore tasks, resulting from frames of a same DGMF task, cannot all block another task, as if they are individual tasks, since they represent jobs of a same DGMF task. □

Equations for the computation of B_{ij} are then adapted with the new set $\beta_{ij,R}$. Equation 1, for the WCBT computation with PIP, becomes:

$$B_{ij} = \sum_{R} \max_{\tau_{xy} \in \beta_{ij,R}} (\text{Critical section duration of } \tau_{xy}) \tag{97}$$

where R denotes a shared resource.

Equation 2, for the WCBT computation with PCP, becomes:

$$B_{ij} = \max_{\tau_{xy} \in \beta_{ij,R}, R} \left(D_{xy,R} \mid prio(\tau_{xy}) < prio(\tau_{ij}), C(R) \leqslant prio(\tau_{ij}) \right) \tag{98}$$

where R denotes a shared resource, $C(R)$ the ceiling priority of R, and $D_{xy,R}$ the duration of the longest critical section of task τ_{xy} using shared resource R.

5.2.2.3 *Express Precedence Dependencies*

The goal of **Step** 3 is to model precedence dependencies in the transactions set, with respect to how they are modeled in the transaction model with offsets. **Step** 3 is divided into three sub-steps:

– **Step** 3.**A**: Express precedence dependency, i.e. precedence dependencies between frames are expressed as precedence dependencies between tasks.
– **Step** 3.**B**: Model precedence dependency in the transaction model, i.e. precedence dependencies between tasks are modeled with offsets according to [105]. Precedence dependent tasks must also be in a same transaction.
– **Step** 3.**C**: Reduce precedence dependencies, i.e. simplify the transactions set by reducing number of precedence dependencies.

The following paragraphs present each of these sub-steps.

3.A, EXPRESS PRECEDENCE DEPENDENCY Two kinds of precedence dependency are defined in the DGMF model: intra-dependency and inter-dependency. A intra-dependency is a precedence dependency that is implicitly expressed between frames of a same DGMF task. Indeed, frames of a DGMF task execute in the order defined by the vector. An inter-dependency is a precedence dependency between frames belonging to different DGMF tasks.

An intra-dependency in the DGMF set is expressed in the transaction set by a precedence dependency between tasks, representing successive frames, if they are part of a same transaction Γ_i with N_i tasks, resulting from **Step** 1: $\forall j < N_i, \tau_{ij} \prec \tau_{i(j+1)}$.

This also ensures that these tasks are part of the same precedence dependency graph, which is important for determining if the transaction is linear, tree-shaped or graph-shaped.

Inter dependencies must also be expressed in the transactions set resulting from **Step** 1. If a frame F_p^q, corresponding to task τ_p^q, is in the set of predecessor frames $[F_p^q]_i^j$ of frame F_i^j, corresponding to task τ_{ij}, then a precedence dependency $\tau_{pq} \prec \tau_{ij}$ is expressed.

*Proof of **Step** 3.A.* By definition frames of G_i are released in the order defined by the vector of G_i so F_i^j precedes F_i^{j+1} ($j < N_i$). Task τ_{ij} (resp. $\tau_{i(j+1)}$) is the result of the transformation of F_i^j (resp. F_i^{j+1}), thus by construction we must have $\tau_{ij} \prec \tau_{i(j+1)}$. The same proof is given for $\tau_{pq} \prec \tau_{ij}$, resulting from the transformation of $F_p^q \in [F_p^q]_i^j$. □

3.B, MODEL PRECEDENCE DEPENDENCY IN THE TRANSACTION MODEL In the transaction model, precedence dependencies should be modeled with offsets. This is done in **Step** 3.**B** with three algorithms: **(ALG1)** Task Release Time Modification; **(ALG2)** Transaction Merge; and **(ALG3)** Transaction Release Time Modification.

(ALG1) Task Release Time Modification In **ALG1** the release time r_{ij} of each task τ_{ij}, in the transactions set, is modified according to precedence dependencies. This enforces that the release time of τ_{ij} is later or equal to the latest completion time $(r_{pq} + C_{pq})$ of a predecessor τ_{pq} of τ_{ij}.

Task release times are changed by modifying offsets because $r_{ij} = r_i + O_{ij}$, where r_i is the release time of Γ_i. The release time modification algorithm is shown in Algorithm 5.2. Since the release time of τ_{pq} may also be modified when the algorithm runs, release time modifications are made until no more of them occur. Note that when the offset O_{ij} of τ_{ij} is increased, its relative deadline (d_{ij}) is shortened and then compared to its WCET (C_{ij}) to verify if the deadline is missed.

Proof of Algorithm 5.2. Let us assume $\tau_{pq} \prec \tau_{ij}$. The earliest release time of τ_{ij} is r_{ij}. According to [34], $\tau_{pq} \prec \tau_{ij} \Rightarrow r_{pq} + C_{pq} \leqslant r_{ij}$ is true. The implication is false if $\neg(r_{pq} + C_{pq} \leqslant r_{ij})$, otherwise said $r_{pq} + C_{pq} > r_{ij}$. Therefore if we have $\tau_{pq} \prec \tau_{ij}$ then we cannot have $r_{pq} + C_{pq} > r_{ij}$. Thus, for all $\tau_{pq} \prec \tau_{ij}$, release time r_{ij} must be modified to satisfy $r_{pq} + C_{pq} \leqslant r_{ij}$, if $\tau_{pq} \prec \tau_{ij}$ and $r_{pq} + C_{pq} > r_{ij}$. Since $r_{ij} = r_i + O_{ij}$, the offset O_{ij} is increased to increase r_{ij}. Relative deadline d_{ij} is relative to O_{ij}, thus d_{ij} must be decreased by the amount O_{ij} is increased. □

Algorithm 5.2 Task Release Time Modification

```
 1: repeat
 2:     NoChanges ← true
 3:
 4:     for each $\tau_{pq} \prec \tau_{ij}$ do
 5:         if $r_{pq} + C_{pq} > r_{ij}$ then
 6:             NoChanges ← false
 7:
 8:             diff ← $r_{pq} + C_{pq} - r_{ij}$
 9:             $O_{ij} \leftarrow O_{ij}+$ diff
10:             $d_{ij} \leftarrow d_{ij}-$ diff
11:             $r_{ij} \leftarrow r_i + O_{ij}$
12:
13:             if $d_{ij} < C_{ij}$ then
14:                 STOP (Deadline Missed)
15:             end if
16:         end if
17:     end for
18: until NoChanges
```

(ALG2) Transaction Merge Up until now, the transformation algorithm produces separate transactions even if they contain tasks that have precedence dependencies with other tasks from other transactions. This does not respect the modeling of precedence dependencies in [105]. Indeed two tasks with a precedence dependency should be in the same transaction and they should be delayed from same event that releases the transaction. Two transactions are thus merged into one single transaction if there exists a task in one that has a precedence dependency with a task in the other:

$$\exists \tau_{pq}, \tau_{ij} \mid (\Gamma_i \neq \Gamma_p) \wedge (\tau_{pq} \prec \tau_{ij} \vee \tau_{ij} \prec \tau_{pq}) \tag{99}$$

Merging two transactions consist in obtaining a final single transaction, containing the tasks of both. Algorithm 5.3 merges transactions two by two until there are no more transactions to merge.

Algorithm 5.3 Transaction Merge

```
1: for each $\tau_{pq} \prec \tau_{ij}$ do
2:     if $\Gamma_p \neq \Gamma_i$ then
3:         for each task $\tau_{ij}$ in $\Gamma_i$ do
4:             Assign $\tau_{ij}$ to $\Gamma_p$
5:         end for
6:     end if
7: end for
```

Proof of Algorithm 5.3. As a reminder, tasks of a transaction are related by precedence dependencies and a task in a transaction is released after the periodic event that releases the transaction. Let us consider two tasks τ_{ij} and τ_{pq}, with $\tau_{pq} \prec \tau_{ij}$. Task τ_{ij} (resp. τ_{pq}) is originally a frame F_i^j (resp. F_p^q). We have $F_p^q \in [F_p^q]_i^j \Rightarrow P_i = P_p$. G_i (resp. G_p) is transformed into Γ_i (resp. Γ_p) with period T_i (resp. T_p). We then have $T_i = P_i = P_p = T_p$. Thus τ_{ij} and τ_{pq} are released after periodic events of period $T_i = T_p$. Since $\tau_{pq} \prec \tau_{ij}$, τ_{ij} is released after τ_{pq}. Thus τ_{ij} is released after the periodic event after which τ_{pq} is released. Therefore τ_{ij} and τ_{pq} are released after the same periodic event, that releases transaction Γ_p. Both tasks then belong to Γ_p. □

(ALG3) Transaction Release Time Modification After merging two transactions into Γ_m, the offset O_m^j of a task τ_m^j (originally denoted by τ_o^j and belonging to Γ_o) is still relative to the release time r_o of Γ_o, no matter the precedence dependencies. In Γ_m, each offset must thus be set relatively to r_m, the release time of Γ_m. Release time r_m is computed beforehand. This is done in Algorithm 5.4.

Algorithm 5.4 Transaction Release Time Modification

```
1: for each merged transaction $\Gamma_m$ do
2:     $r_m \leftarrow +\infty$
3:     for each $\tau^j_m$ in $\Gamma_m$, originally in $\Gamma_o$ do
4:         $r_m \leftarrow \mathbf{min}(r_m, r_o + O^j_m)$
5:     end for
6:     for each $\tau^j_m$ in $\Gamma_m$, originally in $\Gamma_o$ do
7:         $O^j_m \leftarrow r_o + O^j_m - r_m$
8:     end for
9: end for
```

Algorithm 5.4 starts by finding the earliest (minimum) task release time in a merged transaction Γ_m (as a reminder, O^j_m is still relative to r_o at this moment). The earliest task release time becomes r_m. The offset O^j_m of each task τ^j_m is then be modified to be relative to r_m.

Note that when transactions are merged and all of them have at least one task released at time $t = 0$, then Algorithm 5.4 produces the same merged transaction. This is the case for the transaction resulting from the transformation of the DGMF tasks set example (Section 5.1.2), which was used to model tasks constrained by a TDMA frame.

Proof of Algorithm 5.4. Let Γ_m be a merged transaction. Tasks in Γ_m were originally in Γ_o. The event that releases Γ_m occurs at r_m, which must be the earliest release time r^j_m of a task τ^j_m in Γ_m, otherwise the definition of a transaction is contradicted. A task τ^j_m should be released at $r^j_m = r_o + O^j_m$. Once r_m is computed, when task offsets have not been modified yet, it is possible to have $r_o + O^j_m \neq r_m + O^j_m$. If we assign $r^j_m \leftarrow r_m + O^j_m$ then τ^j_m may not be released at $r_o + O^j_m$. This contradicts the fact that τ^j_m should be released at $r^j_m = r_o + O^j_m$. Therefore O^j_m must be shortened to be relative to r_m: $O^j_m \leftarrow r_o + O^j_m - r_m$. Since $r_m = \min_{\tau^j_m \in \Gamma_m} (r_o + O^j_m)$, the minimum value of $r_o + O^j_m - r_m$ is 0 and thus the assignment $O^j_m \leftarrow r_o + O^j_m - r_m$ will never assign a negative value to O^j_m. □

3.C, REDUCE PRECEDENCE DEPENDENCIES In **Step 3.C**, the transactions set can be simplified by reducing the number of precedence dependencies expressed in **Step 3.A**. This could not be done in **Step 3.A**, before **Step 3.B**, because we did not yet know the latest completion time of a task's predecessors. Now that offsets have been modified, some precedence dependencies can be reduced. Reducing precedence dependencies has the effect of reducing the number of immediate predecessors/successors of a task.

Algorithm 5.5 iterates through tasks with more than 1 predecessor. For a specific task τ_{ij}, the algorithm reduces predecessors τ_{iq} that have a global deadline (i.e. $O_{iq} + d_{iq}$) smaller than the offset O_{ij} of τ_{ij}. This is the first precedence dependency reduction.

The algorithm also reduces redundant precedence dependencies, similar to a graph reduction algorithm [68]. A precedence dependency is said redundant if it is already expressed by some other precedence dependency. Redundancy is due to the transitivity of precedence dependency. For example, if $\tau_{iq} \prec \tau_{ij}$ and $\tau'_{iq} \prec \tau_{ij}$ were expressed previously, and we have $\tau_{iq} \prec \tau'_{iq}$ due to some other expressed precedence dependencies and the transitivity of precedence dependency, then $\tau_{iq} \prec \tau_{ij}$ is reduced.

Proof of Algorithm 5.5. Let us assume $\tau_{iq} \prec \tau_{ij}$ and $O_{iq} + d_{iq} < O_{ij}$. By definition $t_0 + O_{ij}$ is the earliest release time of a job of τ_{ij}, corresponding to the job of Γ_i released at t_0. For the job of Γ_i released at t_0, the absolute deadline of a corresponding job of τ_{iq} is $t_0 + O_{iq} + d_{iq}$. The job

Algorithm 5.5 Reduce Precedence Dependencies

```
1:  for each task $\tau_{ij}$ with multiple predecessors do
2:      for each $\tau_{iq} \prec \tau_{ij}$ do
3:          if $O_{iq} + d_{iq} < O_{ij}$
4:              or $\exists \tau'_{iq} \mid \tau'_{iq} \prec \tau_{ij} \wedge \tau_{iq} \prec \tau'_{iq}$ then
5:              Remove $\tau_{iq} \prec \tau_{ij}$
6:          end if
7:          if $\tau_{ij}$ has only one predecessor then
8:              break
9:          end if
10:     end for
11: end for
```

Table 5.3: Transaction from DGMF Transformation

	C_{ij}	O_{ij}	d_{ij}	J_{ij}	B_{ij}	$prio(\tau_{ij})$	$proc(\tau_{ij})$
τ_{11}	1	1	3	0	0	1	cpu1
τ_{12}	1	2	2	0	0	1	cpu2
τ_{13}	1	3	1	0	0	1	cpu1
τ_{14}	1	9	3	0	0	1	cpu1
τ_{15}	4	13	7	0	0	1	cpu1
τ_{21}	1	1	4	0	0	2	cpu1
τ_{22}	1	8	4	0	0	2	cpu1
τ_{23}	1	12	4	0	0	2	cpu1
τ_{24}	2	16	4	0	3	2	cpu1
τ_{31}	1	5	1	0	0	1	cpu1
τ_{32}	1	7	1	0	0	1	cpu1
τ_{41}	1	4	2	0	0	2	cpu1
τ_{42}	1	6	2	0	0	2	cpu1
Tick	0	0	$+\infty$	0	0	0	cpu3
Critical Sections							
(τ_{13} R, 1, 3), (τ_{24} R, 0, 1)							

of τ_{iq} must finish before $t_0 + O_{iq} + d_{iq}$ and τ_{ij} is released at earliest after $t_0 + O_{iq} + d_{iq}$ since $O_{iq} + d_{iq} < O_{ij}$. Thus the precedence dependency constraint $\tau_{iq} \prec \tau_{ij}$ is already encoded in the relative deadline d_{iq} of τ_{iq}. Tasks in a transaction are related by precedence dependency so τ_{ij} must have at least one predecessor. □

The next section shows an example of a DGMF to transaction transformation.

5.2.3 *Transformation Example*

The DGMF tasks set in Section 5.1.2 is transformed into a transaction Γ_1 of period $T_1 = 20$. Tasks of transaction Γ_1 are defined with parameters shown in Table 5.3. PCP [127] is assumed for computation of B_{ij}.

Tasks are allocated on *cpu1* except τ_{12}, which is allocated on *cpu2*. Task *Tick* represents a ghost root task (Definition 55) added by the test in [118] for the analysis. In the specific example, *Tick* can

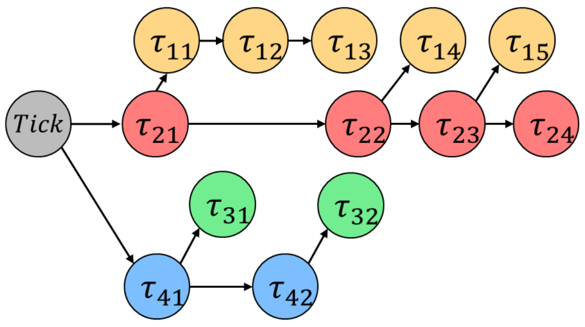


Figure 5.3: Tasks Precedence Dependency Graph

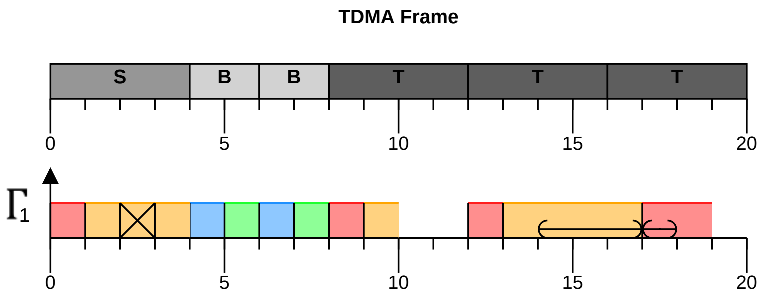


Figure 5.4: Transaction from DGMF Tasks Transformation: Curved arrows are shared resource critical sections; Tasks execute on same processor except the crossed task

represent the first TDMA tick that starts the whole TDMA frame. Therefore ghost root task ensures that all tasks, constrained by the TDMA frame, are part of the same precedence dependency graph.

For completeness, parameters of *Tick* are also given in Table 5.3. Conform to its definition, *Tick* is allocated alone on a processor (*cpu3*) modeled only for the analysis. It has an execution time of 0, and offset of 0, an infinity deadline, no jitter, a blocking time of 0, and its priority does not matter since it is allocated alone on a processor.

Figure 5.3 shows the precedence dependency graph of tasks. An example of a schedule over 20 time units is shown in Figure 5.4. Notice that the schedule of tasks in Figure 5.4 is the same as the schedule of frames in in Figure 5.1.

The next section determines the characteristics of the transaction given by the transformation example presented this section, and in the general case. A suitable schedulability test is also discussed.

5.2.4 *Assessing Schedulability of Resulting Transactions*

The task parameters in Table 5.3, the precedence dependency graph in Figure 5.3, and the schedule in Figure 5.4 show that the transaction presented in the last section has the following characteristics:

– Tree-shaped
– Tasks may be non-immediate as defined below.

Definition 58 (Non-Immediateness). *A task τ_{ix} and its immediate successor task τ_{iy} are said to be non-immediate tasks if $\tau_{ix} = \mathrm{pred}(\tau_{iy}) \wedge O_{iy} > O_{ix} + C^{b}_{ix}$. Task τ_{ix} is called a non-immediate predecessor and τ_{iy} a non-immediate successor.*

A non-immediate task is thus one that is not necessarily immediately released by its predecessor. It is released at an earliest time t if the predecessor completes before or at t, but immediately by the predecessor if the predecessor completes after t.

In the general case, tree-shaped transactions with non-immediate tasks are the results of the DGMF to transaction transformation, if the DGMF tasks set has the *Unique Predecessor* property:

Theorem 18. *A DGMF tasks set with the Unique Predecessor property (Property 4) is transformed into a transaction set without tasks that have more than one predecessor.*

Proof. Let a DGMF tasks set have the *Unique Predecessor* property. A frame F_i^j with inter and intra-dependencies is transformed into a task τ_{ij} with several immediate predecessors. Task τ_{ij} has several immediate predecessors that correspond to predecessor frames of F_i^j. Let τ_{iy} be an immediate predecessor of τ_{ij}.

Algorithm 5.5 removes the precedence dependency $\tau_{iy} \prec \tau_{ij}$, if it is redundant, i.e. if τ_{iy} does not result from a frame in the reduced set of predecessors (see Property 4) of F_i^j. From now on, let us consider that redundant precedence dependencies have been removed. Otherwise said, only frames of the reduced set of predecessors of F_i^j are considered, as well as tasks that result from them.

At most one predecessor frame F_x^y of F_i^j can have a global deadline (i.e. $r_x^y + D_x^y$) greater than the release time r_i^j of F_i^j. By construction, at most one immediate predecessor τ_{iy} of τ_{ij} (resulting from F_x^y and assigned to the same transaction as τ_{ij}) can have a global deadline greater than the offset of τ_{ij} (i.e. $O_{iy} + d_{iy} \geqslant O_{ij}$). Algorithm 5.5 removes a precedence dependency $\tau_{iy} \prec \tau_{ij}$ if $O_{ij} > O_{iy} + d_{iy}$. Since there is at most one immediate predecessor τ_{iy} of τ_{ij} that has $O_{iy} + d_{iy} \geqslant O_{ij}$, all other immediate predecessors will be reduced until task τ_{ij} has at most one immediate predecessor.

This proves that tree-shaped transactions result from transformation of DGMF tasks respecting the *Unique Predecessor* property. The transformation example in Section 5.2.3 showed that there is at least one case where there are non-immediate tasks in the transactions set resulting from the transformation. □

To assess schedulability of transactions that are the result of the transformation, a schedulability test applicable to tree-shaped transactions with non-immediate tasks must be applied.

Furthermore, without knowledge that the tasks represent frames initially, the schedulability test considers that it is possible for job k of a task τ_{i1} representing the first frame of a DGMF task G_i, to interfere with job $k-1$ of a task τ_{iN_i} representing the last frame of the same DGMF task G_i.

In reality this is not possible because job k of τ_{i1} executes after job τ_{iN_i} because frames represent jobs of a DGMF task. If the *Cycle Separation* property is respected, then job k of τ_{i1} will only interfere job $k-1$ of τ_{iN_i} if τ_{iN_i} misses its deadline. This is why the *Cycle Separation* property is assumed.

This chapter does not focus on proposing a schedulability test for tree-shaped transactions with non-immediate tasks. The test will be the subject of the next chapter.

The following section sums up the proposed scheduling analysis method for DGMF tasks.

5.3 SUMMARY OF DGMF ANALYSIS METHOD

Figure 5.5 sums up the scheduling analysis method proposed for the DGMF task model.

The analysis method consists first in transforming the DGMF tasks set to a transactions set. Then the analysis is performed on the transactions set.

The transformation first transforms independent GMF tasks with frames to transactions with tasks. Then critical sections of frames are expressed as those of tasks that are the result of the last step. The WCBTs of tasks are then computed without overestimating the blocking times since some tasks originally represent frames of a same DGMF task. Precedence dependencies between frames are then expressed as those between tasks, by respecting the semantics and constraints of the transaction model.

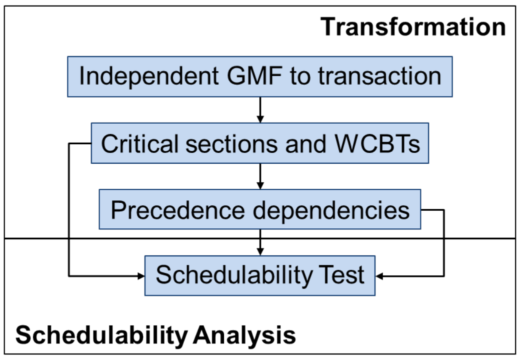


Figure 5.5: Summary of Analysis Method

After the transformation, the method proceeds to the schedulability analysis. Schedulability is assessed by computing WCRTs of tasks with a schedulability test and RTA method adapted for transactions that are the result of the transformation. This schedulability test is the subject of the next chapter.

The following section shows how the DGMF and transaction models are implemented in the Cheddar scheduling analysis tool.

5.4 IMPLEMENTATION IN CHEDDAR

The DGMF scheduling analysis method is implemented in Cheddar. Therefore the DGMF and GMF task models, transaction model, transformation of DGMF tasks to transactions, and some schedulability tests for transactions are implemented [1].

In the following sections, the Cheddar framework is first presented. Afterwards the extension of the framework is exposed. Finally advantages and drawbacks of the implementation solution are discussed.

5.4.1 *Cheddar*

Figure 5.6 illustrates the Cheddar framework. Users first specify their *architecture* in Cheddar-ADL. The GUI provided by Cheddar can be used to generate the *Cheddar-ADL model*. Model transformation [125] can also be applied to produce an architecture in Cheddar-ADL from a standard ADL (e.g. MARTE to Cheddar, AADL to Cheddar [45]). A scheduling analysis method provided by the tool (*schedulability test* or *simulator*) then computes the *analysis results* (i.e. schedulability or simulation trace).

A *Cheddar-ADL model* is conform to its *Cheddar-ADL meta-model*, which is specified in EXPRESS [128, 111], a data modeling language. Cheddar-ADL is a language that is close to seminal scheduling analysis methods [85, 64, 127]. For example, entities of tasks, processors and shared resources are defined. Figure A.1 in Appendix A shows the complete Cheddar-ADL meta-model.

Through a model-driven process [111], the *Cheddar-ADL meta-model* is used to generate code of *Cheddar-ADL classes*, a part of the *Cheddar framework*. This ensures that the code of the *Cheddar-ADL classes* is always conform to the *Cheddar-ADL meta-model*. The Cheddar framework is composed of generated code for Cheddar-ADL, and of manually written code in the *schedulability tests library* and the *schedulers library*.

1. All sources available at `http://beru.univ-brest.fr/svn/CHEDDAR/branches/shuaili/src/`; Examples of use available at `http://beru.univ-brest.fr/svn/CHEDDAR/trunk/project_examples/dgmf_sim/` and `http://beru.univ-brest.fr/svn/CHEDDAR/trunk/project_examples/wcdops+_nimp/`

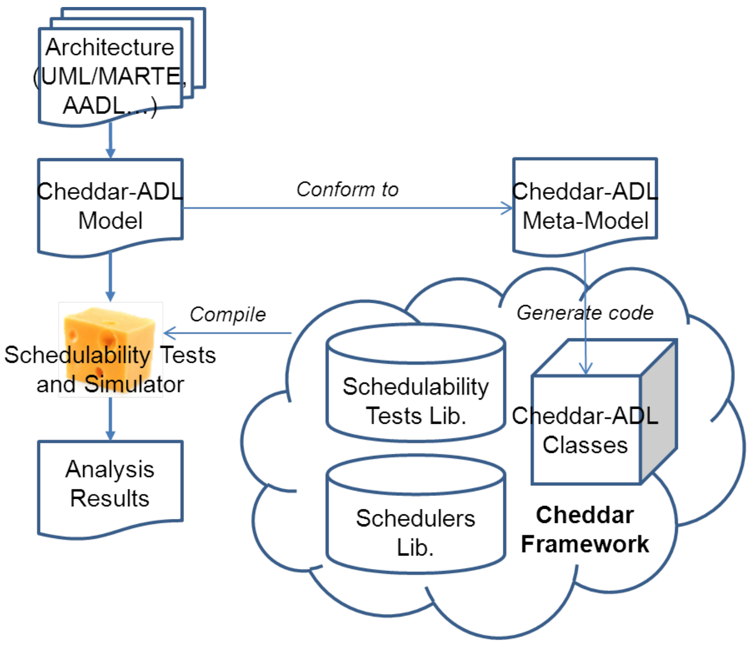


Figure 5.6: Cheddar Framework

To extend Cheddar, the general approach is to extend the *schedulability tests* or *schedulers libraries* of the framework. If necessary, the *Cheddar-ADL meta-model* is modified, and code of *Cheddar-ADL classes* is generated. This is the approach used to implement the whole DGMF analysis method, presented in the next section.

5.4.2 *Implementation of DGMF and Transaction*

To assess if Cheddar-ADL is sufficient to implement the DGMF and transaction models, let us focus on a partial meta-model of Cheddar-ADL with task entities, in Figure 5.7.

5.4.2.1 *Re-use of Existing Entities*

A number of entities that already exist in Cheddar-ADL can be re-used to implement the DGMF and transaction models.

A task entity in Cheddar is any entity that extends the *Generic_Task* entity. In Cheddar-ADL any task entity can have a *Critical_Section* where it uses a *Generic_Resource* entity, representing a shared resource. A task is allocated on a *Generic_Processor* entity. Through the *Precedence_Dependency* entity, a precedence dependency can be specified between two tasks. Thus the approach chosen to implement the DGMF and transaction models, is to re-use the task concept in Cheddar.

The *Periodic_Task* entity has attributes of a task in the transaction model so it can be re-used directly. The concept of a frame in the DGMF model does not exist in Cheddar. A new *Frame_Task* entity is thus proposed. By extending a *Generic_Task*, it is possible to express critical sections and precedence dependencies of a frame.

The main entity in the DGMF and transaction models that cannot be modeled in the current Cheddar-ADL, are the transaction and DGMF entities themselves.

5.4.2.2 *New Task Group Entities*

To model a transaction and a DGMF entity, a *task group* concept is introduced in the Cheddar-ADL meta-model. A task group is modeled by the new entity *Generic_Task_Group* in Figure 5.7.

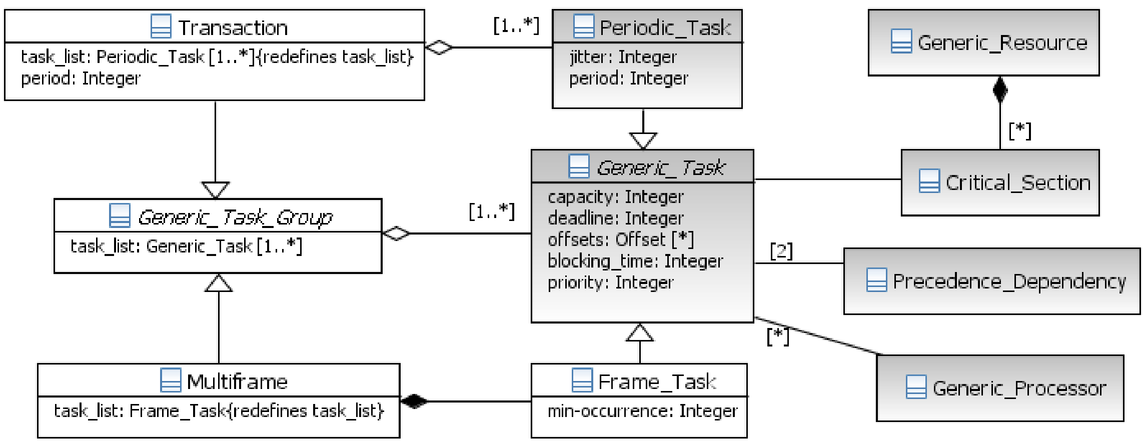


Figure 5.7: Cheddar-ADL Partial Meta-Model: White boxes highlight extensions

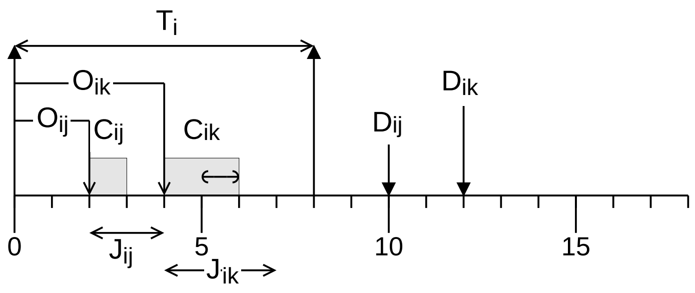


Figure 5.8: Transaction with Two Tasks

A *Generic_Task_Group* is a set of tasks. Like task entities, any task group entity inheriting from *Generic_Task_Group* may have attributes. Task group attributes constrain attributes of tasks in the task group. The type of tasks that can be in a task group are also constrained.

For example the *Transaction* entity in Figure 5.7 is used to model a transaction with period T_i represented by its attribute *period*. A *Transaction* can only contain *Periodic_Task* entities. Figure 5.4 illustrates a simple transaction Γ_i with tasks τ_{ij} and τ_{ik}. This transaction is modeled in Cheddar-ADL, in XML, with the new task group entities as shown in Figure 5.9.

With the extended meta-model, the next section presents what has been implemented in Cheddar. The implementation choices are then discussed.

5.4.3 *Discussion on Implementation*

After extending the Cheddar-ADL meta-model with DGMF tasks and transactions, code for the Cheddar-ADL classes was generated. In total, about 1800 lines of code were generated for the new task group entities and the *Frame_Task* entity. No extra entities or structures were added to the framework.

The DGMF to transaction transformation, and several tests for transactions in [5, 143, 106, 118], were implemented using the generated code. The following sections discuss some implementation choices and issues.

```
<periodic_task id="tau_ij">
 <name>tau_ij</name>
 <capacity>1</capacity>
 <offsets>
  <offset_type>
   <offset_value>2</offset_value>
   <activation>0</activation>
  </offset_type>
 </offsets>
 <jitter>2</jitter>
 <deadline>10</deadline>
 <blocking_time>0</blocking_time>
 <priority>1</priority>
 <period>8</period>
</periodic_task>
```

```
<transaction id="tdma_tasks">
 <name>TDMA_Tasks</name>
 <task_list>
  <periodic_task ref="tau_ij"/>
  <periodic_task ref="tau_ik"/>
 </task_list>
 <period>8</period>
</transaction>
```

```
<periodic_task id="tau_ik">
 <name>tau_ik</name>
 <capacity>2</capacity>
 <offsets>
  <offset_type>
   <offset_value>4</offset_value>
   <activation>0</activation>
  </offset_type>
 </offsets>
 <jitter>3</jitter>
 <deadline>10</deadline>
 <blocking_time>0</blocking_time>
 <priority>1</priority>
 <period>8</period>
</periodic_task>
```

Figure 5.9: Task Group Example

5.4.3.1 *Advantages of Implementation Solution*

The solution proposed to model transactions, introduces the task group entities but re-uses most of the Cheddar-ADL mechanism for tasks. This has an advantage in terms of meta-model and code maintenance.

The *Transaction* (resp. *Multiframe*) entity that was introduced, is generic enough to model any type of transaction (resp. multiframe task).

For transactions, the main difference between different types of transactions is their release pattern, i.e. tasks can have more or less immediate successors and predecessors [106, 118, 66]. Since the *Precedence_Dependency* entity in Cheddar-ADL is used to determine precedence between tasks, the order in which tasks are grouped in a *Transaction* does not determine their precedence dependencies. Any task precedence dependency can be represented with the *Precedence_Dependency* entity.

For multiframe tasks, a multiframe task is just a vector of frames and the *Multiframe* is a list of *Frame_Tasks*, which can be extended.

5.4.3.2 *Drawbacks of Implementation Solution*

The main issue of the implementation choice is the time performance of operations that handle precedence dependency. For example consider the *Transaction* entity. With the *Precedence_Dependency* entity, they are enough to represent any type of transaction. On the other hand, these entities may not be the best structure to represent some types of transaction.

For example, consider the operation to get the immediate successors/predecessors of a task in a transaction. A linear transaction, where tasks have at most one immediate successor/predecessor, is best represented with a table. A table reduces considerably the complexity of the operation to get the successor/predecessor of a task in a linear transaction. Indeed, with a table, the operation is $O(1)$ while in the current implementation, we have to iterate through all entries in the set of *Precedence_Dependency*, so the operation to get a successor/predecessor is $O(n)$.

The general solution to the complexity problem is to implement each type of transaction with the best adapted data structure. Then the Cheddar-ADL architecture model is transformed to the best adapted data structure for the analysis. The optimization of the implementation of schedulability tests for transactions [89] is not the purpose of this thesis. Therefore the solution to the complexity problem is not implemented. However experimental results will show that the current implementation stays scalable to a real software radio protocol.

The following section shows some experiments that evaluate the DGMF modeling and the transformation of DGMF tasks to transactions.

Table 5.4: DGMF Generator Constraints and Assumptions: GA-R are constraints; GA-H are assumptions

Ref.	Description
GA-R1	Frames have a WCET not less than 1.
GA-R2	A DGMF task has at least one frame.
GA-R3	No violation of the desired GMF period for synced DGMF tasks
GA-R4	Precedence dependencies are unique.
GA-R4	No deadlocks due to precedence dependencies
GA-H1	Synced DGMF tasks are ordered arbitrarily and F_i^j can only have F_p^q as a predecessor if $p < i$.
GA-H2	The required number of frames is respected "as best as possible"
GA-H3	The required number of precedence dependencies is respected "as best as possible"

5.5 EXPERIMENT AND EVALUATION

Three aspects of the DGMF to transaction transformation are evaluated through experiments. The following sections first evaluates the transformation correctness, then the transformation time performance, and finally the DGMF modeling and transformation scalability, when it is applied to a real case-study.

5.5.1 *Transformation Correctness*

The transformation correctness is evaluated by scheduling simulation performed on randomly generated system architecture models. Cheddar provides a function that is able to generate random architecture models. To randomly simulate scheduling of DGMF tasks and transactions, the generator is extended for these models.

5.5.1.1 *Generator Update*

The generator is updated so DGMF tasks can be produced. The generator respects some parameters defined by the user:

- Number of processors
- Scheduling policy (for all processors)
- Number of DGMF tasks
- Number of frames in total
- GMF period for synced DGMF tasks.
- Sync ratio, which is the ratio of DGMF tasks, with the same GMF period, by the total number of DGMF tasks. DGMF tasks with the same GMF period are called synced tasks in the experiments.
- Number of shared resources
- Number of critical sections
- Number of precedence dependencies between frames of different DGMF tasks

Besides constraints inherited from the DGMF task model, the generator respects some other constraints. Some assumptions are also made to ease the generation. Constraints and assumptions, enforced during generation, are described in Table 5.4.

The generator produces five kinds of entities through several steps:

- Processors

– DGMF tasks
– Frames of DGMF tasks
– Shared resources and critical sections
– Precedence dependencies

The following paragraphs show the details of each step of the generation.

GENERATE PROCESSORS This step is straightforward as the required number of processors, scheduled by the required policy, are simply added to the system architecture model.

GENERATE DGMF TASKS Again this step is straightforward as the required number of DGMF tasks (task groups) are simply added to the system architecture model.

GENERATE FRAMES OF DGMF TASKS The algorithm to generate frames of DGMF tasks proceeds in 3 steps:

– **Step** 1: Add a first frame to each DGMF task.
– **Step** 2: Distribute frames randomly among DGMF tasks and compute the frame parameters.
– **Step** 3: Verify that synced DGMF tasks have the same GMF period.

In **Step** 1, DGMF tasks are iterated through and a frame is added to each.

In **Step** 2, if the required number of frames hasn't been reached, the remaining number of frames are distributed randomly among DGMF tasks. When adding a frame $F_i^{N_i+1}$ to a DGMF task G_i with N_i frames, if G_i is a synced, a random $P_i^{N_i+1}$ is computed as a random value in $[1; P_i - P_i^{N_i}]$ If $P_i - P_i^{N_i} < 0$, then $P_i^{N_i+1} = 0$. The WCET $E_i^{N_i+1}$ of $F_i^{N_i+1}$ is computed as:

$$E_i^{N_i+1} = \text{rand}\left(\left\lfloor \frac{P_i}{2} \right\rfloor\right) \tag{100}$$

where $\text{rand}(x)$ computes a random integer in $[1; x]$.

In **Step** 3, each synced DGMF task's last frame $F_i^{N_i}$ has its $P_i^{N_i}$ increased, if necessary, so the GMF period of the synced DGMF tasks is respected.

GENERATE SHARED RESOURCES AND CRITICAL SECTIONS In this step, the the required number of shared resources are added to the system architecture model. A random resource is used in each critical section. The required number of critical sections is added by choosing a random frame each time a critical section is added.

GENERATE PRECEDENCE DEPENDENCIES The main issue with adding precedence dependencies is deadlock. To avoid deadlocks, a F_i^j frame can only have F_p^q as a predecessor frame, if $p < i$, with DGMF tasks ordered arbitrarily. G_i and G_p are synced due to the definition of DGMF tasks.

The algorithm to add frames to DGMF tasks proceeds in 2 steps:

– **Step** 1 Compute a number of precedence dependencies with guarantee of no deadlocks.
– **Step** 2 Add precedence dependencies randomly among frames of synced DGMF tasks.

In **Step** 1, a number of precedence dependencies, with guarantee of no deadlocks occurring, is computed:

$$\sum_{i=1}^{n-1}\left(N_i \times \left(\sum_{j=i+1}^{n} N_j\right)\right) \tag{101}$$

If the required number of precedence dependencies is greater than the value computed in Equation 101, then the required number of precedence dependencies is set to this value. Consider Figure 5.10. In such a system, a number of precedence dependencies of 26, guarantees no deadlocks.

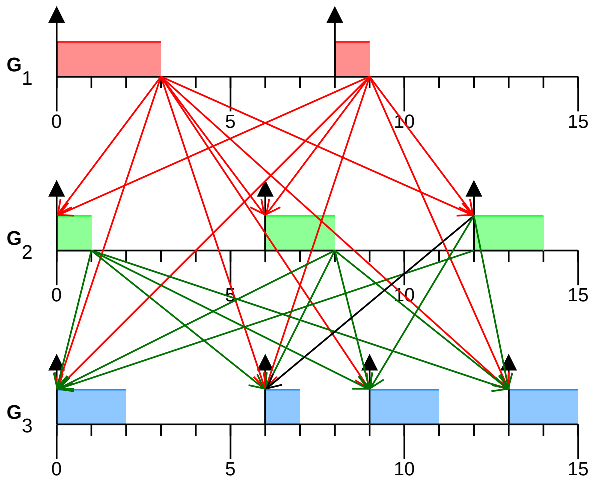


Figure 5.10: Maximum Precedence Dependencies: Arrows on timeline are releases; Boxes are frames; Arrows between boxes are precedence dependencies

In **Step** 2, two random synced DGMF tasks G_i and G_p are chosen, with $p < i$. A random frame in each is chosen (F_i^j and F_p^q) and F_p^q is added to the set of predecessor frames of F_i^j only if it is not already present and adding it won't create a deadlock. **Step 2** is repeated until the number of precedence dependencies has reached the expected number of precedence dependencies.

5.5.1.2 *Experimental Parameters*

By using the architecture generator, DGMF tasks sets are randomly generated with the following constant parameters:

- 1 processor
- Preemptive FP scheduling
- Sync ratio of 0.5 (50% synced DGMF tasks)

The varying parameters of the generator are:

- Between 2 and 5 DGMF tasks (increase by 1)
- Between 2 and 10 frames (increase by 1)
- Between 1 and 3 shared resources (increase by 1)
- A number of critical sections equal to the number of frames
- A GMF period between 10 and 50 (increase by 10)
- A number of precedence dependencies equal to the number of frames

5.5.1.3 *Results*

From these parameters, 540 DGMF architecture models are generated. Each of them is transformed to an architecture model with transactions. Both DGMF and transaction models are then simulated in the time interval proposed in [36]. Schedules are then compared. Each DGMF schedule is strictly similar to the corresponding transaction schedule.

The number of tasks, critical sections, and precedence dependencies in the transaction models are equal to the number of frames, critical sections, and precedence dependencies in the DGMF models. The number of Cheddar task groups in the transaction models is less than the number of Cheddar task groups in the DGMF models, since some transactions were merged into a same one, during the transformation.

Along with the proofs in Section 5.2.2, this experiment enforces the transformation correctness and its implementation correctness in Cheddar.

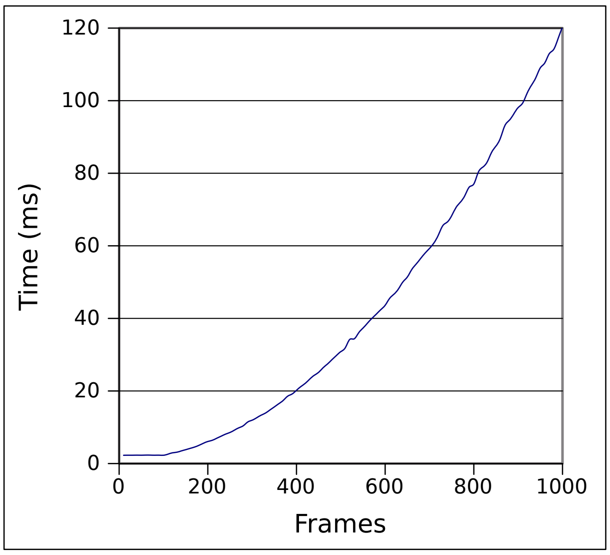


Figure 5.11: Transformation Duration by Number of Frames

5.5.2 *Transformation Time Performance*

In the Cheddar implementation, the time complexity of the transformation algorithm depends on two parameters: n_F the number of frames, and n_D the number of task dependencies (both precedence and shared resource). The complexity of the transformation is $O(n_D^2 + n_F)$. When n_F is the varying parameter, the complexity of the algorithm should be $O(n_F)$. When n_D is the varying parameter, the complexity of the algorithm should be $O(n_D^2)$ due to Algorithm 5.2, which has the same complexity as the algorithm in [34].

The experiment in this section checks that the duration of the transformation, implemented in Cheddar, is consistent with these time complexities. Measurements presented below are taken on a Intel Core i5 @ 2.40 GHz processor.

Figure 5.11 shows the transformation duration by the number of frames. The number of precedence dependencies is set to 0 (i.e. no intra-dependencies either).

Figure 5.11 shows that the duration is polynomial when the number of frames varies. One can think that this result is inconsistent with the time complexity of the algorithm, which should be $O(n_F)$ when n_F is the varying parameter. In practice, the implementation in Cheddar introduces a loop to verify that a task is not already present in the system's tasks set. Thus the time complexity of the implementation is $O(n_F^2)$.

Figure 5.12 shows the transformation duration by the number of precedence dependencies. The number of frames is set to 1000, the number of DGMF tasks to 100, and the number of shared resources to 0. In the Cheddar implementation it does not matter which dependency parameter (precedence or shared resource) varies to verify the impact of n_D. Indeed, all dependencies are iterated through once and precedence dependency has more impact on the transformation duration. Since there are 1000 frames and 100 DGMF tasks, the minimum number of precedence dependencies starts at 900, due to intra-dependencies.

The transformation duration should be polynomial when the number of precedence dependencies vary but this is not apparent in Figure 5.12 due to the scale of the figure. Indeed, there is already a high minimum number of precedence dependencies starting at 900. The local minimum of the polynomial curve is at 0, like in Figure 5.11. The curve is a polynomial curve and the result is consistent with the complexity, which is $O(n_D^2)$ when n_D is the varying parameter.

Overall a system with no dependency, 1000 frames, and 100 DGMF tasks, takes about 120 ms to be transformed on the computer used for the experiment. A system with 1100 precedence dependencies, 1000 frames, and 100 DGMF tasks, takes less than 160 ms to be transformed. The

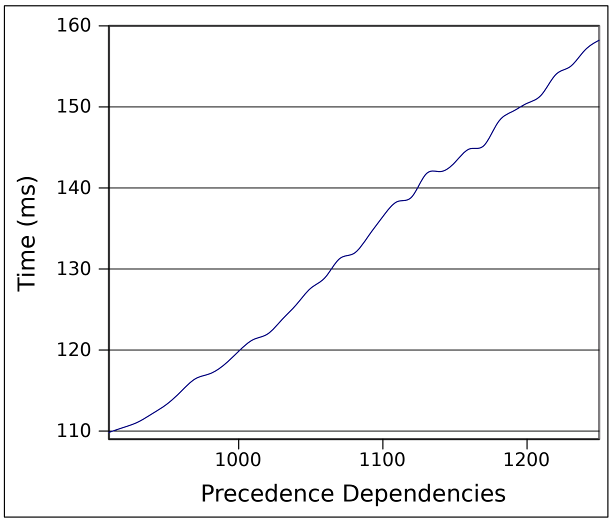


Figure 5.12: Transformation Duration by Number of Precedence Dependencies

transformation duration is acceptable for Thales. Indeed, a typical Thales system has 10 tasks and a TDMA frame of 14 slots (1 S, 5 B, 8 T). There would be a maximum of 140 frames (14×10), and 256 precedence dependencies ($10 \times 14 - 10 + 9 \times 14$) if all tasks are part of a same precedence dependency graph, where a task releases at most one task, and a first task is released at each slot.

5.5.3 *Case-Study Modeling with DGMF*

The scalability of the DGMF task model, and its transformation to transaction, is assessed by applying DGMF for the modeling of a real software radio protocol developed at Thales.

The case-study is implemented with 8 POSIX threads (threads) [28] on a processor called *GPP1*, and 4 threads on a processor called *GPP2*. The threads are scheduled by the SCHED_FIFO scheduler of Linux (preemptive FP policy).

The threads have precedence dependencies, whether they are on the same processor or not. For example threads on *GPP1* may make blocking calls to functions handled by threads on *GPP2*. When a thread makes a blocking call to a function, it has to wait for the return of the function, before continuing execution. There is one thread on *GPP2* dedicated to each thread on *GPP1* that may make a blocking call.

The TDMA frame of the case-study has 1 *S* slot, 5 *B* slots, and 8 *T* slots. The threads are released at different slots. The release logic is a thread dedicated to reception is released at a *Rx* slot, while a thread dedicated to transmission is released so its deadline coincides with the start of a *Tx* slot.

In total, there are 36 jobs from 8 threads on *GPP1*, and 4 threads on *GPP2*. The threads, their precedence dependencies, and the time parameters of their jobs are illustrated in Figure 5.13.

The case-study is modeled with a DGMF task model of 43 frames. There are more frames than the 36 jobs because some jobs that make a blocking call to a function, are divided into several frames. After the transformation, a transaction of 44 tasks is the result. The extra task, compared to the 43 frames, is a ghost root task added by a test like [118]. The transaction is illustrated in Figure 5.14.

Notice that different jobs of a thread, released at different slots, become non-immediate tasks in the transaction, related by precedence dependency (e.g. $RS\_B1 \prec RS\_B2$). Furthermore, a thread that makes a blocking call, to a function handled by a thread on *GPP2*, is divided into several frames, and then several tasks (e.g. *Frame_Cycle* becomes $FC\_S1\_1 \prec FC\_S1\_2 \prec FC\_S1\_3$).

Notice also it seems that some tasks should have two predecessors. For example we should have $AB\_B1\_1 \prec AB\_B2\_1$ because these two tasks represent two jobs of a same thread. We should also

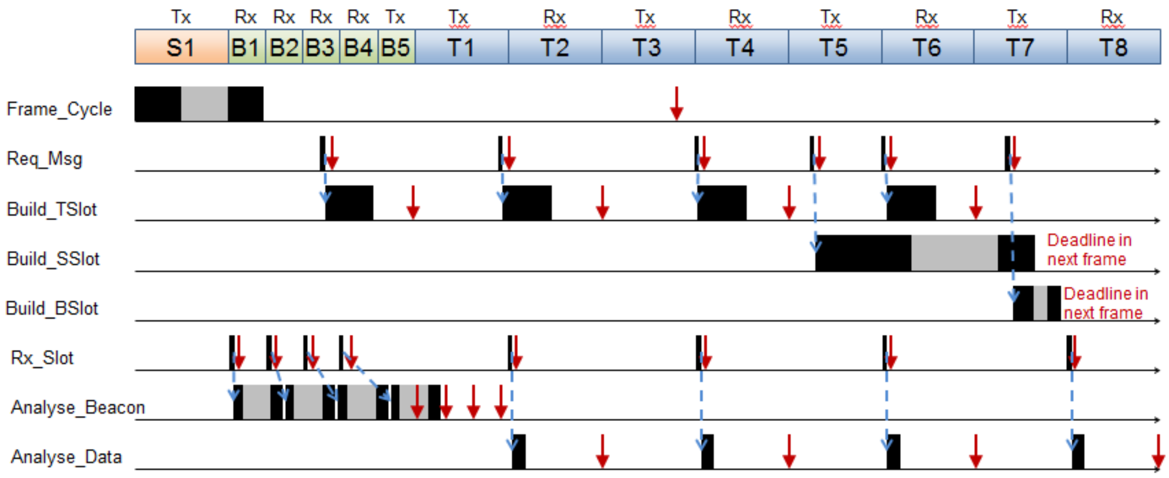


Figure 5.13: TDMA Frame and Threads of Real Case-Study: Line = Instances of a thread; Down arrow = Deadline; Dashed arrow = Precedence; Black = Exec on *GPP1*; Gray = Exec on *GPP2*; 36 jobs in total from 8 threads on *GPP1* and 4 threads on *GPP2*; Sizes not proportional to time values

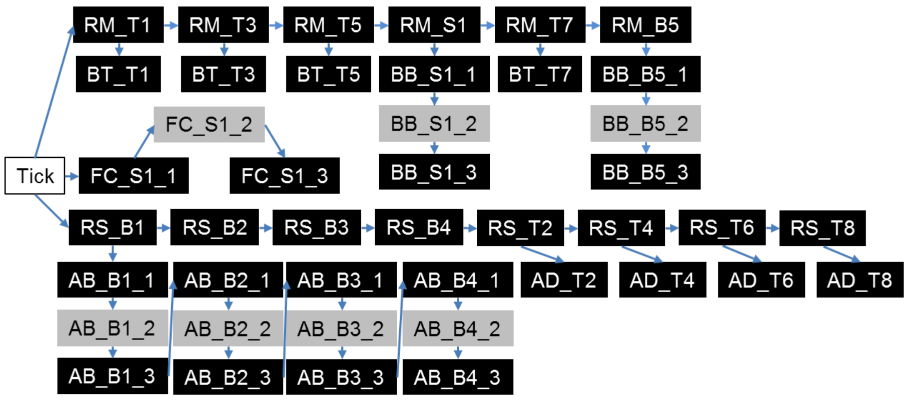

Tick
RM_T1 RM_T3 RM_T5 RM_S1 RM_T7 RM_B5
BT_T1 BT_T3 BT_T5 BB_S1_1 BT_T7 BB_B5_1
FC_S1_2 BB_S1_2 BB_B5_2
FC_S1_1 FC_S1_3 BB_S1_3 BB_B5_3
RS_B1 RS_B2 RS_B3 RS_B4 RS_T2 RS_T4 RS_T6 RS_T8
AB_B1_1 AB_B2_1 AB_B3_1 AB_B4_1 AD_T2 AD_T4 AD_T6 AD_T8
AB_B1_2 AB_B2_2 AB_B3_2 AB_B4_2
AB_B1_3 AB_B2_3 AB_B3_3 AB_B4_3

Figure 5.14: Tree-Shaped Transaction of Real Case-Study: Black tasks on *GPP1*, gray tasks on *GPP2*, Tick task is ghost root task; Task nomenclature is [Name]_[Slot]_[Part (Optional)]; Abbreviated names are RM = *Request_Msg*, BT = *Build_TSlot*, BB = *Build_BSlot*, BS = *Build_SSlot*, FC = *Frame_Cycle*, RS = *Rx_Slot*, AB = *Analyse_Beacon*, AD = *Analyse_Data*

have $Rx\_Slot\_B2 \prec AB\_B2\_1$ since these two tasks represent jobs of two threads with a precedence dependency. But since the offset of AB_B2_1 is strictly greater than the deadline of Rx_Slot_B2, the precedence dependency $RS\_B2 \prec AB\_B2\_1$ is reduced by the DGMF transformation. Other cases of tasks with multiple predecessors are due to the same kind of precedence dependencies as the example. Multiple predecessors are thus reduced to one in the same way as the example, by the transformation. In the end the resulting transaction is tree-shaped.

In general, the more jobs of a thread there are, the more frames there are. Similarly, the more there are blocking calls to functions allocated on other processors, the more frames there are.

Notice that the DGMF task model has 43 frames to represent the 12 threads of case-study. The DGMF tasks set should thus be produced automatically from an architecture model.

5.6 CONCLUSION

In this chapter a task model was proposed for characteristics of a software radio protocol implementing the TDMA channel access method. The proposed task model is called DGMF and it extends the GMF task model with task dependencies (shared resource and precedence). DGMF is applicable to partitioned multiprocessor systems.

The DGMF analysis method is divided into two parts. This chapter focused on the first part, which consists in transforming a system with DGMF tasks to one with transactions. The correctness of the transformation was proven and verified through simulation. The time performance of the transformation was also evaluated and deemed acceptable for a Thales case-study.

Indeed, experimental results showed that the implementation of the transformation algorithm in Cheddar is polynomial in practice. It was also observed that less than 160 ms are taken to transform a system with more frames and dependencies than a system developed at Thales.

After transforming DGMF tasks to transactions, characteristics of the resulting transactions were determined. These transactions are tree-shaped and they have non-immediate tasks. These tasks are not necessarily released immediately by their predecessor. A schedulability test for this type of transaction is proposed in the next chapter.

PUBLICATIONS The DGMF task model and its analysis method are published in [84, 81].

Chapter 6

SCHEDULABILITY ANALYSIS OF TREE-SHAPED TRANSACTIONS WITH NON-IMMEDIATE TASKS

Previously it was shown that DGMF tasks can be transformed into transactions for schedulability analysis. The transactions that are the result of this transformation are tree-shaped and they have non-immediate tasks that are not necessarily released immediately by their predecessor.

Schedulability tests for tree-shaped transactions have been proposed [118] but they do not handle non-immediate tasks. Such tasks have a consequence on the schedulability analysis results. In this chapter a schedulability test is proposed for tree-shaped transactions with non-immediate tasks. The test is called WCDOPS+NIM and it extends the WCDOPS+ test proposed in [118]. The proposed test completes the general DGMF analysis method presented in the previous chapter.

In the following sections, first the consequence of non-immediateness is shown. Then the extension of the WCDOPS+ test in [118] is exposed. Finally experiments first show some simulations that compare the WCDOPS+ test to the WCDOPS+NIM test. The proposed test is then applied to real case-studies from Thales. This chapter uses notations of the transaction model, presented in Section 2.4.1.

6.1 APPLICABILITY OF WCDOPS+ ON NON-IMMEDIATE TASKS

To evaluate the applicability of WCDOPS+ on non-immediate tasks, consider the example in Figure 6.1. The figure shows an example of tasks in two software radio protocol layers arbitrarily named *L2* and *L3*. There is a TDMA frame with two slots. Transaction Γ_1 is used to model tasks constrained by the TDMA frame. Clearly Γ_1 is a tree-shaped transaction.

Task *IO1* is released at slot 1. *IO1* then leads to the releases of other tasks. Task *PR22* cannot be released earlier than slot 2 and it can only be released if *PR21* has completed execution. Tasks *IO1* and *IO2* have lower priorities than all other tasks. *IO1* and *IO2* are input-output tasks and they must not preempt the processing tasks, which are *PR1*, *PR21*, *PR22*, *PR3*, and *PR4*.

Transaction Γ_2 has a single task, called *MGT*, that is released periodically. This kind of task has a period greater or equal to the TDMA frame duration, and it must not be delayed by more than a TDMA frame duration. For this reason, *MGT* has a priority higher than *IO1* and *IO2*, to ensure that *MGT* won't be preempted too long after its release.

Consider WCDOPS+ applied directly to the system in Figure 6.1. It is assumed that each task of Γ_i has a WCET of 1. Then we have *PR22* a non-immediate successor of *PR21*, because *PR22* is released at earliest at $t = 4$ and *PR21* can complete execution as early as $t = 3$.

If WCDOPS+ is applied directly for the analysis of *MGT*, since non-immediateness is not handled by WCDOPS+, the test considers that *PR22* can only execute with at most *PR1* and *PR21* in the same *MGT* busy period. This is not true as shown by one possible schedule in Figure 6.2. If, for example, *PR3* has a WCET equal to 2 instead, the interference of Γ_1 to *MGT* is underestimated.

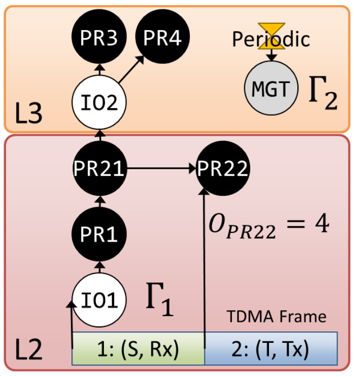


Figure 6.1: Tree-Shaped Transaction with Non-Immediate Tasks: Black circles are high-priority tasks; Gray circles are mid-priority tasks; White circles are low-priority tasks; TDMA slots have a duration of 4 time units; Slot "1: (S, Rx)" is of type Service (S), mode Reception (Rx); Slot "2: (T, Tx)" is of type Traffic (T), mode Transmission (Tx)

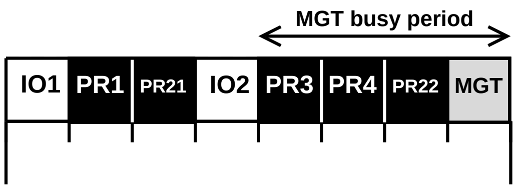


Figure 6.2: Underestimation due to Non-Immediateness: *PR3*, *PR4*, and *PR22* in same *MGT* busy period

A possible solution to have the correct combinations of tasks that can interfere, is to model the non-immediateness between two tasks by *ghost intermediate tasks*. Before defining a ghost intermediate task, the terminology of tree-shaped transaction entities must first be modified, due to non-immediate tasks.

Definition 59 (Direct Predecessor and Direct Successors of Tree-shaped Transaction)**.** *A task τ_{ij} is said to have one **direct** predecessor, denoted by $\mathrm{pred}(\tau_{ij})$, and a set of **direct** successors, denoted by $\mathrm{succ}(\tau_{ij})$. A task τ_{ix} is $\mathrm{pred}(\tau_{ij})$ (resp. in $\mathrm{succ}(\tau_{ij})$) if there is no task τ_{iy} such that $\tau_{ix} \prec \tau_{iy} \prec \tau_{ij}$ (resp. $\tau_{ij} \prec \tau_{iy} \prec \tau_{ix}$). For the root task of a tree-shaped transaction, $\mathrm{pred}(\tau_{i1})$ is undefined.*

The concept of ghost tasks is introduced in [118] (Definition 55). The concept of intermediate tasks is proposed in [52]. An intermediate task represents some extra execution time or message transmission time. A ghost intermediate task can be defined as follows:

Definition 60 (Ghost Intermediate Task)**.** *A ghost intermediate task τ_{ixy} is a task between τ_{ix} and its non-immediate direct successor τ_{iy} ($\tau_{ix} \prec \tau_{iy}$). Precedence dependency $\tau_{ix} \prec \tau_{iy}$ is replaced by $\tau_{ix} \prec \tau_{ixy} \prec \tau_{iy}$. Ghost intermediate task τ_{ixy} is allocated alone on a processor, modeled only for the ghost intermediate task. Parameters of task τ_{ixy} are formally defined as follows:*

$$\begin{aligned}
&C_{ixy} = C^{b}_{ixy} = O_{iy} - (O_{ix} + C^{b}_{ix} \\
&O_{ixy} = O_{ix} + C^{b}_{ix} \\
&J_{ixy} = R^{w}_{ix} - O_{ixy} \\
&D_{ixy} = \infty \\
&B_{ixy} = 0 \\
&\mathrm{prio}(\tau_{ixy}) = 1 \\
&\forall \tau_{kl}, \mathrm{proc}(\tau_{ixy}) \neq \mathrm{proc}(\tau_{kl})
\end{aligned} \tag{102}$$

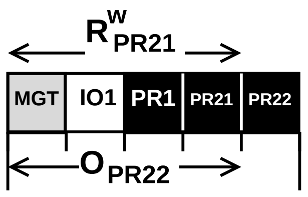


Figure 6.3: Non-Immediateness and Jitter: R^w_{PR21} increases due to *MGT* ($C_{MGT} = 1$) preempting *IO1*. R^w_{PR21} is not longer than O_{PR22} therefore J_{PR22} does not increase.

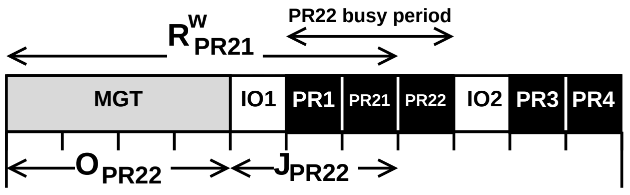


Figure 6.4: Non-Immediateness and Interference: *PR22* experiences jitter, since R^w_{PR21} is longer than O_{PR22}, due to the preemption of *IO1* by *MGT* ($C_{MGT} = 4$). *PR22* is then released immediately by *PR21*. Therefore *IO2* cannot execute between *PR21* and *PR22*, and only after *PR22* completes execution. *PR3* and *PR4* are released by *IO2* so they execute after the busy period of *PR22*. Thus *PR3* and *PR4* cannot interfere *PR22*.

Adding ghost intermediate tasks introduces pessimism to WCRT computation. For example if a ghost intermediate tasks is added between *PR21* and *PR22*, any increase in R^w_{PR21} will increase J_{PR22} and thus R^w_{PR22}. This is not always the case, as shown by a possible schedule in Figure 6.3, so R^w_{PR22} can be overestimated.

Furthermore, with a ghost intermediate task, the test considers that *PR3* and *PR4* can interfere *PR22* even if *PR22* experiences jitter, since *IO2* is allowed to execute during the execution of the ghost intermediate task on another processor. Then both J_{PR22} and interference from *PR3* and *PR4*, contribute to R^w_{PR22}. This is not possible as shown by the schedule in Figure 6.4, so R^w_{PR22} can again be overestimated.

In summary, two problems are observed when applying WCDOPS+ to transactions with non-immediate tasks. First, if applied directly, tasks interference may be underestimated. Second, by modeling non-immediate tasks with ghost intermediate tasks, jitter and task interference can both be overestimated. In conclusion one problem leads to underestimated WCRTs and the other to overestimated WCRTs. In the following section a test proposes to solve these problems by considering the effects of non-immediateness directly.

6.2 A SCHEDULABILITY TEST FOR NON-IMMEDIATE TASKS

To consider the effects of non-immediateness directly, this thesis proposes the Worst Case Dynamic Offset with Priority Schemes Plus for Non-Immediate tasks (WCDOPS+NIM) schedulability test, which extends the original WCDOPS+ test.

6.2.1 *Overview of the RTA Method*

The general approach of the WCDOPS+NIM algorithm is the same as the WCDOPS+ algorithm, presented in Section 2.4.4, and illustrated with Figure 2.7. The system to analyze has some transactions Γ_i. It is assumed that the WCRT of τ_{ab}, belonging to Γ_a, is computed.

The WCDOPS+ algorithm first defines some tasks sets to help the analysis of τ_{ab} **(Op0)**. Then the WCRT of τ_{ab} is computed for each scenario where a τ_{ac} starts the τ_{ab} busy period **(Op1)**. Within a scenario, the algorithm computes the WCRT of each job p_{ab} of τ_{ab}, released in the τ_{ab} busy period of length w. To compute the WCRT of job p_{ab} of τ_{ab}, various interferences, from transactions in the system, are computed **(Op2)**. The WCRT of τ_{ab} is then the maximum of WCRTs of each job p_{ab} of each scenario **(Op3)**.

The algorithm has several key operations that need some adaptations for non-immediate tasks. In the following sections each key operation is explained, focusing on their difference with the original WCDOPS+ algorithm.

6.2.2 *Op0: Task Sets and Execution Conflicts*

Op0 consists in defining some tasks sets to help the analysis of τ_{ab} when resolving execution conflicts.

As a reminder, two sets of tasks are defined in [118]. An H segment is a set of tasks that must all execute within a same τ_{ab} busy period. Two tasks in hp_i belong to the same H segment if there is no other task that is not in hp_i that precedes one but not the other. An H section is a set of tasks that may execute in the same τ_{ab} busy period. Two tasks in hp_i belong to the same H section if there is no other task in lp_i that precedes one but not the other.

These sets are adapted for non-immediate tasks. An H segment is re-defined by modifying its conditions: there is also no non-immediate predecessor that precedes one of the task but not the other, and there is no non-immediate predecessor that is a direct predecessor of both tasks. Formally, the new definition of an H segment is then:

$$\begin{aligned} H_{ij}^{seg}(\tau_{ab}) = \{\tau_{im} \mid \tau_{im} \in hp_i(\tau_{ab}) \wedge \\ (\neg\exists \tau_{il} \in \Gamma_{ij} \Delta \Gamma_{im} \mid \tau_{il} \notin hp_i(\tau_{ab}) \\ \vee \neg(\tau_{il} = \tau_{i1} \vee O_{il} > O_{pred(\tau_{il})} + C_{pred(\tau_{il})}))\} \end{aligned} \tag{103}$$

The definition of an H section does not need any modification since a non-immediate successor may belong to the same busy period as its predecessor (both in hp_i).

Consider the system in Figure 6.1, assuming *MGT* is under analysis. Sets {*PR1*, *PR21*}, {*PR3*, *PR4*}, and {*PR22*} are H segments. Set {*PR1*, *PR21*, *PR22*} is an H section.

6.2.3 *Op1: Worst Case Scenario*

Op1 consists in creating a scenario where $\tau_{ik} \in \Gamma_i$ (resp. $\tau_{ac} \in \Gamma_a$) starts the τ_{ab} busy period at t_c. Two modifications are necessary for **Op1**. First the set of tasks, that may start the τ_{ab} busy period, must be modified. When a task starts the τ_{ab} busy period, its jitter may also need modification. The following sections describe these modifications.

6.2.3.1 *Modifications to* XP_i

In [118], tasks in Γ_i that may start the busy period are in a set $XP_i(\tau_{ab})$, which is the set of tasks that come first in their respective H segments.

XP_i must re-defined due to non-immediate tasks. It is a set that contains tasks in hp_i whose predecessors are not in hp_i, but the set also tasks in hp_i that are non-immediate successors.

$$\begin{aligned} XP_i(\tau_{ab}) = \{\tau_{if} \in hp_i(\tau_{ab}) \mid pred(\tau_{if}) \notin hp_i(\tau_{ab}) \vee \\ \neg(\tau_{il} = \tau_{if} \vee O_{if} > O_{pred(\tau_{if})} + C_{pred(\tau_{if})}))\} \end{aligned} \tag{104}$$

For example in Figure 6.1, *PR1*, *PR22*, *PR3* and *PR4* are in $XP_i(MGT)$.

6.2.3.2 *Jitter Canceling*

Consider that a task $\tau_{ik} \in XP_i$ starts a τ_{ab} busy period. If τ_{ik} is not an immediate successor, then it may not be able to start the τ_{ab} busy period, if τ_{ik} is released immediately by $\mathtt{pred}(\tau_{ik})$. For example, a non-immediate successor τ_{ik} cannot start the τ_{ab} busy period if it is released immediately by its direct predecessor in hp_i. Therefore the following theorem applies if τ_{ik} starts the τ_{ab} busy period:

Theorem 19. *Let τ_{ik} be a task in XP_i that starts a τ_{ab} busy period. If τ_{ik} is a non-immediate successor, and $\mathtt{pred}(\tau_{ik}) \in hp_i$, then τ_{ik} must not experience jitter to be released at t_c. If $\mathtt{pred}(\tau_{ik}) \notin hp_i$, the scenario where τ_{ik} experiences enough jitter to be released at t_c must also be analyzed.*

Proof. Assume that $\tau_{ik} \in XP_i$ is a non-immediate successor that starts the τ_{ab} busy period. If τ_{ik} experiences jitter in a scenario, then it means that the response time of $\mathtt{pred}(\tau_{ik})$ is greater than the offset of τ_{ik}. τ_{ik} is then released immediately after $\mathtt{pred}(\tau_{ik})$ completes execution. If $\mathtt{pred}(\tau_{ik}) \in hp_i(\tau_{ab})$ then, according to Lemma 5-1 in [106], τ_{ik} cannot start the busy period, which contradicts the initial assumption. The case where τ_{ik} experiences jitter is analyzed in the scenario where the H segment of $\mathtt{pred}(\tau_{ik})$ starts the τ_{ab} busy period.

If $\mathtt{pred}(\tau_{ik}) \notin hp_i(\tau_{ab})$ then there is no predecessor H segment that may start the busy period with τ_{ik} experiencing jitter. This is why, in this case, the scenario where τ_{ik} starts the busy period, after having experienced J_{ik}, is also analyzed. □

For example in Figure 6.1, if *PR22* starts the *MGT* busy period, then the jitter of *PR22* is canceled.

To create a scenario where τ_{ij} is released at t_c without experiencing jitter, J_{ij} is set to 0. This is what is called *jitter canceling*. In this case, we say that J_{ij} is canceled.

Definition 61 (Jitter Canceling). *Jitter canceling is the operation of setting the jitter J_{ij} of a task τ_{ij} to 0, for the purpose of some analysis.*

To integrate jitter canceling in the schedulability test, whenever the algorithm iterates through τ_{ik} tasks in XP_i to create scenarios, it memorizes the original value of J_{ik}, sets J_{ik} to 0, compute interference for the scenario, and afterwards resets J_{ik} to the memorized value. It also creates the scenario where J_{ik} is not canceled if $\mathtt{pred}(\tau_{ik}) \notin hp_i$, so interference for both scenarios can be compared.

6.2.4 *Op2: Worst Case Interference*

Op2 consists in computing the interference from transactions to job p_{ab} of τ_{ab}. Like WCDOPS+, the WCDOPS+NIM test computes two kinds of interference for a transaction: blocking and non-blocking. As a reminder, the existence of blocking interference is due to execution conflicts. Only one blocking interference from any transaction in the system, can contribute to the τ_{ab} busy period. If a transaction's blocking interference is not chosen to contribute, then its non-blocking interference contributes to the τ_{ab} busy period.

The following sections show how to compute interference of jobs of a transaction Γ_i and Γ_a before or at t_c, then jobs after t_c, and finally the total interference. In these sections, it is assumed that task τ_{ik} from XP_i starts the τ_{ab} busy period, of length w, at $t_c = 0$.

6.2.4.1 *Jobs before or at t_c ($p \leqslant 0$)*

Blocking interference and non-blocking interference are computed for jobs $p \leqslant 0$ of Γ_i similarly to the original test in [118] but some differences exist due to non-immediate tasks.

Before computing the interferences of a job p of Γ_i, the WCDOPS+NIM test checks which non-immediate successors τ_{ij}, at job p, are released immediately in the scenario where τ_{ik} starts the τ_{ab} busy period.

Theorem 20. *A non-immediate successor task $\tau_{ij} \in hp_i$ is released immediately by $pred(\tau_{ij})$, at job p, when τ_{ik} starts the τ_{ab} busy period, if τ_{ij} is released before $t_c = 0$:*

$$\varphi_{ijk} + (p-1) \times T_i < 0 \tag{105}$$

Proof. Let $\tau_{ij} \in hp_i$ be a non-immediate successor. Value $\varphi_{ijk} + (p-1) \times T_i$ is the release time of τ_{ij} at job p, when τ_{ik} starts the τ_{ab} busy period. If τ_{ij} is released before $t_c = 0$, τ_{ij} needs to have experienced enough jitter to be released at t_c [106]. If τ_{ij} experiences jitter, then τ_{ij} is immediately released by $pred(\tau_{ij})$. □

For example, in Figure 6.4, if *PR1* starts a busy period after having experienced $J_{PR1} = 4$, *PR22* is released at $t = -1$ and experiences a jitter of 1 to be released at t_c. *PR22* is thus released immediately by *PR21* and belongs the same H segment as *PR21*.

Thus, checking if a non-immediate successor $\tau_{ij} \in hp_i$ is to be considered immediately released by $pred(\tau_{ij}) \in hp_i$ also determines to which H segment τ_{ij} belongs to in the given scenario: either the H segment of τ_{ij} or the H segment of $pred(\tau_{ij})$. This has an effect on blocking interference computation. Therefore when computing the interference of job p of Γ_i, in the scenario where τ_{ik} starts the τ_{ab} busy period, the definition of an H segment is the following:

$$\begin{aligned} H_{ij}^{seg'}(\tau_{ab}, p, \tau_{ik}) = \{\tau_{im} \mid \tau_{im} \in hp_i(\tau_{ab}) \wedge \\ (\neg \exists \tau_{il} \in \Gamma_{ij} \Delta \Gamma_{im} \mid \tau_{il} \notin hp_i(\tau_{ab}) \\ \vee \neg(\tau_{il} = \tau_{i1} \vee (O_{il} > O_{pred(\tau_{il})} + C_{pred(\tau_{il})} \wedge \varphi_{ilk} + (p-1) \times T_i \geqslant 0))))\} \end{aligned} \tag{106}$$

Like in [118], the blocking interference and the non-blocking interference of a particular job p of Γ_i are computed by exploring the tree representing the transaction. This gives the combination of H segments, that gives the highest blocking and non-blocking interferences for job p.

Since the definition of an H segment is different for WCDOPS+NIM, the test does not give the same combination of H segments. There are more H segments that can interfere together. This can be explained as if a ghost intermediate task is defined between a task and its non-immediate predecessor. Therefore there are more non-blocking H segments that are not in blocking conflict.

Like in [118], some H segments cannot interfere together because they are in blocking or precedence conflict. Since there are non-immediate tasks, there is a new precedence conflict due to the following theorem:

Theorem 21. *Let τ_{ik} be a non-immediate successor that starts the τ_{ab} busy period at t_c, with J_{ik} canceled. A job p of a task $\tau_{ij} \in hp_i(\tau_{ab})$, that precedes τ_{ik}, does not contribute to the τ_{ab} busy period, if $p \leqslant p_{0,ikk}^{seg}$.*

Proof. Assume that $\tau_{ik} \in XP_i$ is a non-immediate successor that starts the τ_{ab} busy period, and J_{ik} is canceled. Task τ_{ik} does not experience jitter and is released at t_c. If any job of a task that precedes τ_{ik}, earlier than or same as $p_{0,ikk}^{seg}$, executes in the τ_{ab} busy period, then the task executes after t_c. If a preceding task executes after t_c, then τ_{ik} is not released at t_c. This contradicts the assumption that τ_{ik} is released at t_c. □

For jobs $p \leqslant 0$ of Γ_a, the blocking and non-blocking interferences computation does not need more modifications than the Γ_i case.

The computation of interferences of jobs $p \leqslant 0$ of Γ_i and Γ_a is performed with three functions that are described in a later Section 6.2.5.4.

6.2.4.2 *Jobs after* t_c *(*$p > 0$*)*

For jobs $p > 0$, the original test does not need any modification. Indeed, jobs $p > 0$ of τ_{ij} can only interfere τ_{ab} if τ_{ij} belongs to the first H section of Γ_i, and the first H segment is not a blocking segment. For Γ_a, jobs $p > 0$ of task τ_{aj} can only interfere τ_{ab} if τ_{aj} at least belongs to the first H section of Γ_a and the first H segment is not a blocking segment.

If a non-immediate successor $\tau_{ij} \in hp_i$, of $\mathtt{pred}(\tau_{ij}) \in hp_i$, is in the first H section, we do not need to check if τ_{ij} belongs to its own H segment or the H segment of $\mathtt{pred}(\tau_{ij})$, because both H segments belong to the first H section. The same applies for a non-immediate successor $\tau_{aj} \in hp_a$.

6.2.4.3 *Total interference*

No other modifications are necessary for the computation of the total interference from all transactions Γ_i and Γ_a.

6.2.5 *Op2 cont.: Worst Case Interference Algorithms*

The previous sections showed that interference must be computed for jobs $p \leqslant 0$ of transactions Γ_i and transaction Γ_a. Interference must also be computed for jobs $p > 0$ of Γ_i and Γ_a. No modifications are necessary for jobs $p > 0$. This section shows how modifications for jobs $p \leqslant 0$ are introduced into the algorithms that compute interferences of jobs $p \leqslant 0$.

As a reminder, the interferences of jobs $p \leqslant 0$ of Γ_i are computed with three functions that have the following objectives:

– **(f1)** Compare/sum interference of each job $p \leqslant 0$ of Γ_i
– **(f2)** Compute interference of a particular job p of Γ_i
– **(f3)** Compute interference of a particular task of job p of Γ_i

Before presenting the modifications to these functions, the definition of non-immediateness (Definition 58) is integrated into a `IM` function in Algorithm 6.1. The `IM` function is used by the interference computation functions. This function checks if the direct successor τ_{ij} of a task τ_{ip} is immediate.

Algorithm 6.1 Immediate Function

```
1: function IM($\tau_{ip}, \tau_{ij}$)
2:     return $\tau_{ip}$ = undefined $\vee$ $O_{ij} > O_{ip} + C_{ip}$ $\vee$ IS_IMMEDIATE($\tau_{ij}$)
3: end function
```

IS_IMMEDIATE(τ_{ij}) returns true for a non-immediate task τ_{ij}, when the interference of a job of τ_{ij} is computed within a scenario, and the job of τ_{ij} must be released immediately by the predecessor of the τ_{ij} (Theorem 20). Otherwise IS_IMMEDIATE(τ_{ij}) returns false. Without loose of generality, this will simplify explanations of algorithms.

In the following paragraphs, the three interference computation functions are modified to take into account non-immediate tasks.

6.2.5.1 ***(f1)*** *TransactionInterference*

As a reminder, interference from jobs before or at t_c is computed by the `TransactionInterference` function. This function returns a transaction's blocking and non-blocking interference. It iterates through each pending job p of Γ_i, released before or at t_c, that may interfere. Assuming tasks are ordered by increasing offsets in Γ_i, the first pending job of Γ_i that may interfere is the first pending job of its last task's H segment: $p^{seg}_{0,iNk}(\tau_{ab})$ [118] computed by Equation 37 applied to the first task of H^{seg}_{iN}, with τ_{iN} being the last task of Γ_i. Algorithm 6.2 shows the modification of `TransactionInterference` for WCDOPS+NIM.

Algorithm 6.2 Modified TransactionInterference Function

```
1: function TransactionInterference($\tau_{ab}, \tau_{ik}, w, \tau_{ac}$)
2:     Add ghost root task $\tau_{i0}$ as predecessor to $\tau_{i1}$
3:
4:     for p in $p^{seg}_{0,iNk}(\tau_{ab})$..0 do
5:         for $\tau_{ij} \in \Gamma_i$ do
6:             if $\tau_{ij} \in hp_i(\tau_{ab}) \wedge \neg IM(pred(\tau_{ij}), \tau_{ij}) \wedge (\varphi_{ijk} + (p-1) \times T_i) < 0$ then
7:                 Make IS_IMMEDIATE($\tau_{ij}$) return true
8:             end if
9:         end for
10:
11:        [jobI, jobDelta] ← BranchInterference($\tau_{ab}, \tau_{ik}, \tau_{i0}, w, p, \tau_{ac}$)
12:        transI_NoB ← transI_NoB + jobI
13:        transDelta ← max(transDelta, jobDelta)
14:
15:        for $\tau_{ij} \in \Gamma_i$ do
16:            Make IS_IMMEDIATE($\tau_{ij}$) return false
17:        end for
18:    end for
19:
20:    transI_B ← transI_NoB + transDelta
21:    return [transI_NoB, transI_B]
22: end function
```

At line 5 to 9, the algorithm checks which non-immediate successors τ_{ij}, at job p, are released immediately in the scenario where τ_{ik} starts the τ_{ab} busy period. This is done according to Theorem 20.

6.2.5.2 *(f2) BranchInterference*

As a reminder, to compute the interference of a particular job $p \leqslant 0$ of Γ_i, since the transaction is tree-shaped, the tree is explored by a depth-first search algorithm in the `BranchInterference` function. The tree is explored by branches defined by tasks denoted by τ_{iB}. The general idea is to compute interference of a branch and compare/sum it with interference from sub-branches SB (branches that arrive after it in the tree). Algorithm 6.3 shows the modification of `BranchInterference` for WCDOPS+NIM.

In Algorithm 6.3, the modified definition of a branch-defining task τ_{iB} is:

$$\tau_{iB} \notin hp_i \vee \neg IM(pred(\tau_{iB}), \tau_{iB}) \tag{107}$$

For example, in Figure 6.1, *IO1*, *IO2*, and *PR22* define branches, when analyzing *MGT*.

Compared to [118], sub-branches of τ_{iB} (SB) can now contain hp_i tasks. For example, in Figure 6.1, *IO2* and *PR22* are in SB of the branch defined by *IO1*.

Due to the existence of non-immediate tasks, the exploration and computation of interference need some modifications at lines 3, 10, and 22. These modifications model the existence ghost intermediate task between a task τ_{ij} and its non-immediate predecessor $pred(\tau_{ij})$, so correct values are returned by the function. Their details are given below.

At line 3, to see if τ_{iB} precedes immediately an H segment, the `BranchInterference` function checks if there is a task τ_{im} in $succ(\tau_{iB})$ such that $\tau_{im} \in hp_i$. We have to add the condition that τ_{im} is released immediately by τ_{iB} because direct successors of τ_{iB} are not necessarily immediate and interference is only computed if τ_{iB} precedes immediately an H segment.

At line 10, since τ_{iB} can be in hp_i, after the interference of the H section it precedes (*sectionI*) is computed, we need to add interference of τ_{iB} itself if $\tau_{iB} \in hp_i$.

Algorithm 6.3 Modified BranchInterference Function

```
1: function BranchInterference($\tau_{ab}, \tau_{ik}, \tau_{iB}, w, p, \tau_{ac}$)
2:     SB ← $succ(\tau_{iB})$
3:     if $\exists \tau_{im} \in SB \mid \tau_{im} \in hp_i(\tau_{ab}) \wedge IM(\tau_{iB}, \tau_{im})$ then
4:         $S \leftarrow \{\tau_{il} \in H_{im}(\tau_{ab}) \mid \tau_{iB} \prec \tau_{il}\}$
5:         sectionI ← $\sum_{\tau_{ij} \in S}$ TaskInterference($\tau_{ab}, \tau_{ik}, \tau_{ij}, w, p, \tau_{ac}$)
6:         $SB \leftarrow \{SB \cup succ(H^{seg}_{im}(\tau_{ab}))\} \setminus \{succ(\tau_{iB}) \cap H^{seg}_{im}(\tau_{ab})\}$
7:     end if
8:
9:     if $\tau_{iB} \in hp_i(\tau_{ab})$ then
10:        sectionI ← sectionI + TaskInterference($\tau_{ab}, \tau_{ik}, \tau_{iB}, w, p, \tau_{ac}$)
11:    end if
12:
13:    for $\tau_{iS} \in SB$ do
14:        [bI, bD] ← BranchInterference($\tau_{ab}, \tau_{ik}, \tau_{iS}, w, p, \tau_{ac}$)
15:        subBranchesI ← subBranchesI + bI
16:        subBDelta ← max(subBDelta, bD)
17:    end for
18:
19:    if $\tau_{iB} \in lp_i(\tau_{ab})$ then
20:        branchI ← subBranchesI
21:        branchDelta ← max(sectionI - subBranchesI, subBDelta)
22:        if $\neg IM(pred(\tau_{iB}), \tau_{iB})$ then
23:            branchDelta ← max(branchDelta, 0)
24:        end if;
25:    else
26:        branchI ← max(sectionI, subBranchesI)
27:        branchDelta ← max(subBranchesI + subBDelta - branchI, 0)
28:    end if
29:
30:    return [branchI, branchDelta]
31: end function
```

```
branchI = max(sectionI, subBranchesI)
        = max(0, subBranchesI)
        = subBranchesI

branchDelta = max(subBranchesI + subBDelta - branchI, 0)
            = max(subBranchesI + subBDelta - subBranchesI, 0)
            = max(subBDelta, 0)
```

Figure 6.5: Interference returned by τ_G: *subBranchesI* computed as *branchI* of τ_{iB} and *subBranchesI* computed as *branchDelta* of τ_{iB}

Finally at line 22, when `BranchInterference` computes *branchI* and *branchDelta*, if $\tau_{iB} \in lp_i$ we have to check if $\mathtt{pred}(\tau_{iB})$ releases immediately τ_{iB}. If not, then *branchDelta* gets the value of $\max($*branchDelta*$, 0)$ because interference cannot be negative. No changes are needed for *branchI*. When $\tau_{iB} \notin lp_i$, no changes are needed for both values. To illustrate these statements, let τ_{iB} be a non-immediate successor. Let us assume that a ghost intermediate task is between $\mathtt{pred}(\tau_{iB})$ and τ_{iB}. Let us call this task τ_G. The values of [*branchI, branchDelta*], returned by the ghost intermediate task to $\mathtt{pred}(\tau_{iB})$, are shown in Figure 6.5.

6.2.5.3 *(f3) TaskInterference*

As a reminder, the `TaskInterference` function computes the interference of a particular job $p \leqslant 0$ of τ_{ij}. This function returns the task's WCET if it can interfere. A task can interfere if it is released in $[0, w)$ and if it passes a number of reduction rules that eliminate execution conflicts due to tasks that must be in the τ_{ab} busy period, given the scenario.

A new *reduction rule* is added to the original ones in [118] due to Theorem 21. The new *reduction rule* is formally defined as:

$$\tau_{ij} \prec \tau_{ik} \;\wedge\; p < p^{seg}_{0,ikk} \;\wedge\; \neg \mathrm{IM}(\mathtt{pred}(\tau_{ik}), \tau_{ik}) \;\wedge J_{ik} = 0 \tag{108}$$

For example, in Figure 6.2, let us assume that *PR22* starts a busy period with J_{PR22} canceled. *PR22* is released at t_c so tasks *PR1* and *PR21* must have completed execution before t_c or *PR22* is not released at t_c.

The complete `TaskInterference` function is in Algorithm 6.4. The new reduction rule is at line 9.

Algorithm 6.4 Modified TaskInterference Function

```
1:  function TaskInterference($\tau_{ab}, \tau_{ik}, \tau_{ij}, w, p, \tau_{ac}$)
2:      if $p \geqslant p^{seg}_{0,ijk} \wedge w > \varphi^{seg}_{ijk} + (p-1)T_i$ then
3:          taskI ← $C_{ij}$
4:      end if
5:      if $p \geqslant p^{seg}_{0,ikk} \wedge H^{seg}_{ik} \prec \tau_{ij} \wedge H_{ik} \neq H_{ij}$
6:       or $\mathtt{pred}(H^{seg}_{ik}) \in lp_i \wedge \mathtt{pred}(H^{seg}_{ij}) \in lp_i \wedge (H^{seg}_{ik} \neq H^{seg}_{ij} \vee p \neq p^{seg}_{0,ikk})$
7:       or $\mathtt{pred}(H^{seg}_{ab}) \in lp_i \wedge \mathtt{pred}(H^{seg}_{ij}) \in lp_i$
8:       or $\mathtt{pred}(H^{seg}_{ac}) \in lp_i \wedge \mathtt{pred}(H^{seg}_{ij}) \in lp_i$
9:       or $\tau_{ij} \prec \tau_{ik} \wedge p < p^{seg}_{0,ikk} \wedge \neg \mathrm{IM}(\mathtt{pred}(\tau_{ik}), \tau_{ik}) \wedge J_{ik} = J_{clk}$ then
10:         taskI ← 0
11:     end if                                            ▷ Reduction rules 1 to 5
12:     return taskI
13: end function
```

6.2.5.4 *Modifications for Interference of* Γ_a

As a reminder, for jobs $p \leqslant 0$ of Γ_a the interference computation is similar to the Γ_i case. The only difference is the addition of a new reduction rule in the `TaskInterference` function for Γ_a.

The new reduction rule (Equation 108), presented in the previous paragraph, is added to the `TaskInterference` function for Γ_a. This is done by replacing τ_{aj} for τ_{ij} and τ_{ac} for τ_{ik} in the rule.

6.2.6 *Op3: Worst Case Response Time*

Op3 consists in computing the WCRT of τ_{ab} from the WCRTs computed for each scenario τ_{ac}.

The modifications introduced to the general algorithm, has a consequence on the contribution of jitter to a WCRT computed for a scenario. When the scenario $\tau_{ac} = \tau_{ab}$ is created, if τ_{ab} is a non-immediate successor, J_{ab} is canceled. Then J_{ab} does not contribute to the WCRT of τ_{ab}, computed for the scenario $\tau_{ac} = \tau_{ab}$. Otherwise the WCRT computed for $\tau_{ac} = \tau_{ab}$ is overestimated.

The WCDOPS+NIM test completes the scheduling analysis method for DGMF tasks. The following section shows some experiments and results.

6.3 EXPERIMENT AND EVALUATION

The WCDOPS+NIM test is implemented in Cheddar. This section presents some experiments done to evaluate the analysis results of the test.

A first experiment evaluates the WCDOPS+NIM test by simulation, while the second applies it to real case-studies from Thales. The following sections present these experiments. In each section the experimental setup is exposed, then experimental results are presented and discussed.

6.3.1 *WCDOPS+NIM Evaluation*

The WCDOPS+NIM test is compared to the original WCDOPS+ test by simulation. This experiment evaluates the pessimism of the original test when it is applied to transactions where there are non-immediate tasks. Afterwards the complexity of the WCDOPS+NIM test is discussed.

6.3.1.1 *Simulations*

In order to compare WCDOPS+NIM with WCDOPS+, the tests are applied to randomly generated system architecture models. The models are generated according to the same parameters as the WCDOPS+ simulations in [118] so both tests can be compared. The Cheddar generator is updated for the simulations.

GENERATOR UPDATE The generator produces system architecture models composed of a number of processors all with the same scheduling policy. The scheduling policies are the same because WCDOPS+ and WCDOPS+NIM are only applicable to a partitioned multiprocessor systems with a preemptive FP scheduling on all processors. A processor has a utilization factor, which can be defined by the user.

A model also has a number of transactions with a number of tasks per transaction. A transaction has a period between a minimum value and a maximum value.

Initially a task has the same priority, and is allocated on the same processor, as its direct predecessor in the transaction. The direct predecessor is chosen randomly. Both priority and processor parameters can vary. If a parameter varies, a random priority (resp. a random processor) is chosen for a task.

Table 6.1: Transaction Generator Constraints and Assumptions: GB-R are constraints; GB-H are assumptions

Ref.	Description
GB-R1	Tasks have a WCET not less than 1.
GB-R2	A processor utilization must be respected.
GB-R3	The range of a task's priority is 1 to 255.
GB-H1	Task WCET and period are positive integers.
GB-H2	Tasks have a WCET equal to their BCET.
GB-H3	Tasks on a same processor have the same WCET.
GB-H4	Parameters minimum period and maximum period are respected "as best as possible".

Tasks are immediate initially. When its offset is computed, a generated task can become non-immediate randomly. if a task becomes non-immediate, its offset is increased by a random value.

The generator's parameters, that can be defined by the user, are then:

– Number of processors
– Scheduling policy (for all processors)
– Processor utilization (same value for all processors)
– Number of transactions
– Number of tasks per transaction
– Potential minimum period of transactions
– Potential maximum period of transactions
– Probability to choose a random priority for a task
– Probability to choose a random processor to allocate a task on
– Probability to increase the offset of a task
– Maximum increase to a task's offset

Besides constraints inherited from the transaction model definition, the generator respects some other constraints. Some assumptions are also made to ease the generation. Constraints and assumptions, enforced during generation, are described in Table 6.1.

The generator produces three kinds of entities through several steps:

– Processors
– Transactions
– Tasks of transactions

The following paragraphs show the details of each step of the generation.

Generate processors This step is straightforward as the required number of processors, scheduled by the policy defined by the user parameter, are simply added to the system architecture model.

Generate transactions Again this step is straightforward as the required number of transactions are simply added to the system architecture model.

Generate tasks of transactions The algorithm to generate tasks of transactions proceeds in 5 steps:

– **Step** 1: Add tasks to each transaction, and at each addition, choose a random task in a transaction to be the predecessor of the added task.
– **Step** 2: Compute a random period for each transaction, making sure that tasks won't need to have a WCET less than 1 to respect the desired utilization of each processor.
– **Step** 3: Scale WCETs of tasks, on each processor, to achieve the utilization of each processor, while respecting the assumption that tasks on a same processor have the same WCET but not necessarily the same period.

– **Step** 4: Priorities of tasks, in each transaction, are set inversely proportional to their period, with the possibility to choose a random priority.
– **Step** 5: Offsets of a transaction's tasks are computed (since WCETs have been computed) by exploring the precedence dependency graph of the tree-shaped transaction, with a depth-search first approach.

In **Step** 1, the required number of tasks per transaction are added to each transaction Γ_i. The root task τ_{i1} of a transaction is allocated on a random processor. The next tasks τ_{ij} added to the same transaction gets a predecessor $\mathtt{pred}(\tau ij)$ as a random task that already exists in the transaction. The $\mathtt{proc}(\tau_{ij})$ is first set to $\mathtt{proc}(\mathtt{pred}(\tau ij))$. Then $\mathtt{proc}(\tau_{ij})$ can vary randomly. If it varies, a random processor is chosen for $\mathtt{proc}(\tau_{ij})$, in order to allocate τ_{ij} on random processor.

In **Step** 2, a minimum period T_{cpu} allowed for any task on a processor *cpu* is computed. This minimum period is computed in such a way that if tasks on the processor all have a WCET of 1, the required processor utilization is achieved by setting tasks with this minimum period value. As a reminder, tasks on a same processor are assumed to have the same WCET, and not less than 1.

Assume that there are N^{cpu} tasks on a processor *cpu* of a utilization denoted by *Cpu_Utilization*. If all tasks have a WCET of 1, then the minimum period allowed for any task on *cpu* is:

$$T_{cpu} = \left\lceil \frac{N^{cpu} \times 1}{\mathit{Cpu\_Utilization}} \right\rceil \tag{109}$$

Transactions are then iterated through to compute each of their period T_i. A transaction contains tasks on different processors. When a transaction is being processed, first its maximum allowed period, denoted by $T_{i,cpu}^{max}$, is computed. Value $T_{i,cpu}^{max}$ is computed by comparing the T_{cpu} of each processor, on which the transaction's tasks are allocated on. Afterwards the user defined minimum period and maximum period are set to $T_{i,cpu}^{max}$ if these are less than $T_{i,cpu}^{max}$.

After the minimum and maximum period are computed, the period of the transaction can be computed as a random value in an interval. Let *Min_Period* denote minimum period and *Max_Period* the maximum period. T_i is then computed as follows:

$$T_i = \mathrm{rand}(\mathit{Min\_Period}, \mathit{Max\_Period}) \tag{110}$$

where *rand(x, y)* returns a random integer in $[x; y]$.

While iterating through transactions to compute their periods, the maximum T_i^{max} of periods T_i is also computed:

$$T_i^{max} = \max_{\forall \Gamma_i}(T_i) \tag{111}$$

In **Step** 3, the goal is to respect the utilization factor of each processor. As a reminder, the utilization factor is $\sum_{\forall \tau_{ij}} \frac{C_{ij}}{T_i}$. Each transaction's period T_i is already computed. Therefore the goal of **Step** 3 is to compute the WCET of tasks.

As a reminder, a task is allocated on a processor *cpu*, with a required utilization, and all tasks on *cpu* have the same WCET. Let C_{cpu} denote the WCET of tasks allocated on *cpu*, which allows to achieve the required utilization. Value C_{cpu} is computed as follows:

$$C_{cpu} = \frac{\mathit{Cpu\_Utilization}}{\sum_{\Gamma_i} \frac{N_i^{cpu}}{T_i}} \tag{112}$$

where N_i^{cpu} is the number of tasks in Γ_i allocated on *cpu*. Each task allocated on *cpu* then has its WCET set to C_{cpu}.

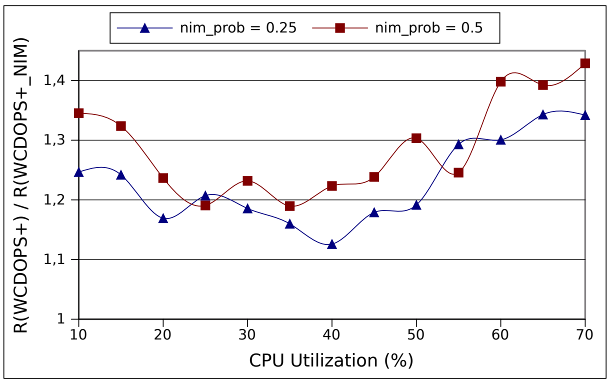


Figure 6.6: Comparison between WCDOPS+ and WCDOPS+NIM by Processor Utilization and Offset Increase Probability: *Nim_Prob* denotes the probability to increase a task's offset

In **Step** 4, priorities of tasks are set inversely proportional to their period:

$$\text{prio}(\tau_{ij}) = \frac{T_i^{max}}{T_i} \tag{113}$$

A priority cannot be higher than 255. After a priority is set, the priority can vary randomly. When a task's priority varies, it is set to a random priority between 1 and 255.

Finally in **Step** 5, offsets of tasks can be computed since their WCET are set. Offsets are set according to a depth-search first. Each time a task's offset is set, it has a probability, of a certain value, to be increased by a random value between 1 and the maximum offset increase defined by the user.

EXPERIMENTAL PARAMETERS The system architecture models are generated with the following parameters:

– 10 transactions
– 10 tasks per transaction
– 4 processors
– Preemptive FP scheduling policy
– Processor utilization between 10% and 70%
– Potential minimum period of 0
– Potential maximum period of 100000
– Probability of 0.0, 0.25, and 0.50 to choose a random priority for a task, after its default priority is set.
– Probability of 0.25 to choose a random processor for a task, after its default processor is set.
– Probability of 0.25 and 0.50 to increase a task's offset, after its default offset is set.
– Maximum increase of 1000 to a task's offset

WCDOPS+ is applied with ghost intermediate tasks added to model non-immediateness.

RESULTS Two simulations are conducted by making different parameters of the generator vary. For each set of parameters of the generator, 5 system architecture models are generated and the response times are computed for each model. For a set of generator parameters, the average ratio between WCRTs given by WCDOPS+ and WCDOPS+NIM is computed.

Figure 6.6 shows results of the first simulation where the processor utilization varies between 10% and 70%. The probability to choose a random processor and a random priority remains at 0.25. The evolution of the ratio is shown for a probability of 0.25 and 0.5 to increase offsets.

Figure 6.7 shows results of the second simulation where the processor utilization varies between 10% and 70%. The probability to choose a random processor and to increase offsets remains both

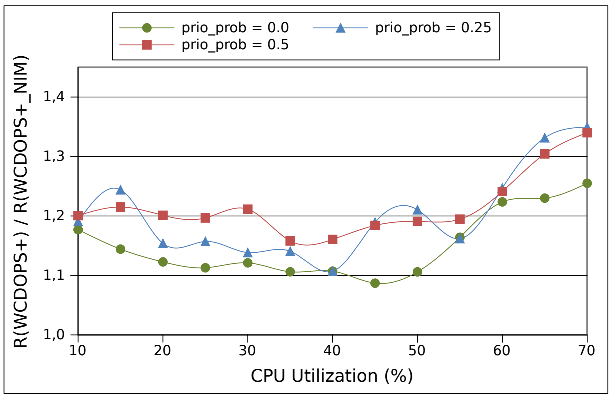


Figure 6.7: Comparison between WCDOPS+ and WCDOPS+NIM by Processor Utilization and Random Priority Probability: *Prio_Prob* denotes the probability to choose a random priority

at 0.25. The evolution of the ratio is shown for a probability of 0.0, 0.25 and 0.5 to choose a random priority for a task.

These simulation results show that results vary according to the processor utilization. The WCDOPS+NIM test gives less pessimistic WCRTs for lower and higher processor utilizations. The highest average ratio between a response-time given by WCDOPS+ and WCDOPS+NIM is 1.43. Like in [118], due to the nature of the experiment, the simulation becomes unfeasible for a high processor utilization. In the experiment the threshold is 70%.

In conclusion the WCRTs computed by WCDOPS+NIM are more than 40% less than WCRTs computed by the original test, for a processor utilization of 70%. Furthermore, the higher the processor utilization is, the less pessimistic the WCRTs given by WCDOPS+NIM are, compared to WCDOPS+.

6.3.1.2 *Complexity*

Although WCDOPS+NIM gives less pessimistic WCRTs, its time complexity needs to be discussed. The general algorithm is pseudo-polynomial [143, 105, 118]. In the WCDOPS+NIM test, two algorithms may increase the complexity of the general algorithm: jitter canceling and checking immediateness. Each of these algorithms is a loop.

Jitter canceling adds a scenario to create, if τ_{ik} is non-immediate and $\mathtt{pred}(\tau_{ik}) \notin hp_i$. Let n_τ be the number of tasks, n_Γ the number of transactions. The maximum number of extra scenarios that are created is $(n_\tau - n_\Gamma)/2$. The complexity is then $O(n_\tau - n_\Gamma)$. The complexity of jitter canceling is thus linear.

Checking immediateness is done by iterating through tasks of a transaction. It is done each time a transaction's interference is computed. Thus checking immediateness depends on the number of tasks in the system and so the algorithm is also linear.

In conclusion both jitter canceling and checking immediateness do not add a significant increase in time complexity, since they are both linear algorithms and the general algorithm is pseudo-polynomial.

6.3.2 *Experimentation on Software Radio Protocol*

As a reminder, the WCDOPS+NIM test is proposed for tree-shaped transactions that have non-immediate tasks. Such transactions may be the result of DGMF transformation. The test is thus applied by first modeling the system with DGMF. To assess the gain of applying WCDOPS+NIM on a real software radio protocol, the test is compared to current practices at Thales. In the following sections, case-studies are used to evaluate the proposed analysis method.

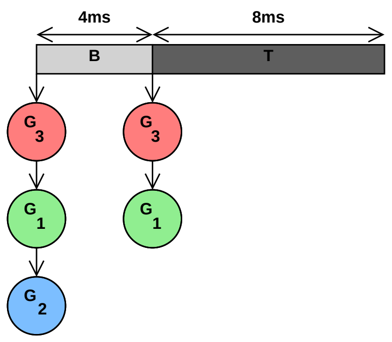


Figure 6.8: Partial Software Radio Protocol Case-Study: Circles are tasks; Arrows are precedence dependencies

Table 6.2: Case-Study 1 Task Set: Time values in ms

DGMF or GMF					
		E_i^j	D_i^j	P_i^j	$[F_p^q]_i^j$
$G_1, prio(G_1) = 1$	F_1^1	955	4000	4000	F_3^1
	F_1^2	1874	8000	8000	F_3^2
$G_2, prio(G_2) = 2$	F_2^1	5722	12000	12000	F_1^1
$G_3, prio(G_3) = 3$	F_3^1	986	4000	4000	
	F_3^2	986	8000	8000	
Periodic Model					
		C_i	d_i	T_i	priority
G_1		1874	4000	4000	1
G_2		5722	12000	12000	2
G_3		986	4000	4000	3

The first case-study is a simpler case-study that is tuned to emphasize the advantages of WCDOPS+NIM, when applied to a software radio protocol, compared to tests for other task models.

The second case-study is a complete system from Thales. Besides evaluating the advantages of the proposed analysis method, this case-study also determines its scalability.

6.3.2.1 *Case-Study 1*

DESCRIPTION OF CASE-STUDY In the first case-study, a partial software radio protocol is modeled in DGMF so the WCDOPS+NIM test can be applied. The results given by WCDOPS+NIM are compared to results given by GMF WCRT analysis in [138] and periodic task WCRT analysis in [64]. The case-study is illustrated in Figure 6.8.

In the system there are three tasks: G_1, G_2 and G_3. Task G_3 is released at the beginning of each slot. After it finishes execution, it releases G_1. G_1 releases G_2 at the first *B* slot but not at the *T* slot. G_3 and G_1, when released at a slot, must finish before the end time of the slot. G_2 when released at the *B* slot, must finish before the end time of slot *T*.

Tasks are scheduled by a preemptive FP policy and they execute on a uniprocessor. PCP protects shared resources. Tasks have parameters shown in Table 6.2. Task priorities are in highest priority first order (e.g. a priority level 3 task has a higher priority than a priority level 1 task). Execution times come from a real software radio protocol. Time units are in µs.

To compare the DGMF analysis method, through WCDOPS+NIM, with the GMF WCRT analysis and the periodic task WCRT analysis, these tasks are also modeled in the GMF and periodic task models. The parameters of each task model are also shown in Table 6.2.

Table 6.3: Case-Study 1 WCRTs: A WCRT with "/" means a missed deadline; Time values in ms

		DGMF WCRT	GMF WCRT	Periodic WCRT
G_1	F_1^1	1941	/	/
	F_1^2	6523	/	
G_2	F_2^1	8649	7694	7694
G_3	F_3^1	986	986	986
	F_3^2	986	986	

Since shared resources cannot be modeled in the GMF task model, they are not considered. If the GMF WCRT analysis is still more pessimistic without shared resources, then pessimism will not be improved with shared resources.

As for the periodic model, the DGMF tasks are modeled as periodic. Their longest frame execution time is taken as their WCET. Their smallest min-separation time between two frames is taken as their period.

ANALYSIS EVALUATION Computed WCRTs are shown in Table 6.3. The computed WCRTs show that no deadlines are missed by DGMF analysis, through the WCDOPS+NIM test. This is not the case for the two other task models.

For G_2, GMF and periodic WCRT analysis gives a lower WCRT than the WCDOPS+NIM test but the values, computed for the GMF and periodic task models, are underestimated. Indeed GMF WCRT analysis considers that F_2^1 is only interfered by F_3^1 and F_3^2, without considering the fact that F_2^1 is released after F_1^1. Similarly, the periodic task model does not consider the precedence dependency between $G_1 \prec G_2$.

In conclusion DGMF analysis, through WCDOPS+NIM, determines a schedulable system. Precedence dependencies are also considered by the analysis, in order to not underestimate WCRTs.

6.3.2.2 *Case-Study 2*

DESCRIPTION OF CASE-STUDY The second case-study is the real software radio protocol that was presented in Section 5.5.3.

As a reminder, the case-study is implemented with 8 POSIX threads on a processor called *GPP1*, and 4 threads on a processor called *GPP2*. Both processors are scheduled by the SCHED_FIFO scheduler of Linux (preemptive FP policy). The threads have precedence dependencies and they are released at the start of different slots of a TDMA frame of 14 slots.

The case-study, modeled with DGMF, is transformed to a tree-shaped transaction of 44 tasks with non-immediate tasks, that is shown in Figure 5.14 of Section 5.5.3. The task parameters are in Table B.1 of Appendix B.

ANALYSIS EVALUATION Currently the analysis approach used at Thales is similar to the approach of the Joseph & Pandya test in [64] (abbreviated as the "J&P test" in the following paragraphs). To assess the advantage of applying WCDOPS+NIM to the real software radio protocol, WCRTs given by WCDOPS+NIM are compared with those computed by the J&P test. This test was used in Chapter 4 for the initial experiment presented in this thesis. As a reminder, it was shown that the test gives pessimistic WCRTs.

The J&P test cannot be applied directly to the case-study for the following reasons:

– Precedence dependency between tasks that may be on different processors
– Constraint of deadline less than or equal to period, assumed by the J&P test

Table 6.4: Task WCRTs of MAC Layer Case-Study: R_{RM} is WCRT given by [64]; $R_{WCDOPS+NIM}$ is WCRT (global WCRT minus offset) given by WCDOPS+NIM; : Time values in ms

Task	R_{RM}	$R_{WCDOPS+NIM}$
FC_S1_3	33405	573
AB_B1_3	6505	1241
AB_B2_3	6505	1241
AB_B3_3	6505	1241
AB_B4_3	6505	741
AD_T2_1	3955	651
AD_T4_1	3955	651
AD_T6_1	3955	651
AD_T8_1	3955	651
BT_T1_1	1915	463
BT_T3_1	1915	463
BT_T5_1	1915	463
BT_T7_1	1915	463
BB_B5_3	16859	3313
BS_S1_3	16959	3634

For the problem of precedence dependency between tasks on different processors, in the case-study, the priority assignment in [5] is first used when analyzing a particular task: a task has a priority lower than its predecessor's priority. All tasks are then allocated to a same processor and a synchronous system is assumed for the J&P test.

The J&P test assumes a task deadline less or equal to its period. Therefore the test only computes the WCRT of the first job of a task. The constraint on deadlines is not respected by the case-study tasks, so the response time of the first job is not sufficient to determine schedulability. On the other hand, as we will see with the results in the following paragraph, even the response time of the first job is overestimated by the J&P test, compared to a WCRT computed by WCDOPS+NIM.

Only the WCRTs of tasks, without any successor, are compared. These tasks represent the completion of some service, thus their response time is of interest to determine if the service misses some deadline. For example in the service represented the precedence dependency chain $RM\_S1 \prec BB\_S1\_1 \prec BB\_S1\_2 \prec BB\_S1\_3$, only the WCRT of *BB_S1_3* is of interest to determine if the service misses its deadline, i.e. if the deadline of *BB_S1_3* is missed.

When applying WCDOPS+NIM on the case-study, the convergence of response times takes 7 seconds on a Intel Core i5 @ 2.40 GHz. The computed WCRTs are shown in Table 6.4.

For each analyzed task, the WCRT computed by the J&P test is denoted by R_{RM}, and the WCRT computed by WCDOPS+NIM is denoted $R_{WCDOPS+NIM}$. From results in Table 6.4, a ratio of response times, given by both tests, is computed for each task. This ratio is denoted $^{R_{RM}}/_{R_{WCDOPS+NIM}}$.

In average this ratio is 8.89 so in average the J&P test gives a WCRT almost 9 times higher than WCDOPS+NIM. This result shows that considering TDMA task releases reduces the pessimism of WCRTs.

The WCDOPS+ test was also applied to the case-study by modeling non-immediateness with ghost intermediate tasks. The ratio of WCRTs computed by WCDOPS+, compared to WCRTs computed by WCDOPS+NIM, is 1.08. The pessimism of WCRTs computed by WCDOPS+ is thus negligible.

This result is explained by the fact that the processor utilization of the case-study is low since the duration of slots is high compared to the WCETs of tasks. As shown by the simulations in Section 6.3.1.1, for low a processor utilization, WCDOPS+ does not give significantly more pessimistic WCRTs than WCDOPS+NIM.

The slot durations are high because the initial TDMA configuration is not necessarily optimized. Furthermore some processor time must be dedicated to execution of other applications than the software radio protocol. Later specifications may decrease the slot durations, which means an increase in processor utilization.

6.4 CONCLUSION

This chapter proposed a schedulability test called WCDOPS+NIM, an extension of the WCDOPS+ test. The new test is applicable to tree-shaped transactions with non-immediate tasks.

Simulation results showed that WCRTs computed by WCDOPS+NIM are more than 40% less than WCRTs computed by the original test, for a processor utilization of 70%. As a matter of fact, the higher the processor utilization is, the less pessimistic the WCRTs given by WCDOPS+NIM are, compared to WCDOPS+.

Since the WCDOPS+NIM completes the DGMF analysis method, it was applied to two real case-studies from Thales. This experiment showed not only that the original GMF task model computes underestimated response times, but also overestimated ones. The experiment also showed that WCRTs computed by DGMF, through WCDOPS+NIM, are more than 9 times less than those computed by the periodic task model, through the test in [64].

In conclusion, the WCDOPS+NIM test gives less pessimistic results, through lower WCRTs, than the WCDOPS+ test in [118], the periodic task model test in [64], and the GMF task model test in [138]. For engineers, the reduced pessimism may offer more design choices. System resources (e.g. processors) may also be less over-dimensioned, which may lead to a reduction in costs. Finally, the cost to redesign systems, considered by previous tests as unschedulable, is also lessened.

Models with DGMF tasks or transactions must be produced from an architecture model of the system, for automatic scheduling analysis. In the next chapter, the architecture model in Chapter 4 is extended so it can be transformed and exploited by the DGMF scheduling analysis method.

PUBLICATIONS The WCDOPS+NIM schedulability test is published in [83]. The implementation work of WCDOPS+NIM is published in [80].

Chapter 7

AN ARCHITECTURE MODEL FOR AUTOMATIC SCHEDULING ANALYSIS

In Chapter 4 an architecture model of a software radio protocol was proposed in UML MARTE, then transformed to a Cheddar-ADL model, so automatic scheduling analysis could be performed with the periodic task model. This chapter shows a similar experiment for DGMF.

The previously proposed architecture model only contains UML structural entities (see Section 1.7.1.1). Behavioral entities of UML are not present in the model so it does not contain enough information to exploit a task model like DGMF. Indeed, one of the purpose of DGMF is to model tasks with different parameters at each job, due to different behaviors.

This chapter shows an example of modeling Thales specification documents in UML MARTE, in order to transform the specification to a set of DGMF tasks, and then a set of transactions.

This chapter thus presents an extension of the original UML MARTE model. Some entities of the proposed UML model are described with behavioral diagrams of UML. The UML MARTE model can be transformed to a Cheddar-ADL model, where the DGMF and transaction models are implemented. This way the tool can perform the scheduling analysis automatically.

The following sections first present a Thales specification document. Then the extended UML MARTE model is exposed. Afterwards its transformation to a Cheddar-ADL model is shown. Finally some experiments are done to evaluate the approach.

7.1 EXPLOITING SPECIFICATION DOCUMENTS

Since the original UML MARTE model in Chapter 4 was proposed only for the periodic task model, it does not necessarily contain enough information for another task model such as DGMF or transaction. Some information that the original model lacks are:

– Precedence dependency: In the original model, there is no entity to represent a precedence dependency.
– Individual job parameter: In the original model, a task is modeled with a single WCET, period, and deadline.

The UML MARTE model thus needs to be extended. To be integrated into the development cycle at Thales, the model uses some information that are available in some Thales specification documents. These information are not described in MyCCM development models for code generation.

The specification documents describe services. A service is defined in Definition 37, i.e. a set of functionalities. In some Thales specification documents, the functions of a service are represented. In the documents, a function is a sequence of instructions executed by a task, in order to fulfill the functionalities of the service. A task, called a thread in the specification documents, executes on a processor. A function has a WCET, may have shared resource critical sections, and may have precedence dependencies between them.

A service is released by some events. In the context of the Thales specification documents, an event is the occurrence of some incoming data or message, or some interrupt and signaling mecha-

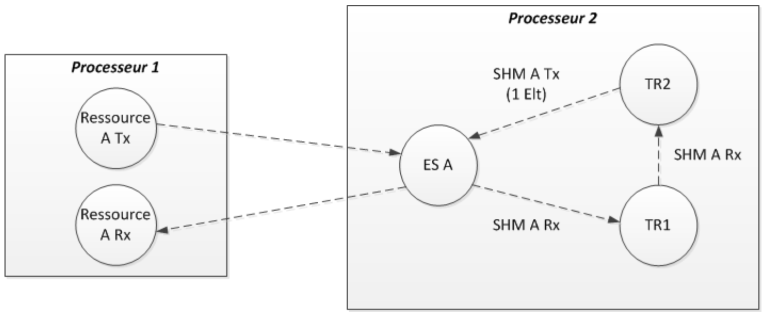


Figure 7.1: Thales Specification Document Figure: Rectangles are processors; Circles are functions; Arrows are precedence dependencies between functions

nism of the OS. Sometimes these events may be associated with some information, indicating the nature of the event. For example a TDMA tick is an event, and the tick may be associated with the type the slot it starts. Depending on the nature of the event, some functions are executed, and others are not.

Figure 7.1 shows a figure in a Thales specification document[1] of a real system. The figure of the specification document illustrates a service. Figure 7.1 thus shows a service implemented on some processors, e.g. a processor called *processeur* 1 and a processor called *processeur* 2. The service has some functions to execute. Each function has a WCET and may use shared resources. For example function *TR2* uses shared resource *SHM_A_Tx*. The functions are also related by precedence dependency.

The extended UML MARTE model is based on information illustrated in Figure 7.1. This kind of figure typically contains information on precedence dependency. It is also possible to define several sequences of execution, and thus different parameters of jobs.

On the other hand, the figure alone is ambiguous, i.e. it is complemented by textual descriptions in the specification document. For example Figure 7.1 is ambiguous because there are two possible next functions after function *ES A*, i.e. either *TR1* or *Ressource A Rx*. To enable automatic scheduling analysis with DGMF, the ambiguities must be leveraged.

The next section proposes to represent the information of Thales specification documents, described in this section, in a UML MARTE model.

7.2 NEW MODEL FOR SCHEDULING ANALYSIS: SERVICE MODEL

The model that exploits the behavioral information, contained in specification documents, is called the Service Model. It extends the model of Chapter 4. The Service Model is described with some views (Definition 40).

In the following sections, some entities of the model are first briefly introduced. Then three representative views of the model are exposed. In the description of each view, the meta-model of its entities is presented, their mapping to UML and MARTE is shown, and an example is given. The full description of each view, diagram palette, entity, attribute, and mapping to UML MARTE concepts, is in Appendix C.

7.2.1 *Service Model Overview*

Figure 7.2 shows some entities of the Service Model, and how they are related to each other. The follwing paragraphs describe these entities and their relationships.

1. The document is in French.

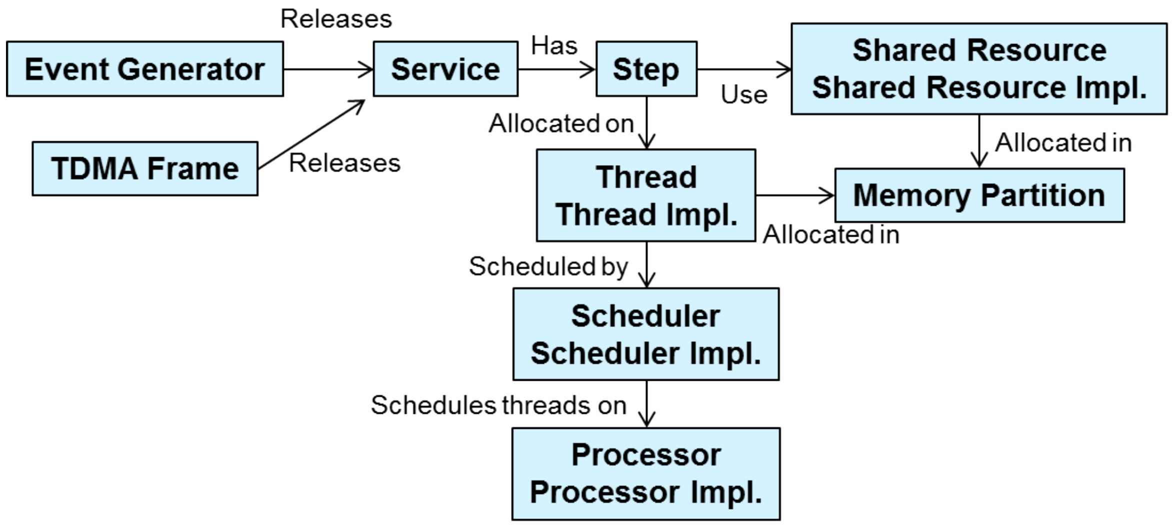


Figure 7.2: Service Model Overview: Rectangles are entities; Arrows represent relationships; "Impl." is abbreviation for "Implementation"

In the Service Model, some software and execution platform entities, of the original model in Chapter 4, are re-used: *threads*, *shared resources*, *memory partitions*, *schedulers*, and *processors*. Some software and execution platform entities are extended with the introduction of the concept of *implementation*. This concept is related to threads, shared resources, schedulers, and processors.

An implementation of an entity is is an instance of that entity. For example, consider an input-output thread. It can be implemented several times, with different parameters. There are then several instances of the input-output thread. An implementation of a thread is called a thread implementation. There are also implementations of shared resource, scheduler and processor in the model. Implementations of thread and shared resource are allocated in memory partitions.

The new entities of the Service Model are *services*, *event generators* and *TDMA frames*. The idea is to model services and how they are released, through the modeling of event generators and TDMA frames. Event generators produce events that release the services.

An event generator is an entity of the model that represents any source of events in the system, for example some interrupt mechanism of the OS. The events are generated following a pattern defined by the event generator. For example an event generator can generate periodic events to release a service.

A service has several functions that are represented by entities called *steps* in the Service Model. A step thus have the parameters and dependencies of a function, as described in Section 7.1. Each release of a step is called an instance of the step.

The steps of a service may use some shared resource implementations. The steps of a service are allocated on thread implementations. A thread implementation is scheduled by a scheduler implementation. The scheduler implementation schedules threads on a processor implementation.

The entities of the Service Model are described in several views of the model. These views are of two types: structural and behavioral views of UML. In the next sections, one behavioral view and two structural views are presented.

7.2.2 *Service Behavior View*

The service behavior view is a UML activity diagram. Steps of a service are modeled in this view. The following sections first present the entities of the service behavior view, then their mapping to UML MARTE concepts, and finally an example.

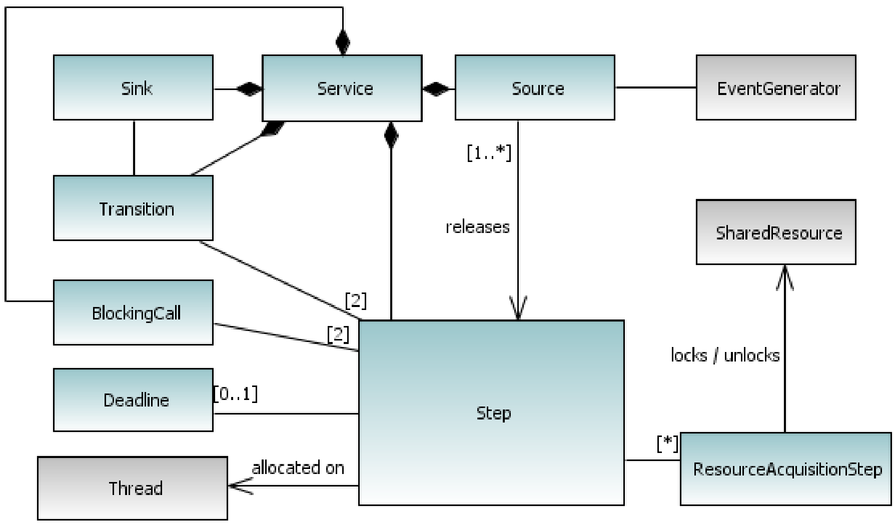


Figure 7.3: Service Behavior View Meta-Model

7.2.2.1 *Entities*

Figure 7.3 shows the meta-model of entities in relation to the service behavior view. A *service* is released at one or several entities called *sources*. These *sources* are released by *event generators*. Once released, a *service* has several *steps* to execute.

Steps are related by precedence dependency through *transitions*. A *transition* is a directed relationship that indicates that one *step* is released after the completion of the preceding one. The release of a *step* also depends on some conditions of its incoming *transition*. For example, to determine if the *step* is released, a *transition* may test some value of some variable. A *blocking call* between two *steps S1* and *S2* means that instance k of *S2* must have finished before instance $k+1$ of *S1* can be released.

Steps may use *shared resources* through *resource acquisition steps* that represent critical sections. A *step* may also have a *deadline*.

7.2.2.2 *Mapping to UML MARTE*

The Time, GQAM, and SAM sub-profiles of MARTE are used in the service behavior view. The Time sub-profile is used to represent time constraints. The GQAM sub-profile is dedicated to generic quantitative analysis models, while the SAM sub-profile is dedicated to scheduling analysis models.

Table 7.1 shows how each entity is mapped to a concept in UML MARTE. The UML meta-classes are described in Section 1.7.1.1.

7.2.2.3 *Example*

Figure 7.4 shows an example of the service behavior view, where a service is modeled. The service is part of the MAC layer of a software radio protocol. Once the service is released, the service may end with step *ACK_A_Tx* or step *ACK_B_Tx*. The *ACK_A_Tx* step is the service in MAC transmitting data to the RLC layer, while *ACK_B_Tx* is the service transmitting data to the PHY layer.

Step *PR1* has a resource acquisition step (critical section), where it uses the *SHM_B_Rx* shared resource. The *IB_ACK_Deadline* is an example of a deadline of the *IB_ACK* step.

A service is released by events generated by entities described in the event generator view. The next section presents this view.

Table 7.1: Service Behavior Mapping

Entity	MARTE Model
Service	**Activity** stereotyped <<GaWorkloadBehavior>>
Source	**InitialNode** stereotyped <<GaWorkloadEvent>>
Step	**Action** stereotyped <<SaStep>>
Deadline	**Constraint** stereotyped <<TimedConstraint>>
Resource Acquisition Step	**Constraint** stereotyped <<GaAcqStep>>
Transition	**ControlFlow** with no stereotype
Blocking Call	**ControlFlow** stereotyped <<SaCommStep>>
Sink	**ActivityFinal**

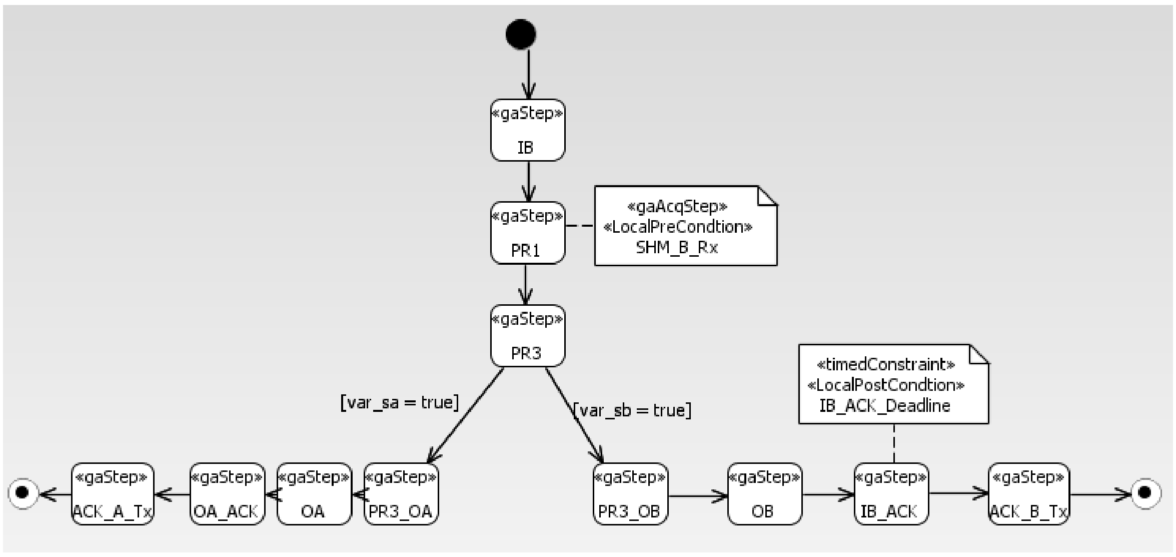


Figure 7.4: Service Behavior View Example: White boxes are steps; Deadlines and resource acquisition steps are represented by UML comments; Arrows are transitions; Filled black circle is source; Circle with dot is sink

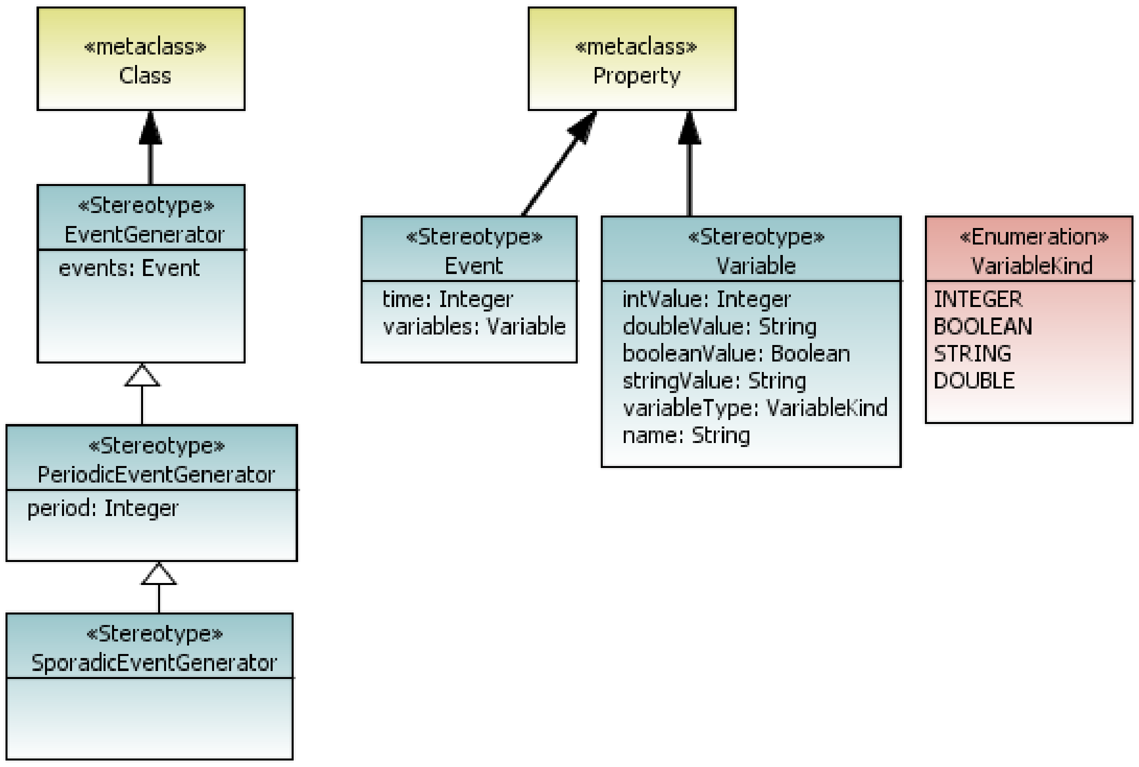


Figure 7.5: Event Generator Profile: Black filled arrows are UML extensions

7.2.3 *Event Generator View*

The event generator view is a UML class diagram. This view is used to model event generators, TDMA frames, and the events they generate. An event may also be associated with some variables having values. These variables represent the information with which the event is associated with. The variable values can be tested by the transitions between steps of a service.

The following sections first present the entities of the event generator view, then their mapping to UML MARTE concepts, and finally an example.

7.2.3.1 *Entities*

Figure 7.5 shows the event generator UML profile proposed to model event generators. The UML profile proposed to model a TDMA frame is shown in Figure 7.6. In the event generator view, the user may use either profile to model an entity that releases a service.

For example, a *periodic event generator* will generate *events* periodically. A *TDMA frame* is also an event generator that will generate *events*, representing TDMA ticks indicating the start of *TDMA slots*. A *sporadic event generator* can be described with the same *period* parameter as the *periodic event generator*.

Events may have *variables*. For example a *periodic event generator* can generate two *events* periodically with different *variables*. The *variables* may have different values according to their *variable type*. A *variable* can be an *integer*, a *boolean*, a *string*, or a *double*.

In the case of a *TDMA frame*, an event is a *TDMA slot*. A *TDMA slot* has a *TDMA slot type*, a *TDMA slot mode*, and a *duration*.

By default all *event generators* generate one *event* without any *variable*. This *event* is generated for the first time at 0 and then according the pattern defined by the *event generator*.

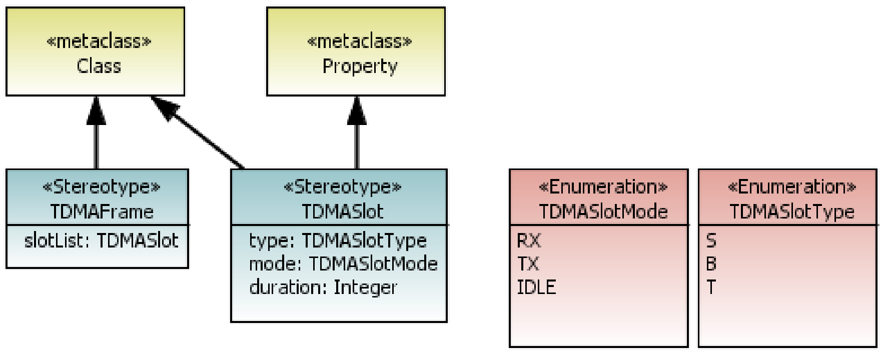


Figure 7.6: TDMA Profile: Black filled arrows are UML extensions

Table 7.2: Event Generator Mapping

Entity	MARTE Model
Event	**Property** stereotyped <<Event>>
Variable	**Property** stereotyped <<Variable>>
Event Generator	**Component** stereotyped <<ClockResource>> from MARTE and <<EventGenerator>>
Periodic Event Generator	**Component** stereotyped <<ClockResource>> from MARTE and <<PeriodicEventGenerator>>
Sporadic Event Generator	**Component** stereotyped <<ClockResource>> from MARTE and <<SporadicEventGenerator>>
TDMA Frame	**Component** stereotyped <<ClockResource>> from MARTE and <<TDMAFrame>>
TDMA Slot	**Property** stereotyped <<TDMASlot>>

In case the user models several *events* for an *event generator*, then the first release time of these *events* may also be modeled. For example a *periodic event generator* may generate two *events*: one *event* for the first time at 0, then another *event* for the first time at 5. If the *period* is 10, then the next two *events* occur at 10 and 15 respectively.

7.2.3.2 *Mapping to UML MARTE*

Table 7.2 shows how each entity is mapped to a concept in UML MARTE and one of the two profiles presented in the previous section.

7.2.3.3 *Example*

Figure 7.7 shows an example of the event generator view. The entity called *IO_A_TDMAFrame* is a TDMA frame that contains several slots. For example *BSlot1* is a slot.

The entity called *IO_B_Generator* is a periodic event generator, with two events: *ev_tr3_sa* and *ev_tr3_sb*. The events have some variables called *var_sa* and *var_sb*.

Events release services with steps that are allocated on thread implementations. Threads are described in the threads implementation view. The next section presents this view.

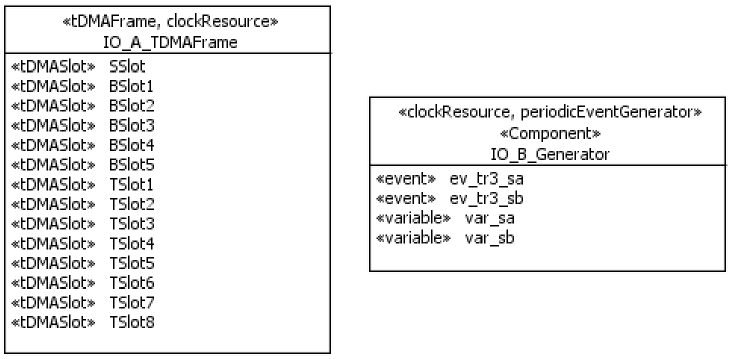


Figure 7.7: Event Generator View Example

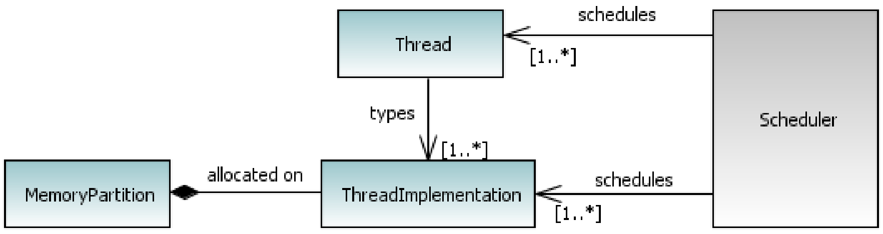


Figure 7.8: Thread Implementation View Meta-Model

7.2.4 *Thread Implementation View*

Steps are allocated on thread implementations. The thread implementation view offers the possibility to model threads and their implementations. The view is a UML composite structure diagram.

The following sections first present the entities of the view, then their mapping to UML MARTE concepts, and finally an example.

7.2.4.1 *Entities*

Figure 7.8 shows the meta-model of entities in relation to the thread implementation view. A *thread implementation* either has its own parameters, or those of the *thread* that types it. A *thread implementation* is allocated in a *memory partition*. Threads and their implementations are scheduled by a *scheduler implementation*.

7.2.4.2 *Mapping to UML MARTE*

The SRM sub-profile of MARTE is used to model entities of the shared resource implementation view. The SRM sub-profile is dedicated to design models of the software. Table 7.3 shows how each entity and is mapped to a concept in UML MARTE.

Table 7.3: Thread Implementation Mapping

Entity	**MARTE Model**
Memory Partition	**Component** stereotyped <<MemoryPartition>>
Thread	**Component** stereotyped <<SwSchedulableResource>>
Thread Implementation	**Property**, stereotyped <<SwSchedulableResource>>, of a partition, and typed by a thread

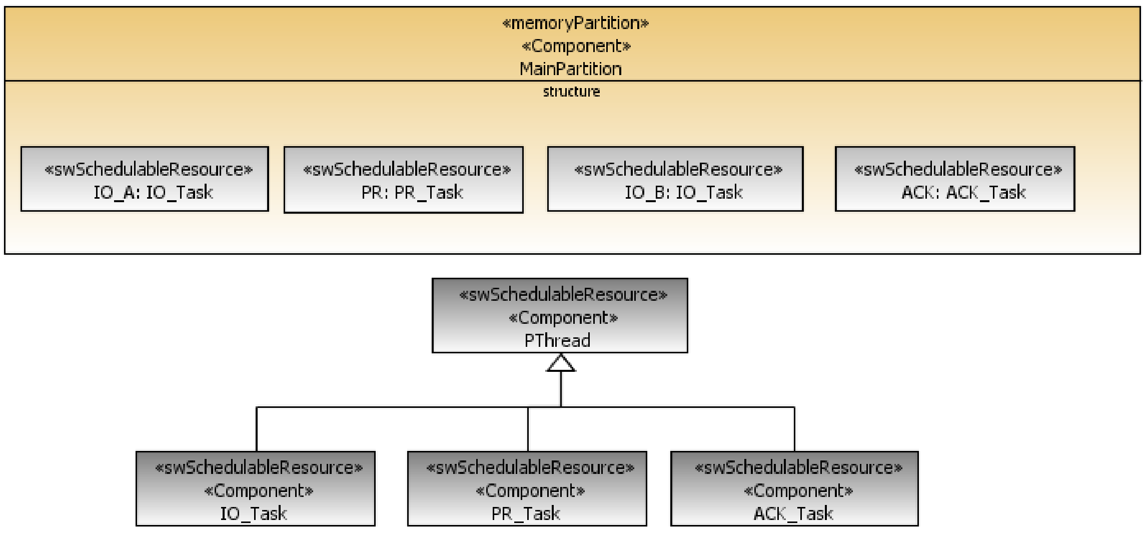


Figure 7.9: Thread Implementation View Example

7.2.4.3 *Example*

Figure 7.9 shows an example of the thread implementation view. Threads called *IO_Task*, *PR_Task*, and *ACK_Task* are POSIX threads, modeled by the entity called *PThread*. The implementation of these threads are allocated in the memory partition called *MainPartition*. The thread implementations are *IO_A*, *IO_B*, *PR*, and *ACK*.

The next section shows how the Service Model is transformed into a Cheddar-ADL model.

7.3 SERVICE TO CHEDDAR

In the previous section the Service Model was described. This section shows how the Service Model is transformed into a Cheddar-ADL model for automatic scheduling analysis with the DGMF analysis method [2].

The transformation is called Service to Cheddar (S2C). The transformation takes as input a Service Model, then transforms it to a DGMF model. Afterwards the S2C transformation uses the DGMF to transaction transformation to produce the final transaction model for schedulability analysis. For readability, this section only presents the final result of the S2C transformation, which is a set of transactions in Cheddar-ADL.

The following section first gives the overview of the transformation. Then the transformation algorithm is presented. The complete explanation of the S2C transformation algorithm is given in Appendix D.

7.3.1 *Overview of the Transformation*

The transformation is summed up in Figure 7.10. The figure represents a service of two steps, *step 1* and *step 2*. The service is released by an *event generator* that produces 2 *events*. After the transformation, we get a transaction of 4 tasks: *task 11*, *task 12*, *task 21*, and *task 22*.

Each instance of a step is transformed to a Cheddar-ADL task entity. The number of instances of a step depend on the events that release the service. For example, two instances of *Step 1* in Figure 7.10 are transformed to two Cheddar-ADL tasks *Task 11* and *Task 12*.

2. Demo video available at `beru.univ-brest.fr/svn/CHEDDAR/trunk/contribs/examples_of_use/s2c/demo_s2c.zip`

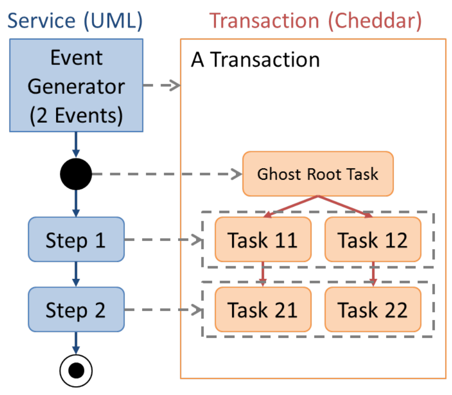


Figure 7.10: Service to Cheddar Overview: Dashed arrows show transformation

The allocation of a step on a thread implementation determines some parameters of the Cheddar-ADL task. The allocation of the thread implementation in a memory partition, and the scheduler that schedules the thread implementation on a processor, also determines some parameters of the Cheddar-ADL task.

Cheddar-ADL tasks that represent step instances released by events of a same event generator, will belong to a same transaction at the end of the transformation. Otherwise said, all Cheddar-ADL tasks resulting from the same event generator are in the same transaction. Indeed, an event generator's events are related in time, thus the tasks are related in time and therefore they are in the same transaction [143].

Tasks in a transaction are preceded by a ghost root task (Definition 55) modeled by the schedulability test for transactions [118]. For example in Figure 7.10, the *ghost root task* represents the first *event* produced by the *event generator*.

The next section shows how the overview of the transformation, presented in this section, is implemented by the transformation algorithm.

7.3.2 *Algorithm Overview*

The transformation algorithm takes as input a Service Model. It transforms all services of the model. A service is transformed by exploring it with a depth-first search approach. The exploration starts at each source of the service. When handling a source, all events releasing the source are considered.

A crucial part of the transformation is to transform a step. Like stated in Section 7.3.1, a step instance results in a Cheddar-ADL task. The parameters of the Cheddar-ADL task are set by transforming the thread implementation on which the step is allocated on.

When transforming a thread implementation, the memory partition that contains it, the scheduler that schedules it, and the processor that hosts it, are also transformed to equivalent entities in the Cheddar-ADL model.

Finally when transforming a step with resource acquisitions, all resource acquisitions result in critical sections of Cheddar-ADL. When a resource acquisition is transformed, the shared resource associated with the resource acquisition is also transformed.

Appendix D presents the detailed algorithms of the transformations described in this section.

The next section exposes some experiments to evaluate the S2C transformation.

7.4 EXPERIMENT AND EVALUATION

The S2C transformation is implemented as an Eclipse plug-in to be applied on a UML model created with Papyrus. The performance of the transformation is evaluated by simulation. The evaluation determines the scalability of the modeling and transformation approach.

The metrics of the evaluation are the time performance, which is measured through the transformation time, and the memory usage, which is measured through the number of Cheddar-ADL tasks produced by the transformation. Each Cheddar-ADL task takes some memory space. Therefore the more there are Cheddar-ADL tasks, the higher the memory usage is.

In the following sections, the case-study and simulation setups are first exposed. In particular, the choice of the simulation parameters is justified. Afterwards the results of the simulation are shown and discussed.

7.4.1 *Simulation Setup*

The simulation consists in transforming a model with the service shown in Figure 7.4, presented in Section 7.2.2.3. The service is part of the MAC layer of a software radio protocol, so it is released by TDMA ticks.

Since it is released by TDMA ticks, the experiment consists in releasing the service by simulating a TDMA frame. This is done with a random event generator, which is an event generator only used for the experiment.

The random event generator produces events between 0 and *end*, which is a parameter equal to the chosen duration of the TDMA frame. Otherwise said, the time interval $[0; end]$ is filled with slots and the interval corresponds to a TDMA frame.

A slot duration is chosen by a parameter called *maxInterOccr*. A slot duration is a random value in the interval $[0; maxInterOccr]$. Therefore slots of different durations are also simulated.

Below, two simulation setups are presented. The first simulation aims at evaluating some average values of transformation time and memory usage. The second simulation evaluates the growth of the transformation time and memory usage. The simulations run on an Intel Core2Duo processor @ 2.53 Ghz.

7.4.1.1 *Simulation to Evaluate Average Values*

With the *end* parameter, TDMA frames of duration 6000 ms, 33000 ms, and 60000 ms are simulated. The 6000 ms duration is typical of software radios developed at Thales, while the other two durations are meant to study the scalability of the transformation.

For each TDMA frame duration, events are generated with a *maxInterOccr* equal to *end* divided by 10, 50, and 100. For example $maxInterOccr = end/10$ means 10 slots in average. These values of *maxInterOccr* are chosen because they represent a realistic number of slots in a TDMA frame specified at Thales.

For each combination of parameters *end* and *maxInterOccr*, 10 event generators are created. Each event generator releases the service in Figure 7.4. The service is thus transformed 10 times.

After a transformation, the number of produced Cheddar-ADL tasks, and the transformation time are computed. For the 10 transformations, the average number of Cheddar-ADL tasks and the average transformation time are then computed.

7.4.1.2 *Simulation to Evaluate Growths*

In the second simulation, the duration of the TDMA frame is kept at 60000 ms. Events are generated with a *maxInterOccr* equal to *end* divided by 10 to 100, by increments of 10.

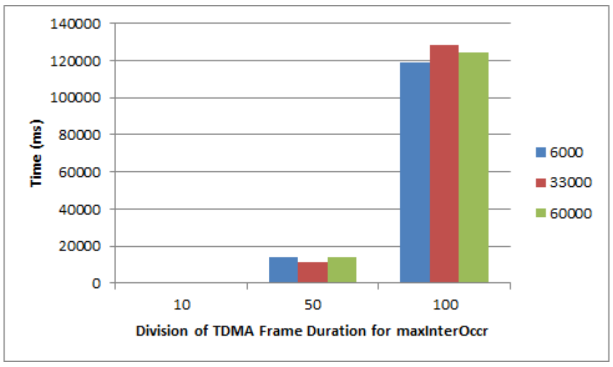


Figure 7.11: Transformation Time for TDMA Frame Durations of 6000 ms, 33000 ms and 60000 ms

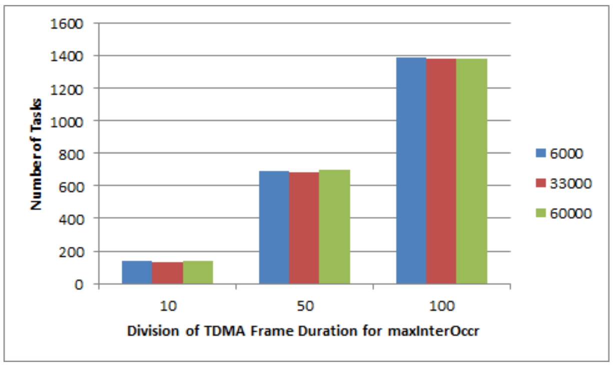


Figure 7.12: Number of Cheddar-ADL Tasks for TDMA Frame Durations of 6000 ms, 33000 ms and 60000 ms

For each *maxInterOccr*, 10 random event generators are created. The average number of Cheddar-ADL tasks and the average transformation time are then computed like in the first simulation in Section 7.4.1.1.

7.4.2 *Results*

The following sections present the results of both simulations, and a discussion on the results.

7.4.2.1 *Results for Average Values*

Figure 7.11 (resp. 7.12) shows the transformation time for three values of the *maxInterOccr* parameter, expressed as a division of the TDMA frame duration.

The difference between the longest transformation time and the shortest is about 128 s. The difference between the greatest number of Cheddar-ADL tasks and the smallest is 1265.

7.4.2.2 *Results for Growths*

Figure 7.13 (resp. 7.14) shows the growth of the transformation time (resp. number of Cheddar-ADL tasks) by the *maxInterOccr* parameter, expressed as a division of the TDMA frame duration.

Figure 7.13 shows that the transformation time is exponential. Figure 7.14 shows that the growth of the number of Cheddar-ADL tasks is linear.

7.4.2.3 *Evaluation of the Results*

The results of both simulations show that the transformation time and memory usage depend on the number of events that release the service.

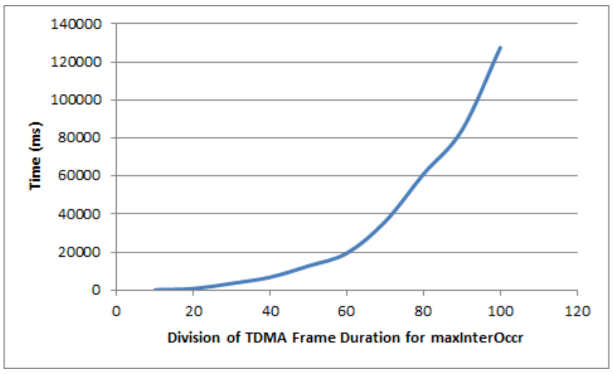


Figure 7.13: Transformation Time Growth

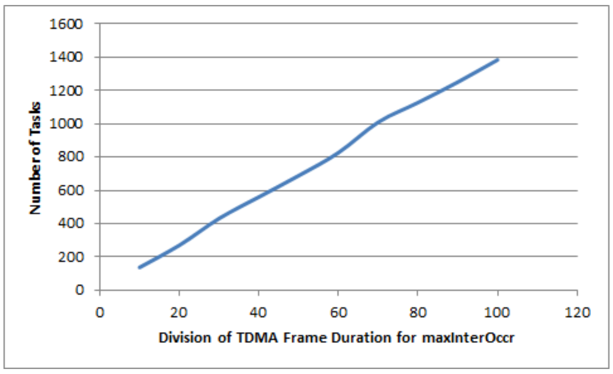


Figure 7.14: Number of Tasks Growth

transformation time The growth of the transformation time seems exponential. The exponential growth is due to a procedure called *reduceJobPrecs*, shown in Algorithm D.2 in Appendix D. The procedure is a graph reduction algorithm [68], i.e. it reduces edges in a graph. Without going into the details of the procedure, let us see how it impacts the transformation time.

During the transformation, precedence dependencies are specified between Cheddar-ADL tasks that represent different instances of a same step. These precedence dependencies are called *job precedence dependencies*. There are some redundant precedence dependencies when there are more than two instances of a same step. As a reminder, a precedence dependency is said redundant, if it is already expressed by some other precedence dependency, due to the transitivity of precedence dependency. For example a precedence dependency $S11 \prec S13$ is unnecessary if we have $S11 \prec S12 \prec S13$. The *reduceJobPrecs* procedure removes these unnecessary dependencies.

The procedure has two nested loops that iterate through the number of job precedence dependencies. Let n_{jp} denote the number of job precedence dependencies. The complexity of Algorithm D.2 is thus $O(n_{jp}^2)$.

Parameter n_{jp} is the number of job precedence dependencies so it can be expressed as $n_{jp} = n_s^{n_e}$, with n_s the maximum number of steps explored at each release of the service, and n_e the number of release events. The complexity of the transformation algorithm is then $O(n_s^{n_e^2})$.

Parameter n_s is fixed because it does not evolve according to the number of events. Thus the complexity is exponential. For example in Figure 7.4, we have $n_s = 7$ and the complexity is $O(7^{n_e^2})$.

For a high number of events, the transformation faces an issue of scalability. For example, for a high number of events like about 100, the transformation time is 128 seconds. On the other hand, the TDMA frame of the real case-study, analyzed in Section 6.3.2, has 14 slots and thus 14 events. The time taken to transform 14 events is between 173 ms (10 events in Figure 7.13) and 857 ms (20 events in 7.13).

MEMORY USAGE The growth of the number of Cheddar-ADL tasks, that are produced, is linear. This result is consistent, considering that when a service is released, it is explored like a tree. When a service is explored, its steps result in Cheddar-ADL task creations. Thus the more there are events that release the service, the more there are Cheddar-ADL tasks.

For a high number of events, the transformation faces again an issue of scalability. For example, for a high number of events like about 100, the number of produced Cheddar-ADL tasks is 1394. On the other hand, the TDMA frame of the real case-study, analyzed previously, has 14 slots and thus 14 events. The number of produced Cheddar-ADL tasks, to transform 14 events, is between 137 (10 events in Figure 7.14) and 270 (20 events in Figure 7.14).

CONCLUSION ON EVALUATION OF RESULTS Besides services released by TDMA ticks, the other kinds of services in a software radio protocol are released by a sporadic or periodic event. Their transformation does not depend on the number of release events.

In conclusion, it takes less than 1 second to automatically perform scheduling analysis of a software radio protocol, with the proposed UML MARTE model and S2C transformation.

7.5 CONCLUSION

In this chapter, an experiment was done to assess the possibility of automatic scheduling analysis of a software radio protocol with the DGMF task model and its analysis method. To enable automatic scheduling analysis, an extension of the UML MARTE model in Chapter 4 was proposed so the DGMF analysis method can be used.

The proposed UML MARTE model, called Service Model, is based on specification documents from Thales. The Service Model re-uses some entities of the original UML MARTE model, and extends it with new entities. These new entities are services, which are are some functions, released by some events, generated by some event generators.

The Service Model is transformed to a Cheddar-ADL model by a transformation called S2C. One of the metric to evaluate the transformation, is its memory usage. Experimental results showed that the memory usage evolves linearly. Furthermore, results also showed that when a real software radio protocol system is modeled, the number of Cheddar-ADL tasks produced by the transformation is adapted to the Cheddar tool.

Another metric to evaluate the transformation is its transformation time. Experimental results showed that the transformation time is exponential and it can take up to 2 minutes to transform a service released by about 100 events. However, in a the case of a real TDMA frame, there are about 10 to 20 events, which takes between 173 ms to 857 ms to transform.

In conclusion, the experiment in this chapter showed that automatic scheduling analysis of a real software radio protocol, with the DGMF task model and its analysis method, is possible by exploiting an architecture model in UML MARTE.

The experiment focused on software radio protocols. The proposed approach can be extended to other domains. Indeed, MARTE is a generic ADL for RTES. Even at Thales, the analysis method can be adapted for other applications than radio protocols. For example, automatic scheduling analysis of security applications is also studied at Thales Communications & Security. The MARTE architecture model thus needs to be formalized and generalized.

PUBLICATIONS The experimental modeling with MARTE is described in [26].

CONCLUSION

SUMMARY

The work presented in this thesis contributed to scheduling analysis of RTES. This thesis focused on communication systems using TDMA to access the shared communication medium. Systems called software radio protocols were analyzed. Such systems are developed at Thales Communications & Security.

Software radio protocols have some characteristics to consider for scheduling analysis. Among these characteristics, the system has tasks released by TDMA ticks, and the tasks have an execution time and a deadline that depend on the TDMA slots. The tasks are also dependent, through precedence dependency and shared resource. They execute on a partitioned multiprocessor execution platform, with preemptive fixed priority scheduling.

Scheduling analysis is to be performed automatically, so the analysis can be integrated into the development cycle of a software radio protocol. Architecture models of the system are thus to be exploited for the analysis.

For scheduling analysis of a software radio protocol, several issues are solved in this thesis. First, existing task models in the literature are not applicable to all characteristics of a software radio protocol. Second, some task models, considered for extension, are not implemented in existing scheduling analysis tools. Finally, since scheduling analysis is to be applied automatically, a software radio protocol must be modeled in an adapted ADL and transformed to a task model, so the analysis can be performed by a scheduling analysis tool.

The solution to these problems, proposed in this thesis, is to first model the architecture of a software radio protocol in UML MARTE. The proposed model is called Service Model. It contains behavioral views and structural views of UML. The behavioral views are used to describe some services in the system, while the structural views are used to describe some software and execution platform entities.

The UML MARTE architecture model is transformed into the DGMF task model, proposed in this thesis. This task model considers characteristics of a software radio protocol for scheduling analysis. It models individual jobs of a task, called DGMF frames. DGMF extends the GMF task model with task dependencies and the proposed task model is applicable to a partitioned multiprocessor execution platform.

To analyze DGMF tasks, they are transformed to a transaction model. Then an adapted schedulability test, proposed in this thesis, is applied to the transactions. The adapted schedulability test is called WCDOPS+NIM, and it extends the WCDOPS+ test. The proposed test is applicable to tree-shaped transactions with non-immediate tasks.

DGMF, transactions, and their analysis methods are implemented in the Cheddar scheduling analysis tool. Scheduling analysis, with DGMF, can then be applied automatically to a software radio protocol architecture, modeled in UML MARTE.

The proposed solution was evaluated through several experiments. These experiments are either simulations, or they applied of the propositions to real case-studies from Thales.

The transformation of the Service Model in UML MARTE, into a set of DGMF tasks implemented in Cheddar, was evaluated by simulation. A TDMA frame of 14 slots was simulated, since this configuration is typical at Thales. Simulation results showed that the transformation time is between 173 ms and 857 ms. The number of Cheddar-ADL entities, produced by the transformation, is between 137 and 270.

The modeling and transformation of DGMF tasks to transactions was evaluated both by simulation, and its application to a real case-study. Simulation results showed that the implementation of the transformation is polynomial, when the number of DGMF frames increases, or the number of precedence dependencies increases. For a model of 100 DGMF tasks, 1000 DGMF frames, and 1100 precedence dependencies, simulation results showed that the transformation time takes about 160 ms.

DGMF was then applied for the modeling of a real case-study from Thales. This experiment showed that a real system has much less tasks, frames and precedence dependencies. Indeed, the case-study was modeled with 8 DGMF tasks, 43 frames and 14 precedence dependencies. Thus a real case-study takes less than 160 ms to be transformed.

The DGMF model representing the real case-study was transformed into a transaction model. The WCDOPS+NIM test was then applied. Experimental results showed that WCDOPS+NIM computes WCRTs almost 9 times lower in average than the fundamental periodic task model. The approach of the periodic task model analysis is used for some systems at Thales. Simulation results also showed that WCDOPS+NIM gives less pessimistic response times than WCDOPS+, as processor utilization increases. For a processor utilization of a 70%, WCDOPS+NIM gives up to 40% less pessimistic WCRTs.

In conclusion the proposed solution solves the issues faced by automatic scheduling analysis of a software radio protocol. Experimental results also show that the solution is scalable to systems developed at Thales Communications & Security.

FUTURE WORKS

Among the future works of this thesis, some work can be done on the proposed analysis method, the UML MARTE model, some specific execution platforms, and some methodology issues.

Analysis Method

The WCDOPS+NIM schedulability test can be extended so some properties on DGMF tasks do not need to hold anymore.

The analysis method assumes that DGMF tasks respect the *Cycle Separation* property. This means that the deadline of job p of $F_i^{N_i}$ is less than the release of job $p+1$ of F_i^1. By assuming this property, the first frame F_i^1 should not interfere the last frame $F_i^{N_i}$, unless the last frame misses a deadline. To analyze DGMF tasks that do not respect the *Cycle Separation* property, the WCDOPS+NIM schedulability test needs to consider this behavior. When each DGMF task is transformed into a transaction, job $p+1$ of a transaction Γ_a can simply be forbidden to interfere any job $p < p+1$ of the same transaction. A possible solution is to add reduction rules and adapt equations for interference of jobs $p > 0$. But when several DGMF tasks are transformed into a merged transaction, some parameter must indicate that some tasks were originally frames part of a same DGMF task.

Another possible extension of the test concerns the operation that checks which non-immediate tasks are to be considered immediate, for a given scenario created during the analysis. The condition is that a non-immediate task released before t_c is necessarily immediately released by its predecessor, if it contributes to the busy period. In some cases, tasks released after t_c, and within the busy period, are also released immediately by their predecessor. The goal of extending the condition is to compute less pessimistic WCRTs.

Finally the WCDOPS+NIM test is applicable to tree-shaped transactions. There exists an extension of WCDOPS+ for graph-shaped transactions in [66]. In the future, the WCDOPS+NIM test can be adapted for graph-shaped transactions with non-immediate tasks.

Service Model in UML MARTE

The experiment on modeling UML structural and behavioral views to describe a software radio protocol, consisted in defining the Service Model and mapping it to MARTE concepts. The Service Model was not formalized. A formalization work consists in defining all of the constraints of the Service Model. Once formalized, the model can be compared to other similar approaches.

Specific Execution Platform

In this thesis the execution platform has some assumptions described in Section 3.2.1. Although the described execution platform is one that hosts a number of software radio protocols, there are others that have specific characteristics.

For example in the PikeOS [65] operating system, the implemented POSIX scheduler has a fair play feature. This feature allows a task to call a `yield()` instruction in its code. By calling this instruction, the task informs the scheduler that it wants to be rescheduled and put at the tail of the scheduling queue. This behavior must be considered by scheduling analysis.

Some execution platforms of Thales products also have middlewares to ease communication between different tasks on heterogeneous platforms. A middleware lies between the application and the OS. A middlesware has software buses, handling messages exchanged between tasks. The software buses may have a specific real-time behavior [123]. Middlewares are implemented in some products of Thales Communications & Security. In the future, these entities should be considered for the analysis of a software radio protocol. Some issues to solve are the modeling of middleware components and the description of their interaction with the rest of the system.

Finally some execution platforms offer the possibility to vary the frequency of the processor, or to turn the processor off. This capability is called Dynamic Voltage and Frequency Scaling (DVFS) [6, 79] It exploits the fact that jobs of tasks in a system do not always run at their WCET. A job may then have slack time, which is the difference between the task's WCET, and the actual execution time of the job. In the future, DVFS is to be integrated into Thales products because it offers energy consumption gains [6]. For this reason, DVFS mechanisms should be described in the UML MARTE architecture models [2]. The service behavior view, of the Service Model, can also be exploited to estimate some initial energy consumption performance of the system.

Scheduling Simulation

Due to some characteristics, like specific execution platform characteristics, there may not exist a task model and a scheduling analysis method applicable to the system. In this case scheduling simulation can be used to get a first estimation of tasks scheduling. Scheduling simulation may not assess schedulability, but it can assess non-schedulability. Furthermore there is an interest at Thales for scheduling simulation among development teams.

Cheddar has a scheduling simulator. Works on scheduling simulation have been done as requested by Thales. The scheduling simulation exploits the Service Model and the transaction model. It also considers some execution platform characteristics like the possibility to call `yield()`.

Some problems had to be solved because of the difference in semantic between the Service Model and the Cheddar-ADL model. The difference in semantic has no impact on the results of schedulability analysis, but during simulation some behaviors do not respect the reality. This issue has been solved but scheduling simulation should be investigated further.

Methodology Issues

Currently the scheduling analysis method, proposed in this thesis, requires various information. It needs the services in a system, their release pattern, release jitters (if not negligible), execution times, critical sections, deadlines, threads, shared resources, scheduling policies, and processors.

A recent survey was conducted at Thales Communications & Security to assess the availability of these information. The survey took a sample of engineers who worked at different steps of the development cycle. The outcome of the survey was that basic information for scheduling analysis, such as the scheduling policy, were not necessarily available at some steps. Therefore the applicability of the analysis method, at different steps of the V-model development cycle of a software radio protocol, should be investigated.

For example, the impact of the WCET value should be investigated. At earlier steps of the development cycle, the WCET can be specified as an allowed budget. At later steps of the development cycle, a more accurate WCET can be computed or estimated. It is proposed in [149] that future works on scheduling analysis should be combined with works on WCET analysis. This issue is being investigated by Thales through several European collaborative projects (e.g. PRESTO [26]).

Part III

APPENDIX

Appendix A

CHEDDAR-ADL

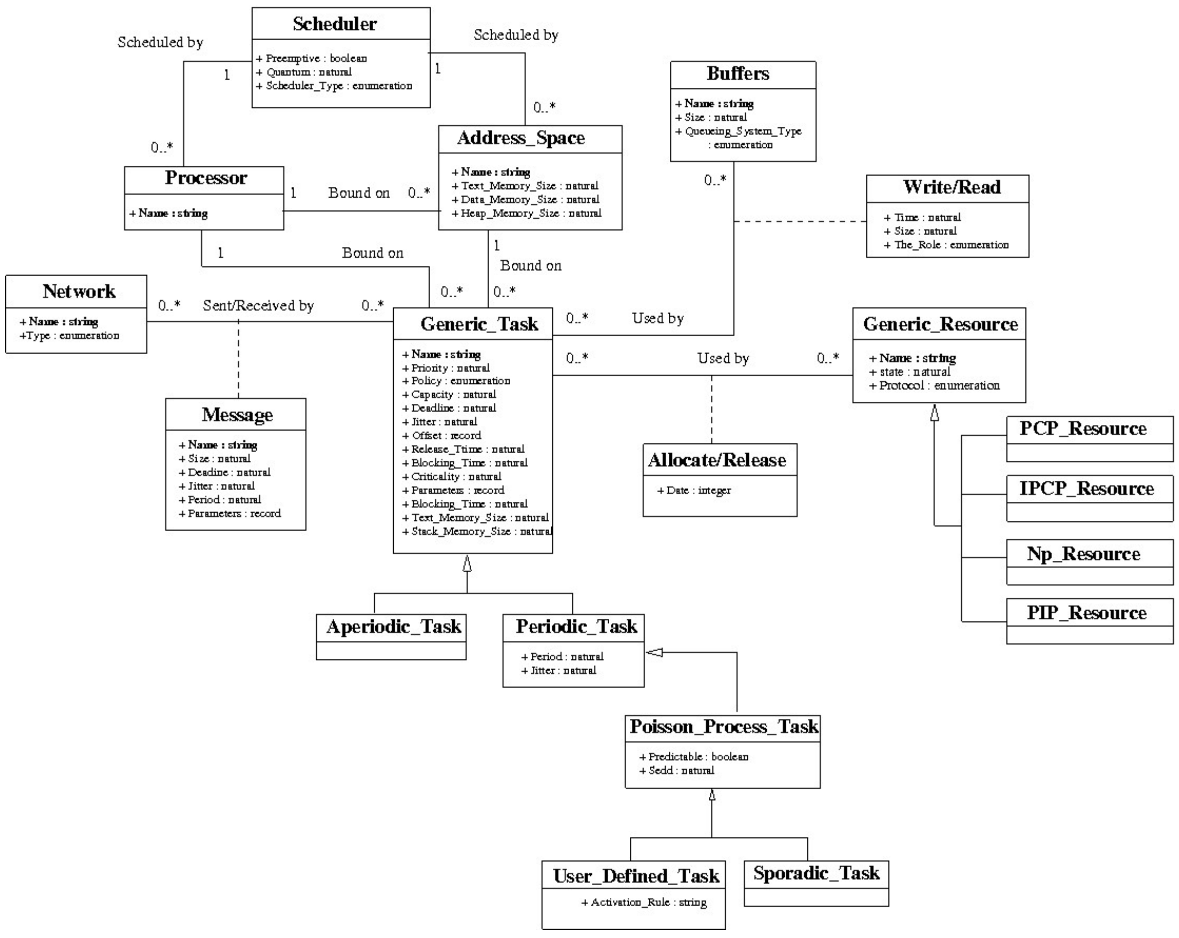


Figure A.1: Original Cheddar-ADL Meta-Model

Appendix B

COMPLETE SOFTWARE RADIO PROTOCOL CASE-STUDY

Table B.1: Task Parameters of Real Case-Study

Task	C_{ij}	O_{ij}	$O_{ij}+d_{ij}$	J_{ij}	B_{ij}	$prio(\tau_{ij})$	$proc(\tau_{ij})$
Tick	0	0	$+\infty$	0	0	1	NoCPU
FC_S1_1	1050	0	30000	0	0	10	GPP1
FC_S1_2	1100	1050	30000	0	0	10	GPP2
FC_S1_3	573	2150	30000	0	0	10	GPP1
RS_B1_1	144	6157	7357	0	0	60	GPP1
AB_B1_1	500	6301	18301	0	0	30	GPP1
AB_B1_2	1500	6801	18301	0	0	30	GPP2
AB_B1_3	597	8301	18301	0	0	30	GPP1
RS_B2_1	144	7467	8667	0	0	60	GPP1
AB_B2_1	500	8898	20898	0	0	30	GPP1
AB_B2_2	1500	9398	20898	0	0	30	GPP2
AB_B2_3	597	10898	20898	0	0	30	GPP1
RS_B3_1	144	8777	9977	0	0	60	GPP1
AB_B3_1	500	11495	23495	0	0	30	GPP1
AB_B3_2	1500	11995	23495	0	0	30	GPP2
AB_B3_3	597	13495	20777	0	0	30	GPP1
RS_B4_1	144	10087	11287	0	0	60	GPP1
AB_B4_1	500	14092	26092	0	0	30	GPP1
AB_B4_2	1500	14592	26092	0	0	30	GPP2
AB_B4_3	597	16092	26092	0	0	30	GPP1
RS_T2_1	144	18864	20064	0	0	60	GPP1
AD_T2_1	651	19008	26364	0	0	40	GPP1
RS_T4_1	144	31178	32378	0	0	60	GPP1
AD_T4_1	651	31322	38678	0	0	40	GPP1
RS_T6_1	144	43492	44692	0	0	60	GPP1
AD_T6_1	651	43636	50992	0	0	40	GPP1
RS_T8_1	144	55806	57006	0	0	60	GPP1
AD_T8_1	651	55950	63306	0	0	40	GPP1
RM_T1_1	50	5507	6707	0	0	60	GPP1
BT_T1_1	463	5651	11651	0	0	50	GPP1
RM_T3_1	50	17821	19021	0	0	60	GPP1
BT_T3_1	463	17965	23965	0	0	50	GPP1
RM_T5_1	50	30135	31335	0	0	60	GPP1
BT_T5_1	463	30279	36279	0	0	50	GPP1
RM_S1_1	50	35763	36963	0	0	60	GPP1
BS_S1_1	7193	35813	60813	0	0	20	GPP1
BS_S1_2	6000	43006	60813	0	0	20	GPP2
BS_S1_3	663	49006	60813	0	0	20	GPP1
RM_T7_1	50	42449	43649	0	0	60	GPP1
BT_T7_1	463	42593	48449	0	0	50	GPP1
RM_B5_1	50	47160	36963	0	0	60	GPP1
BB_B5_1	1017	47210	72210	0	0	20	GPP1
BB_B5_2	1100	48227	72210	0	0	20	GPP2
BB_B5_3	563	49327	72210	0	0	20	GPP1

Appendix C

SERVICE MODEL

C.1 SERVICE ANALYSIS VIEW

C.1.1 *Entities*

Table C.1 lists and describes the entities that are created in the service analysis view.

C.1.2 *Mapping to MARTE*

The GQAM sub-profile of MARTE is used in the service analysis view.
Table C.2 shows how each entity and attribute is mapped to a concept in UML MARTE.

C.1.3 *Diagram Palette*

In Papyrus, the service analysis view is represented by a class diagram. Table C.3 shows the diagram palette of the service analysis view.

C.2 SERVICE BEHAVIOR VIEW

C.2.1 *Entities*

Table C.4 lists and describes the entities that are created in the service behavior view.

C.2.2 *Mapping to MARTE*

The Time, GQAM, and SAM sub-profiles of MARTE are used in the service behavior view. The GQAM and SAM sub-profiles are dedicated to analysis models. The Time sub-profile is used to represent time constraints.
Table C.5 shows how each entity and attribute is mapped to a concept in UML MARTE.

Table C.1: Service Analysis Entities: Grey rows are entities; White rows are attributes

Entity	Description
ServiceAnalysisContext	The classifier that contains the services to analyze
- services	List of services to analyze

Table C.2: Service Analysis Mapping: Grey rows are entities; White rows are attributes

Entity	UML MARTE
ServiceAnalysisContext	*Class* stereotyped <<SaAnalysisContext>>
- services	Attribute *workload* of <<SaAnalysisContext>>

Table C.3: Service Analysis Palette: Grey rows are entities; White rows are attributes

UML Element	UML / MARTE	Attributes
Class	SaAnalysisContext	workload

Table C.4: Service Behavior Entities: Grey rows are entities; White rows are attributes

Entity	Description
Service	An activity of several steps, implemented by one or several threads, released by one or several events
- sources	List of sources of the service that are released by events
Source	A service is released at a source by events
- eventGenerator	Event generator that will produce events to release the service at the designated source
Step	An action to execute
- wcet	Worst case execution time of the step
- jitter	A step is released right after the preceding event (i.e. source event or preceding step's completion) but a step may have a release jitter.
- thread	A step is allocated on a thread implementation.
- deadline	Deadline step relative to the time that the event releases the service
- resourceAcquisitions	Critical sections of the step (for shared resources)
Deadline	Time constraint of a step
- value	Value of the deadline
ResourceAcquisitionStep	To specify a critical section
- resource	Shared resource to access in critical section
- acquisitionTime	Start time of the critical section, within the execution time of a step
- lockingTime	Duration of the critical section
Transition	An arrow representing a precedence dependency between two steps
- guard	The transition may have a guard: a boolean condition that must be passed for the next step to be released. The boolean condition tests variables embedded in the event that releases the service.
BlockingCall	A blocking call, between steps S1 and S2, indicates that job (k+1) of a step S1 can only be released after job k of another step S2 is complete.
Sink	End of service

Table C.5: Service Behavior Mapping: Grey rows are entities; White rows are attributes; If an attribute can be a LiteralInteger or a LiteralString, and it is a LiteralString, then it is interpreted during transformation to get an integer at the end

Entity	UML MARTE
Service	**Activity** stereotyped <<GaWorkloadBehavior >>
- sources	Attribute *demand* of <<GaWorkloadBehavior >>as list of <<GaWorkloadEvent>>
Source	**InitialNode** stereotyped <<GaWorkloadEvent>>
- eventGenerator	Attribute *generator* of <<GaWorkloadEvent>> as <<GaWorkloadGenerator>> owned by <<EventGenerator**>>** or <<TDMAFrame>>
Step	**OpaqueAction** stereotyped <<SaStep>>
- wcet	Attribute *execTime* of <<SaStep>> as **LiteralInteger** or **LiteralString** expression (variables interpreted at transformation time, e.g. "SLOT_DURATION / 2")
- jitter	Attribute ***interOccrT*** of <<***SaStep***>> as ***LiteralInteger*** or ***LiteralString*** expression
- thread	Attribute *concurRes* of <<SaStep>> as <<SwSchedulableResource>>
- deadline	Attribute *localPostcondition* of **OpaqueAction** as **Constraint** stereotyped <<TimedConstraint>>
- resourceAcquisitions	Attribute *localPrecondition* of **OpaqueAction** as **Constraint** stereotyped <<GaAcqStep>> (several **Constraints** possible)
Deadline	**Constraint** stereotyped <<TimedConstraint>>
- value	*Specification* of **Constraint** as **LiteralInteger** or **LiteralString** expression
ResourceAcquisitionStep	**Constraint** stereotyped <<GaAcqStep>>
- resource	Attribute *acqRes* of <<GaAcqStep>> as <<SwMutualExclusionResource>>
- acquisitionTime	*Specification* of **Constraint** as **LiteralInteger**
- lockingTime	Attribute *blockT* of <<GaAcqStep>> as integer
Transition	**ControlFlow**
- guard	*Guard* of **ControlFlow** as **LiteralString** expression (variables interpreted at transformation time, e.g. ' SLOT_TYPE == "T" ')
BlockingCall	**ControlFlow** stereotyped <<SaCommStep>>
Sink	**ActivityFinal**

Table C.6: Service Behavior Palette: Grey rows are entities; White rows are attributes

UML Element	UML / MARTE	Attributes
Activity	GaWorkloadBehavior	demand
Initial Node	GaWorkloadEvent	generator
Opaque Action	SaStep	concurRes
		execTime
		interOccrT
Constraint as LocalPrecondition	GaAcqStep	acqRes
		blockT
	GaRelStep	relRes
	UML	specification
Constraint as LocalPostcondition	TimedConstraint	
	UML	specification
	GaRequestedService	
Activity Final Node		
Control Flow	GaRelStep	
	UML	guard

C.2.3 *Diagram Palette*

In Papyrus, the service behavior view is represented by an activity diagram. Table C.6 shows the diagram palette of the service behavior view.

C.3 EVENT GENERATOR VIEW

C.3.0.1 *Entities*

Table C.7 lists and describes the entities that are created in the event generator view.

C.3.0.2 *Mapping to Event Generator Profile*

Table C.8 shows how each entity and attribute is mapped to a concept in UML MARTE.

C.3.0.3 *Diagram Palette*

In Papyrus, the event generator view is represented by a class diagram. Table C.9 shows the diagram palette of the event generator view.

C.4 THREAD IMPLEMENTATION VIEW

C.4.1 *Entities*

Table C.10 lists and describes the entities that are created in the thread implementation view.

Table C.7: Event Generator Entities: Grey rows are entities; White rows are attributes; RandomEventGenerator is only used for experiments

Entity	Description
Event	An event that releases a service at a source of the service
- time	The time the event occurs for the first time
- variables	A list of variables embedded within the event
Variable	A variable embedded in an event, that may be tested in transitions between steps of the service released by the event
- name	Name of the variable
- variableType	Enum of value INTEGER, BOOLEAN, STRING or DOUBLE
- <intValue, doubleValue, booleanValue, stringValue>	Value of the variable; One attribute has a value, according to variableType
EventGenerator	Generates events and defines the pattern of the occurrences of events
- events	By default a first event is generated at time t=0, with no variables. This default behavior can be overwritten by defining a list of first events (optionally with different times and variables attributes).
PeriodicEventGenerator	Generates periodic events
- period	An event occurs every period. If events are specified in the events list, then these events occur for the first time at their specified time attribute, then they occur periodically (with the same variables, if any).
SporadicEventGenerator	Generates sporadic events with a minimum delay
RandomEventGenerator	Generates pseudo-random events
- start	Time of first event
- end	Last event cannot be after this time
- maxInterOccr	Maximum time between two events
TDMAFrame	A TDMA frame
- slotList	An ordered list of slots
TDMASlot	A TDMA slot
- type	Enum of value S, B or T
- mode	Enum of value TX, RX or IDLE
- duration	Duration of the slot as integer

Table C.8: Event Generator Mapping: Grey rows are entities; White rows are attributes

Entity	UML MARTE
Event	**Property** stereotyped <<Event>>
- time	Attribute *time* of <<Event>> as **Integer**
- variables	Attribute *variables* of <<Event>> as list of <<Variable>>
Variable	**Property** stereotyped <<Variable>>
- name	Attribute *time* of <<Variabl >> as **String**
- variableType	Attribute *variableType* of <<Variable>> as **EnumerationLiteral** of value INTEGER, BOOLEAN, STRING or DOUBLE
- intValue, doubleValue, booleanValue, stringValue	Attribute *[intValue, doubleValue, BooleanValue, stringValue]* of <<Variable>> as **[Integer, Double , Boolean, String]** (A string is used to specify a boolean or double value.)
EventGenerator	Cannot be instantiated but its children can be.
- events	Attribute *events* of <<EventGenerator>> as list of <<Event>>
PeriodicEventGenerator	**Component** stereotyped <<ClockResource>> and <<PeriodicEventGenerator>>
- period	Attribute *period* of <<PeriodicEventGenerator>> as **Integer**
SporadicEventGenerator	**Component** stereotyped <<ClockResource>> and <<SporadicEventGenerator>>
RandomEventGenerator	**Component** stereotyped <<ClockResource>> and <<RandomEventGenerator>>
- start	Attribute *start* of <<RandomEventGenerator>> as **Integer**
- end	Attribute *end* of <<RandomEventGenerator>> as **Integer**
- maxInterOccr	Attribute *maxInterOccr* of <<RandomEventGenerator>> as **Integer**
TDMAFrame	**Component** stereotyped <<ClockResource>> and <<TDMAFrame>>
- slotList	Attribute *slotList* of <<FileEventGenerator>> as list of <<TDMASlot>>
TDMASlot	**Property** stereotyped <<TDMASlot>>
- type	Attribute *type* of <<TDMASLot>> as **EnumerationLiteral** of value S, B or T
- mode	Attribute *mode* of <<TDMASLot>> as **EnumerationLiteral** of value TX, RX or IDLE
- duration	Attribut *duration* of <<TDMASLot>> as **Integer**

Table C.9: Event Generator Palette: Grey rows are entities; White rows are attributes

UML Element	TDMA and Event Generator Profiles
Component	ClockResource
	PeriodicEventGenerator
	SporadicEventGenerator
	RandomEventGenerator
	TDMAFrame
Property	Event
	Variable
	TDMASlot

Table C.10: Thread Implementation Entities: Grey rows are entities; White rows are attributes

Entity	Description
MemoryPartition	A memory partition that contains thread implementations
Thread	A thread
- name	Name of the thread
- priority	Fixed priority of the thread
- scheduledBy	The scheduler implementation on which the thread is hosted on.
ThreadImplementation	An implementation of the thread. A step of a service can only be executed by a thread implementation.
- name	Name of the thread implementation
- priority	Fixed priority of the thread implementation
- scheduledBy	The scheduler implementation on which the thread implementation is hosted on

Table C.11: Thread Implementation Mapping: Grey rows are entities; White rows are attributes

Entity	UML MARTE
MemoryPartition	**Component** stereotyped <<MemoryPartition>>
Thread	**Component** stereotyped <<SwSchedulableResource>>
- name	*Name* of **Component**
- priority	**Property** that is first element in the attribute *priorityElements* of <<SwSchedulableResource>>
- scheduledBy	Attribute *host* of <<SwSchedulableResource>> as <<Scheduler>>
ThreadImplementation	**Property**, stereotyped <<SwSchedulableResource>>, of a memory partition, and typed by a thread
- name	*Name* of **Property**
- priority	*Qualifier* (a **Property**) of the **Property** representing the thread implementation, and first element in the attribute *priorityElements* of <<SwSchedulableResource>>. Unspecified ➔ Takes the priority of the typed thread
- scheduledBy	Attribute *host* of <<SwSchedulableResource>> Unspecified ➔ Takes the scheduledBy of the typed thread

C.4.2 *Mapping to MARTE*

The sub-profiles SRM of MARTE are used to model entities of the shared resource implementation view. The SRM sub-profile is dedicated to design models of the software.

Table C.11 shows how each entity and attribute is mapped to a concept in UML MARTE.

C.4.3 *Diagram Palette*

In Papyrus, the service analysis view is represented by a composite structure diagram. Table C.12 shows the diagram palette of the service analysis view.

Table C.12: Thread Implementation Palette: Grey rows are entities; White rows are attributes

UML Element	UML / MARTE	Attributes
Component	MemoryPartition	
Component	SwSchedulableResource	Same as Property with <<SwSchedulableResource>> stereotype
Property	SwSchedulableResource	qualifier
		priorityElements
		host

Table C.13: Shared Resource Implementation Entities: Grey rows are entities; White rows are attributes

Entity	Description
MemoryPartition	A memory partition containing shared resource implementations
SharedResource	A shared resource
- name	Name of the shared resource
- host	The scheduler implementation (and by extension processor) on which the shared resource is allocated on
- protocol	The protection protocol of the shared resource
SharedResourceImplementation	An implementation of the shared resource. A step of a service can only use a shared resource implementation.
- name	Name of the shared resource implementation
- host	The scheduler implementation (and by extension processor) on which the shared resource implementation is allocated on
- protocol	The protection protocol of the shared resource

C.5 SHARED RESOURCE IMPLEMENTATION VIEW

C.5.1 *Entities*

Table C.13 lists and describes the entities that are created in the shared resource implementation view.

C.5.2 *Mapping to MARTE*

The sub-profile SRM of MARTE is used to model entities of the shared resource implementation view. The SRM sub-profile is dedicated to design models of the software.

Table C.14 shows how each entity and attribute is mapped to a concept in UML MARTE.

C.5.3 *Diagram Palette*

In Papyrus, the shared resource implementation view is represented by a composite structure diagram. Table C.15 shows the diagram palette of the shared resource implementation view.

Table C.14: Shared Resource Implementation Mapping: Grey rows are entities; White rows are attributes

Entity	UML MARTE
MemoryPartition	**Component** stereotyped <<MemoryPartition>>
SharedResource	**Component** stereotyped <<SwMutualExclusionResource>>
- name	*Name* of **Component**
- host	Attribute *scheduler* of <<SwMutualExclusionResource>> as <<Scheduler>>
- protocol	Attribute *concurrentAccessProtocol* of <<SwMutualExclusionResource>> as EnumerationLiteral of value PCP, PIP, Undef
SharedResourceImplementation	**Property**, stereotyped <<SwMutualExclusionResource>>, of a memory partition, and typed by a sharedResource
- name	*Name* of **Property**
- host	Attribute *scheduler* of <<SwMutualExclusionResource>> as <<Scheduler>> Unspecified ➔ Takes the host of the typed shared resource
- protocol	Attribute *concurrentAccessProtocol* of <<SwMutualExclusionResource>> as EnumerationLiteral of value PCP, PIP, Undef Unspecified ➔ Takes the protocol of the typed shared resource

Table C.15: Shared Resource Implementation Palette: Grey rows are entities; White rows are attributes

UML Element	UML / MARTE	Attributes
Component	MemoryPartition	
Component	SwMutualExclusionResource	Same as Property with <<SwMutualExclusionResource>> stereotype
Property	SwMutualExclusionResource	concurrentAccessProtocol
		scheduler

Table C.16: Operating System Implementation Entities: Grey rows are entities; White rows are attributes

Entity	Description
OperatingSystem	An operating system
Scheduler	A scheduler
- name	Name of the scheduler
- preemptive	Preemptivity
- policy	Scheduling policy
- host	The processor implementation scheduled by the scheduler
SchedulerImplementation	An implementation of the scheduler. A thread can only be allocated on a scheduler implementation
- name	Name of the scheduler implementation
- preemptive	Preemptive or non-preemptive
- policy	Scheduling policy
- host	The processor implementation scheduled by the schedulerImplementation

C.6 OPERATING SYSTEM IMPLEMENTATION VIEW

C.6.1 *Entities*

Table C.16 lists and describes the entities that are created in the operating system implementation view.

C.6.2 *Mapping to MARTE*

The sub-profiles GQAM and GRM of MARTE are used to model entities of the operating system implementation view. GQAM is used to model the operating system itself, while GRM is used to model the schedulers.

Table C.17 shows how each entity and attribute is mapped to a concept in UML MARTE.

C.6.3 *Diagram Palette*

In Papyrus, the operating system implementation view is represented by a composite structure diagram. Table C.18 shows the diagram palette of the operating system implementation view.

C.7 HARDWARE IMPLEMENTATION VIEW

C.7.1 *Entities*

Table C.19 lists and describes the entities that are created in the hardware implementation view.

C.7.2 *Mapping to MARTE*

The sub-profiles GQAM and HRM of MARTE are used to model entities of the shared resource implementation view. GQAM is used to model the hardware itself, while HRM is used to model the processors.

Table C.20 shows how each entity and attribute is mapped to a concept in UML MARTE.

Table C.17: Operating System Implementation Mapping: Grey rows are entities; White rows are attributes

Entity	UML MARTE
OperatingSystem	**Component** stereotyped <<GaResourcesPlatform>>
Scheduler	**Component** stereotyped <<Scheduler>>
- name	*Name* of **Component**
- preemptive	Attribute *isPreemptible* of <<Scheduler>> as **LiteralBoolean**
- policy	Attribute *schedPolicy* of <<Scheduler>> as **EnumerationLiteral** of value EarliestDeadlineFirst, FixedPriority
- host	Attribute *host* of <<Scheduler>> as <<HwProcessor>>
SchedulerImplementation	**Property**, stereotyped <<Scheduler>>, of an operating system, and typed by a scheduler
- name	*Name* of **Property**
- preemptive	Attribute *isPreemptible* of <<Scheduler>> as **LiteralBoolean** Unspecified ➔ Takes preemptive of typed scheduler
- policy	Attribute *schedPolicy* of <<Scheduler>> as **EnumerationLiteral** of value EarliestDeadlineFirst, FixedPriority Unspecified ➔ Takes policy of typed scheduler
- host	Attribute *host* of <<Scheduler>> as <<HwProcessor>> Unspecified ➔ Takes host of typed scheduler

Table C.18: Operating System Implementation Palette: Grey rows are entities; White rows are attributes

UML Element	UML / MARTE	Attributes
Component	GaResourcesPlatform	
Component	Scheduler	isPreemptible schedPolicy
Property	Scheduler	Same as component with <<Scheduler>> stereotype

Table C.19: Hardware Implementation Entities: Grey rows are entities; White rows are attributes

Entity	Description
Hardware	Hardware board
Processor	A processor
- name	Name of the processor
ProcessorImplementation	An implementation of the processor. A scheduler can only schedule a processor implementation
- name	Name of the processor implementation

Table C.20: Hardware Implementation Mapping: Grey rows are entities; White rows are attributes

Entity	UML MARTE
Hardware	**Component** stereotyped <<GaResourcesPlatform>>
Processor	**Component** stereotyped <<HwProcessor>>
- name	*Name* of **Component**
ProcessorImplementation	**Property**, stereotyped <<HwProcessor>>, of a hardware, and typed by a processor
- name	*Name* of **Property**

Table C.21: Hardware Implementation Palette: Grey rows are entities; White rows are attributes

UML Element	UML / MARTE
Component	GaResourcesPlatform
Component	HwProcessor
Property	HwProcessor

C.7.3 *Diagram Palette*

In Papyrus, the hardware implementation view is represented by a composite structure diagram. Table C.21 shows the diagram palette of the hardware implementation view.

Appendix D

SERVICE TO CHEDDAR

D.1 MAIN PROGRAM

Algorithm D.1 is the S2C main program.

Algorithm D.1 Service to Cheddar

```
 1: existingEvtGens : Map<EventGenerator, Transaction>
 2: cModel : Cheddar_Model
 3: for each sac : ServiceAnalysisContext in model do
 4:     Clear cModel
 5:     Clear existingEvtGens
 6:     for each serv in sac.services do
 7:         for each src in serv.sources do
 8:             evtGen ← src.eventGenerator
 9:             trans : Transaction
10:             if any <key, val> in existingEvtGens has key = evtGen then
11:                 trans ← val
12:             else
13:                 trans.period ← processEventGenerator(evtGen)            ▷ Compute period
14:                 Put <evtGen, trans> in existingEvtGens
15:                 Add trans to cModel
16:             end if
17:             prevTaskStepMap : Map<Step, Generic_Task>
18:             sortByTimeAsc(evtGen.events)                    ▷ Sort by ascending event time
19:             for each evt in evtGen.events do
20:                 Affect evt.variables
21:                 currTaskStepMap : Map<Step, Periodic_Task>
22:                 jobPrecs : List<Precedence_Dependency>
23:                 processNode(src, evt.time, null, trans, currTaskStepMap, prevTaskStepMap, jobPrecs)
24:                 reduceJobPrecs(jobPrecs)
25:                 Add each prec in jobPrecs to cModel
26:             end for
27:         end for
28:     end for
29: end for
```

Like stated previously, Algorithm D.1 will transform the model for each service analysis context and service. A service is explored by source and event. Event variables are assigned in memory, for transition guard assertion, before the exploration.

As a reminder, a transaction corresponds to an event generator so their mapping is done with the *existingEvtGens* map. At line 13, the transaction period is computed according to the event generator. If the event generator is periodic or sporadic, the period attribute is the transaction's period. If the event generator is a TDMA frame, the duration of the TDMA frame is taken as period of the transaction.

The maps *prevTaskStepMap* and *currTaskStepMap* are used during step processing to create job precedence dependency in *jobPrecs*. Remember that a task is created for each step during the exploration of the service. An exploration is done for each event releasing the source. A job precedence dependency is one that exists between tasks that represent steps released by a same event generator at a same source, but by different events (i.e. different instances of the step). For example in Figure 7.10, *Task11* precedes *Task12* because *Task11* is created for *Step1* released by the first event and *Task12* is created for *Step1* released by the second event.

At line 24, job precedence dependencies are reduced. Indeed, during the processing of the service, some redundant precedence dependencies are created. For example, consider a task where job 1 precedes directly job 2, job 2 precedes directly job 3, and job 1 precedes directly job 3. The precedence dependency from job 1 to job 3 should be removed. This is done with the following rule; consider two precedence dependencies: $\tau_i \prec \tau_j$ and $\tau_p \prec \tau_q$. The precedence dependency $\tau_i \prec \tau_j$ is removed if:

$$(\tau_i = \tau_p \vee \tau_i \prec \tau_p) \wedge (\tau_j = \tau_q \vee \tau_q \prec \tau_j) \tag{114}$$

This reduction is done in the procedure in Algorithm D.2.

Algorithm D.2 ReduceJobPrecs Procedure

```
1:  procedure REDUCEJOBPRECS(jobPrecs : List<Precedence_Dependency>)
2:      $k \leftarrow 0$
3:      while $k <$ jobPrecs.length do
4:          $\tau_i \leftarrow$ jobPrecs[k].predecessor
5:          $\tau_j \leftarrow$ jobPrecs[k].successor
6:          $m \leftarrow 0$
7:          removed $\leftarrow$ false
8:          while $m <$ jobPrecs.length and not removed do
9:              if jobPrecs[m] $\neq$ jobPrecs[m] then
10:                 $\tau_p \leftarrow$ jobPrecs[m].predecessor
11:                 $\tau_q \leftarrow$ jobPrecs[m].successor
12:                 if $(\tau_i = \tau_p \vee \tau_i \prec \tau_p) \wedge (\tau_j = \tau_q \vee \tau_q \prec \tau_j)$ then
13:                     Remove jobPrecs[k] from jobPrecs
14:                     $k \leftarrow k - 1$
15:                     removed $\leftarrow$ true
16:                 end if
17:             end if
18:             $m \leftarrow m + 1$
19:         end while
20:         $k \leftarrow k + 1$
21:     end while
22: end procedure
```

D.2 NODE PROCESSING

Algorithm D.3 is the recursive procedure that explores a service by nodes. A node is either a source, step or sink.

The basic idea behind Algorithm D.3 is to do an exploration by depth-first search. When a node is a step, it is processed (line 3), i.e. a task is created. The created task is added to the transaction corresponding to the event generator that released the step (line (line 4). Before moving on to the

Algorithm D.3 ProcessNode Procedure

```
 1: procedure ProcessNode(node : Node, offset : Integer, pred : Periodic_Task, trans : Transaction, currTaskStepMap : Map<Step, Periodic_Task>, prevTaskStepMap : Map<Step, Periodic_Task>, jobPrecs : List<Precedence_Dependency>)
 2:     if node is Step then
 3:         tsk ← processStep(node, offset, pred, trans, currTaskStepMap, prevTaskStepMap, jobPrecs)    ▷ Returns a Periodic_Task with filled parameters
 4:         Add tsk to trans
 5:         offset ← offset + tsk.capacity
 6:         pred ← tsk
 7:     end if
 8:     transitions ← outgoing Transitions of node
 9:     if transitions is not empty then
10:         for each transition in transitions do
11:             if transition.guard is true then
12:                 Assign variables of transition.guard
13:                 nextNode ← target of transition
14:                 processNode(nextNode, offset, pred, trans, currTaskStepMap, prevTaskStepMap, jobPrecs)
15:             end if
16:         end for
17:     end if
18: end procedure
```

next node, offset and predecessor tasks are updated (lines 5 and 6. The next nodes are connected to the current one by outgoing transitions (line 8). In a transition, the current node is the source and the next node is the target. The exploration moves to a target node (line 14) if the transition guard assertion gives true (line 11).

D.3 STEP PROCESSING

Algorithm D.4 is the procedure that processes a step by creating its corresponding task and setting its parameters.

Setting most parameters of the task (corresponding to a step) is straightforward. Some parameters like priority, address space name, and cpu name are set by first processing the thread on which the step is allocated on (line 8). Shared resource and their critical sections are created by processing resource acquisitions of the step (line D.8).

Between lines 15 and 25, blocking calls are handled. As a reminder, a blocking call going from step S2 to S1 means that instance k of S2 must finish before instance $(k+1)$ of S1 can be released. If task τ_{2k} corresponds to instance k of S2 and $\tau_{1(k+1)}$ to instance $(k+1)$ of S1, then we have $\tau_{2k} \prec \tau_{1(k+1)}$.

Finally between lines 26 and 30 job precedence dependencies are added to *jobPrecs*. Both blocking calls and job precedence dependencies can be created through the maps *currTaskStepMap* and *prevTaskStepMap*.

D.4 THREAD PROCESSING

Algorithm D.5 is the procedure that processes a thread implementation on which a step is allocated on.

Besides setting task parameters for the task that corresponds to the step, the procedure also processes and creates processor and address space entities in the Cheddar model. The processor is created by processing the scheduler by which the thread is scheduled (line 3).

Algorithm D.4 ProcessStep Function

```
 1: function ProcessStep(step : Step, offset : Integer, pred : Periodic_Task, trans : Transaction, currTaskStepMap : Map<Step, Periodic_Task>, prevTaskStepMap : Map<Step, Periodic_Task>, jobPrecs : List<Precedence_Dependency>)
 2:     tsk : Periodic_Task
 3:     Add tsk to cModel                                          ▷ cModel is global
 4:     tsk.name ← <unique identifier>
 5:     tsk.capacity ← step.wcet
 6:     tsk.deadline ← step.deadline.value
 7:     tsk.offset ← offset
 8:     processThread(step.thread, tsk) ▷ Fills priority, cpu_name, address_space_name parameters of tsk and creates processor and scheduler in cModel
 9:     for each resAcq in step.resourceAcquisitions do
10:         processCriticalSection(resAcq, tsk)        ▷ Fills critical_sections list of tsk and creates resources in cModel
11:     end for
12:     if pred ≠ null then
13:         Add Precedence_Dependency pred ≺ tsk to cModel
14:     end if
15:     blockingCalls ← incoming BlockingCalls of step
16:     if blockingCalls is not empty then
17:         for each blockingCall in blockingCalls do
18:             sourceStep ← source of blockingCall
19:             for each <kTask, vStep> in prevTaskStepMap in DESC order do
20:                 if vStep = sourceStep then
21:                     Add Precedence_Dependency kTask ≺ tsk to jobPrecs
22:                 end if
23:             end for
24:         end for
25:     end if
26:     for each <kTask, vStep> in prevTaskStepMap do
27:         if vStep = step then
28:             Add Precedence_Dependency kTask ≺ tsk to jobPrecs
29:         end if
30:     end for
31:     return tsk
32: end function
```

Algorithm D.5 ProcessThread Procedure

```
1: procedure ProcessThread(thread : ThreadImplementation, tsk : Periodic_Task)
2:     tsk.priority ← thread.priority
3:     cpu ← processScheduler(thread.scheduledBy)     ▷ Returns a Generic_Processor and creates processor and scheduler entities in cModel
4:     tsk.cpu_name ← cpu.name
5:     addr ← processPartition(thread, cpu)   ▷ Returns a Address_Space and creates address space entity in cModel
6:     tsk.addr_space_name ← addr.name
7: end procedure
```

D.5 SCHEDULER AND PROCESSOR PROCESSING

Algorithm D.6 is the function that processes a scheduler and the processor that hosts it. The function returns the processor that was created.

Algorithm D.6 ProcessScheduler Function

```
1: function ProcessScheduler(sched : SchedulerImplementation)
2:     sched_params : Scheduling_Parameters
3:     sched_params.preemptive_type ← sched.preemptive
4:     sched_params.scheduler_type ← sched.policy
5:     cpu : Mono_Core_Processor
6:     cpu ← searchCpuByName(cModel, sched.host.name)        ▷ Returns a Mono_Core_Processor with
   matching name, if found
7:     if cpu = null then
8:         cpu.name ← sched.host.name
9:         cpu.scheduling ← sched_params
10:        Add cpu to cModel
11:    end if
12:    return cpu
13: end function
```

The parameters of the scheduler are processed and a scheduling parameter entity in Cheddar is created. A monocore processor is then created and the scheduling parameters of the processor are set.

D.6 PARTITION PROCESSING

Algorithm D.6 is the function that processes a thread or shared resource implementation to determine their partition, and thus the address space to create in Cheddar. After doing some consistency checks, the function returns the created address space.

Algorithm D.7 ProcessPartition Function

```
1: function ProcessPartition(elt : <ThreadImplementation or SharedResourceImplementation>, cpu :
   Generic_Processor)
2:     part ← Partition containing elt
3:     addr : Address_Space
4:     addr.name ← part.name
5:     addr.cpu_name ← cpu.name
6:     for each _addr in cModel do
7:         if _addr.name = addr.name then
8:             if _addr.cpu_name ≠ addr.name then
9:                 Raise Exception
10:            else
11:                return _addr
12:            end if
13:        end if
14:    end for
15:    Add addr to cModel
16:    return addr
17: end function
```

The main operation of importance in this function is the verification at line 8. When an entity (thread or shared resource implementation) allocated on a processor is processed, the function verifies that a Cheddar address space for the partition, containing the entity, does not already exists. If the address space already exists, it was already allocated on a processor. The processor of the existing address space and the processor of the entity must match.

D.7 RESOURCE ACQUISITION PROCESSING

Algorithm D.8 is the procedure that processes a resource acquisition step of a step for which a task was created.

Algorithm D.8 ProcessResAcq Procedure

```
1: procedure PROCESSRESACQ(resAcq : ResourceAcquisitionStep, tsk : Periodic_Task)
2:     critSec : Critical_Section
3:     critSec.start ← resAcq.acquisitionTime
4:     critSec.end ← resAcq.acquisitionTime + lockingTime
5:     critSec.task_name ← tsk.name
6:     res ← processSharedResource(resAcq.resource)  ▷ Creates, adds, and returns a Cheddar resource with
   filled parameters
7:     Add critSec to res.critical_sections
8: end procedure
```

The procedure sets start and end parameters of a Cheddar critical section according to attributes acquisitionTime and lockingTime of the resource acquisition step. The task_name parameter of the critical section is set by the tsk parameter fed to the function. The tsk parameter is the task that was created for the step that has a resource acquisition step. Afterwards the critical section is added to the shared resource which may need to be processed beforehand (line 6).

D.8 SHARED RESOURCE PROCESSING

Algorithm D.9 is the function that processes a shared resource and creates the corresponding resource entity in the Cheddar model.

Algorithm D.9 ProcessSharedResource Function

```
1: function PROCESSSHAREDRESOURCE(sr : SharedResource)
2:     cpu ← processScheduler(sr.host)
3:     addr ← processPartition(sr, cpu)          ▷ These two steps are done before anything else because their
   functions checks that the cpu/addr pair is consistent with existing addrs in cModel
4:     for each _res in cModel.resources do
5:         if _res.name = sr.name then
6:             return _res
7:         end if
8:     end for
9:     res : Generic_Resource
10:    res.name ← sr.name
11:    res.protocol ← sr.protocol
12:    res.address_space_name ← addr.name
13:    res.cpu_name ← cpu.name
14:    Add res to cModel.resources
15:    return res
16: end function
```

Since a shared resource is allocated on a processor (through the scheduler) and contained by a partition, these entities are first processed to create their Cheddar entities (lines 2 and 3). If the processors and partitions were already processed, the call to their processing function is still important for consistency checks. Afterwards the shared resource's parameters are set and its Cheddar entity is created.

BIBLIOGRAPHY

[1] S. Anssi, S. Tucci-Pergiovanni, C. Mraidha, A. Albinet, F. Terrier, and S. Gérard. Completing EAST-ADL2 with MARTE for enabling scheduling analysis for automotive applications. In *Proceedings of the 6th European Congress Embedded Real Time Software*, Toulouse, France, 2010.

[2] T. Arpinen, E. Salminen, T. D. Hämäläinen, and M. Hännikäinen. MARTE profile extension for modeling dynamic power management of embedded systems. *Journal of Systems Architecture*, 58(5):209–219, Apr, 2012.

[3] N. Audsley, A. Burns, R. Davis, K. Tindell, and A. Wellings. Real-time system scheduling. In *Predictably Dependable Computing Systems*, pages 41–52. Springer Berlin Heidelberg, 1995.

[4] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284–292, Sep, 1993.

[5] N. Audsley, K. Tindell, and A. Burns. The end of the line for static cyclic scheduling? In *Proceedings of the 5th Euromicro Workshop on Real-Time Systems*, Oulu, Finland, 1993.

[6] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *Proceedings of Real-Time Systems Symposium 2011*, 2001.

[7] T. Baker. Multiprocessor EDF and deadline-monotonic schedulability analysis. In *Proceedings of the 24th IEEE Real-Time Systems Symposium*, Cancun, Mexico, 2003.

[8] A. Barbalace, A. Luchetta, G. Manduchi, M. Moro, A. Soppelsa, and C. Taliercio. Performance comparison of VxWorks, linux, RTAI and xenomai in a hard real-time application. In *Proceedings of the 15th IEEE-NPSS Real-Time Conference*, Baravia, IL, 2007.

[9] S. Baruah. The non-cyclic recurring real-time task model. In *Proceedings of the 31st Real-Time Systems Symposium*, San Diego, USA, 2010.

[10] S. Baruah. Dynamic- and static-priority scheduling of recurring real-time tasks. *Real-Time Systems*, 24(1):93–128, Jan, 2003.

[11] S. Baruah and E. Bini. Partitioned scheduling of sporadic task systems: an ILP-based approach. In *Proceedings of the 2008 Conference on Design and Architectures for Signal and Image Processing*, Brussels, Belgium, 2008.

[12] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, L. Stougie, and A. Wiese. A generalized parallel task model for recurrent real-time processes. In *Proceedings of the 33rd Real-Time Systems Symposium*, San Juan, Puerto Rico, 2012.

[13] S. Baruah, A. Burns, and R. Davis. Response-time analysis for mixed criticality systems. In *Proceedings of the 32nd Real-Time Systems Symposium*, 2011.

[14] S. Baruah, D. Chen, S. Gorinsky, and A. Mok. Generalized multiframe tasks. *Real-Time Systems*, 17(1):5–22, Jul, 1999.

[15] S. Baruah and J. Goossens. Scheduling real-time tasks: algorithms and complexity. In *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Chapman Hall/CRC Press, 2004.

[16] S. Baruah, A. Mok, and L. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the 11th Real-Time Systems Symposium*, Lake Buena Vista, USA, 1990.

[17] S. Baruah, L. Rosier, and R. Howell. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems*, 2(4):301–324, Nov, 1990.

[18] G. Behrmann, A. David, and K. G. Larsen. A tutorial on uppaal. In *Proceedings of the 2004 International School on Formal Methods for the Design of Computer, Communication, and Software Systems*, Bertinora, Italy, 2004.

[19] J. Bengtsson and Y. Wang. Timed automata: Semantics, algorithms and tools. *Lecture Notes on Concurrency and Petri Nets*, 3098:87–124, 2004.

[20] B. Berthomieu, J. P. Bodeveix, S. D. Zilio, P. Dissaux, M. Filali, P. Gaufillet, S. Heim, and F. Vernadat. Formal verification of aadl models with fiacre and tina. In *Proceedings of the 5th International Congress and Exhibition on Embedded Real-Time Software and Systems*, Toulouse, France, 2010.

[21] E. Bini and G. Buttazzo. Schedulability analysis of periodic fixed priority systems. *IEEE Transactions on Computers*, 53(11):1462–1473, Nov, 2004.

[22] B. Blanchard and W. Fabrycky. *Systems engineering and analysis*, volume 4. Prentice Hall, Englewood Cliffs, NJ, 1990.

[23] R. Boissier, E. Gressier-Soudan, A. Laurent, and L. Seinturier. Enhancing numerical controllers, using MMS concepts and a CORBA-based software bus. *International Journal of Computer Integrated Manufacturing*, 14(6):260–269, 2001.

[24] V. Bonifaci, A. Marchetti-Spaccamela, S. Stiller, and A. Wiese. Feasibility analysis in the sporadic DAG task model. In *Proceedings of the 25th Euromicro Conference on Real-Time Systems*, Paris, France, 2013.

[25] E. Borde, G. Haïk, and L. Pautet. Model-based reconfiguration of critical software component architectures. In *Proceedings of the 2009 Design, Automation & Test in Europe Conference*, Nice, France, 2009.

[26] M. Bourdellès, S. Li, I. Quadri, E. Brosse, E. Gaudin, F. Mallet, A. Goknil, A. Sadovykh, D. George, and J. Kreku. Fostering analysis from industrial embedded systems modeling. In *Industry and Research Perspectives on Embedded System Design*. IGI Global, 2014.

[27] M. Bourdellès and S. Li. PRESTO project, automatic code instrumentation from (modelio based) models to process performance analysis at the software integration phase. Paris, France, 2013.

[28] A. Burns and A. Wellings. *Real-time systems and programming languages: Ada 95, real-time Java, and real-time POSIX*. Pearson Education, 2001.

[29] G. Buttazzo. Rate monotonic vs. EDF: Judgment day. *Real-Time Systems*, 29(1):5–26, Jan, 2005.

[30] T. Chan. Time-division multiple access. In *Handbook of Computer Networks*, pages 769–778. John Wiley & Sons, Hoboken, 2011.

[31] M.-I. Chen and K.-J. Lin. Dynamic priority ceilings: A concurrency control protocol for real-time systems. *Real-Time Systems*, 2(4):325–346, Nov, 1990.

[32] A. M. Cheng. *Real-time systems: scheduling, analysis, and verification*. John Wiley & Sons, New York, 2003.

[33] M. Chéramy, P.-E. Hladik, and A.-M. D'eplanche. SimSo: A simulation tool to evaluate real-time multiprocessor scheduling algorithms. In *Proceedings of the 5th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*, Madrid, Spain, 2014.

[34] H. Chetto, M. Silly, and T. Bouchentouf. Dynamic scheduling of real-time tasks under precedence constraints. *Real-Time Systems*, 2(3):181–194, 1990.

[35] H. Cho, B. Ravindran, and E. Jensen. An optimal real-time scheduling algorithm for multiprocessors. In *Proceedings of the 27th IEEE International Real-Time Systems Symposium*, Rio de Janeiro, Brazil, 2006.

[36] A. Choquet-Geniet and E. Grolleau. Minimal schedulability interval for real-time systems of periodic tasks with offsets. *Theoretical Computer Science*, 310:117–134, Jan, 2004.

[37] D. Clarke, I. Lee, and H. L. Xie. VERSA: a tool for the specification and analysis of resource-bound real-time systems. Technical report, University of Pennsylvania, 1993.

[38] M. Coutinho, J. Rufino, and C. Almeida. Response time analysis of asynchronous periodic and sporadic tasks sheduled by a fixed priority preemptive algorithm. In *Proceedings of the 20th Euromicro Conference on Real-Time Systems*, Prague, Czech Republic, 2008.

[39] J. Davidson. On the architecture of secure software defined radios. In *Proceedings of the 2008 IEEE Military Communications Conference*, San Diego, USA, 2008.

[40] R. Davis and A. Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Computing Surveys (CSUR)*, 43(4):35, 2011.

[41] V. Debruyne, F. Simonot-Lion, and Y. Trinquet. EAST-ADL - an architecture description language. In *Architecture Description Languages*, volume 176, pages 181–195. Springer-Verlag, New York, USA, 2005.

[42] S. Demathieu, F. Thomas, C. André, S. Gérard, and F. Terrier. First experiments using the UML profile for MARTE. In *Proceedings of the 11th IEEE International Symposium on Object Oriented Real-Time Dsitributed Computing*, Orlando, USA, 2008.

[43] M. Dertouzos and A. Mok. Multiprocessor online scheduling of hard-real-time tasks. *IEEE Transactions on Software Engineering*, 15(12):1497–1506, Dec, 989.

[44] U. Díaz-de Cerio, J. P. Uribe, M. G. Harbour, and J. Palencia. Adding precedence relations to the response-time analysis of edf distributed real-time systems. In *Proceedings of the 22nd International Conference on Real-Time Networks and Systems*, Versailles, France, 2014.

[45] P. Dissaux, O. Marc, S. Rubini, C. Fotsing, V. Gaudel, F. Singhoff, A. Plantec, V. Nguyen Hong, and H. N. Nam Tran. The SMART project: Multi-agent scheduling simulation of real-time architectures. In *Proceedings of the 7th European Congress on Embedded Real Time Software and Systems*, Toulouse, France, 2014.

[46] P. Dissaux and F. Singhoff. Stood and cheddar : AADL as a pivot language for analysing performances of real time architectures. In *Proceedings of the 4th European Congress on Embedded Real Time Software and System*, Toulouse, France, Jan, 2008.

[47] P. Ekberg, N. Guan, M. Stigge, and W. Yi. An optimal resource sharing protocol for generalized multiframe tasks. In *Nordic Workshop on Programming Theory*, Vasteras, Sweden, 2011.

[48] E. Enoiu, R. Marinescu, C. Seceleanu, and P. Pettersson. ViTAL : A verification tool for EAST-ADL models using UPPAAL PORT. In *Proceedings of the 17th International Conference on Engineering of Complex Computer Systems*, Paris, France, 2012.

[49] P. H. Feiler, D. P. Gluch, and J. J. Hudak. The architecture analysis & design language (AADL): an introduction. Technical report, Software Engineering Institute, Pittsburgh, 2006.

[50] J. Forget, F. Boniol, E. Grolleau, D. Lesens, and C. Pagetti. Scheduling dependent periodic tasks without synchronization mechanisms. In *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, Stockholm, Sweden, 2010.

[51] M. Forget, E. Grolleau, C. Pagetti, and P. Richard. Dynamic priority scheduling of periodic tasks with extended precedences. In *Proceedings of the 16th Conference on Emerging Technologies & Factory Automation*, Toulouse, France, 2011.

[52] J. Garcia, J. Gutierrez, and M. Harbour. Schedulability analysis of distributed hard real-time systems with multiple-event synchronization. In *Proceedings of the 12th Euromicro Conference on Real-Time Systems*, Stockholm, Sweden, 2000.

[53] G. Gardey, D. Lime, M. Magnin, and O. Roux. Romeo: A tool for analyzing time petri nets. In *Proceedings of the 17th International Conference on Computer Aided Verification*, Edinburgh, UK, 2005.

[54] D. Garlan, R. Monroe, and D. Wile. ACME: an architecture description interchange language. In *Proceedings of 20th Conference of the Centre for Advanced Studies on Collaborative Research*, Toronto, Canada, 2010.

[55] S. Giordano. Mobile ad hoc networks. In *Wiley Series on Parallel and Distributed Computing*, pages 325–346. John Wiley & Sons, New York, 2002.

[56] J. Goossens and R. Devilliers. The non-optimality of the monotonic priority assignments for hard real-time offset free systems. *Real-Time Systems*, 13(2):107–126, Sept, 1997.

[57] M. Harbour, J. Gutierrez, J. Drake, P. Martinez, and J. Palencia. Modeling distributed real-time systems with MAST 2. *Journal of Systems Architecture*, 59(6):331–340, Jun, 2013.

[58] R. Henia and R. Ernst. Improved offset-analysis using multiple timing-references. In *Proceedings of the 2006 Conference on Design Automation and Test in Europe*, Munich, Germany, 2006.

[59] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst. System level performance analysis - the SymTA/s approach. *IEE Proceedings - Computers and Digital Techniques*, 152(2):148, Mar, 2005.

[60] R. Henia, L. Rioux, and V. Thomas. Industrial adaptation of MARTE for early scheduling analysis of component-based applications. In *Proceedings of the 4th International Workshop on Nonfunctional System Properties in Domain Specific Modeling Languages*, Valencia, Spain, 2012.

[61] T.-P. S. Inc. Tri-pacific software inc. : RAPID-RMA, 2014. `http://www.tripac.com/rapid-rma`.

[62] M. Z. Iqbal, S. Ali, T. Yue, and L. Briand. Applying UML/MARTE on industrial projects: challenges, experiences, and guidelines. *Software & Systems Modeling*, pages 1–19, Mar, 2014.

[63] ISO/IEC/IEEE. Systems and software engineering - architecture description. Technical Report 42010:2011, International Organization for Standardization, Internation Electronic Comission, Institute of Electrical and Electronics Engineers, 2001.

[64] M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, 1986.

[65] R. Kaiser and S. Wagner. Evolution of the pikeos microkernel. In *Proceedings of the 1st International Workshop on Microkernels for Embedded Systems*, Kensington, Australia, 2007.

[66] J. Kany and S. Madsen. Design optimisation of fault-tolerant event-triggered embedded systems. Master's thesis, Technical University of Denmark, Lyngby, Denmark, 2007.

[67] S. Kelly and J.-P. Tolvanen. *Domain-Specific Modeling*. John Wiley & Sons, 2008.

[68] S. Klaus. Finding a minimal transitive reduction in a strongly conncected digraph within linear time. In *Proceedings of the 15th International Workshop on Graph-Theoretic Concepts in Computer Science*, volume 411, Berlin, Germany, 1990.

[69] K. Klobedanz, C. Kuznik, A. Thuy, and W. Mueller. Timing modeling and analysis for AUTOSAR-based software development: a case study. In *Proceedings of the 2010 Conference on Design, Automation and Test in Europe*, Dresden, Germany, 2010.

[70] H. Kopetz. Event-triggered versus time-triggered real-time systems. In *Operating Systems of the 90s and Beyond*, volume 563 of *Lecture Notes in Computer Science*, pages 86–101. Springer Berlin Heidelberg, 1991.

[71] H. Kopetz. The time-triggered model of computation. In *Proceedings of the 19th IEEE Real-Time Systems Symposium*, Madrid, Spain, 1998.

[72] H. Kopetz, A. Damm, C. Koza, M. Mulazzani, W. Schwabl, C. Senft, and R. Zainlinger. Distributed fault-tolerant real-time systems: the mars approach. *IEEE Micro*, 9(1):25–40, Feb, 1989.

[73] D. Kum, G.-M. Park, S. Lee, and W. Jung. AUTOSAR migration from existing automotive software. In *Proceedings of the 2008 Internationl Conference on Control, Automation and Systems*, Seoul, South Korea, 2008.

[74] I. Lee, J. Y. Leung, and S. H. Son. *Handbook of real-time and embedded systems*. CRC Press, 2007.

[75] J. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Procedings of the 11th Real-Time Systems Symposium*, pages 201–209, Lake Buena Vista, USA, 1990.

[76] J. Y.-T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2(4):237–250, Dec, 1982.

[77] S. Li. PRESTO: Results from execution trace analysis. In *Joint Proceedings of Co-located Events at the 8th European Conference on Modelling Foundations and Applications (ECMFA 2012)*, Copenhagen, Denmark, 2012.

[78] S. Li, M. Bourdellès, A. Acebedo, J. Botella, and F. Peureux. Experiment on using model-based testing for automatic tests generation on a software radio protocol. In *Proceedings of the 9th International Workshop on Systems Testing and Validation*, pages 79–84, Paris, France, 2012.

[79] S. Li and F. Broekaert. Low-power scheduling with DVFS for common RTOS on multicore platforms. In *Proceedings of the 3rd Workshop on Embedded Operating Systems*, Toulouse, France, 2013.

[80] S. Li, S. Rubini, F. Singhoff, and M. Bourdellès. Applying holistic schedulability tests to industrial systems: Experience and lessons learned. In *Proceedings of the 5th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*, Madrid, Spain, 2014.

[81] S. Li, S. Rubini, F. Singhoff, and M. Bourdellès. A task model for TDMA communications. In *Proceedings of the 9th IEEE International Symposium on Industrial Embedded Systems Work in Progress*, Pisa, Italy, 2014.

[82] S. Li, F. Singhoff, S. Rubini, and M. Bourdellès. Applicability of real-time schedulability analysis on a software radio protocol. *ACM SIGAda Ada Letters*, 32(3):81–94, 2012-12.

[83] S. Li, F. Singhoff, S. Rubini, and M. Bourdellès. Extending schedulability tests of tree-shaped transactions for TDMA radio protocols. In *Proceedings of the 19th IEEE International Conference on Emerging Technology & Factory Automation*, Barcelona, Spain, 2014.

[84] S. Li, F. Singhoff, S. Rubini, and M. Bourdellès. Scheduling analysis of TDMA-constrained tasks: Illustration with software radio protocols. In *Proceedings of the 11th IEEE International Conference on Embedded Software and Systems*, Paris, France, 2014.

[85] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, Jan, 1973.

[86] J. Lopez, L. Diaz, and D. Garcia. Utilization bounds for EDF scheduling on real-time multiprocessor systems. *Real-Time Systems*, 28(1):39–68, Oct, 2004.

[87] Y. Ma, H. Yu, T. Gautier, P. L. Guernic, J.-P. Talpin, L. Besnard, and M. Heitz. Toward polychronous analysis and validation for timed software architectures in aadl. In *Proceedings of the 2013 Design, Automation, and Test in Europe Conference*, Grenoble, France, 2013.

[88] E. Maes and N. Vienne. MARTE to cheddar transformation using ATL. Technical report, Thales Research & Technologies, 2007.

[89] J. Mäki-Turja and M. Nolin. Faster response time analysis of tasks with offsets. In *Procedings of the 10th IEEE Real-Time Technology and Applications Symposium*, 2004.

[90] J. Mäki-Turja and M. Nolin. Tighter response time analysis of tasks with offsets. In *Proceedings of the 10th International Conference on Real-Time Computing and Applications*, Gotherburg, Sweden, 2004.

[91] J. Mäki-Turja and M. Nolin. Fast and tight response-times for tasks with ofsets. In *Proceedings of the 17th Euromicro Conference on Real-Time Systems*, Palma de Mallorca, Spain, 2005.

[92] J. Mäki-Turja and M. Nolin. Efficient implementation of tight response-times for tasks with offsets. *Real-Time Systems*, 40(1):77–116, Feb, 2008.

[93] J. Maki-Turja and M. Sjodin. Response-time analysis for transactions with execution-time dependencies. In *Proceedings of the 19th International Conference on Real-Time Networks and Systems*, Nantes, France, 2011.

[94] N. Malcolm and W. Zhao. The timed-token protocol for real-time communications. *Computer*, 27(1):35–41, Jan, 1994.

[95] S. O. Marinescu, D. Tamas-Selicean, V. Acretoaie, and P. Pop. Timing analysis of mixed-criticality hard real-time applications implemented on distributed partitioned architectures. In *Proceedings of the 17th International Conference on Emerging Technologies & Factory Automation*, Krakow, Poland, 2012.

[96] J. Medina and l. G. Cuesta. From composable design models to schedulability analysis with UML and the UML profile for MARTE. In *Proceedings of the 3rd Workshop on Compositional Theory and Technology for Real-Time Embedded Systems*, San Diego, USA, 2010.

[97] N. Medvidovic and R. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering*, 26(1):70–93, 2000.

[98] T. Mens and P. Van Gorp. A taxonomy of model transformation. *Electronic Notes in Theoretical Computer Science*, 152:125–142, 2006.

[99] J. Mitola. The software radio architecture. *IEEE Communications Magazine*, 33(5):26–38, 1995.

[100] A. Mok and D. Chen. A multiframe model for real-time tasks. *IEEE Transactions on Software Engineering*, 23(10):635–645, 1997.

[101] S. Mubeen, J. Mäki-Turja, and M. Sjödin. Implementation of holistic response-time analysis in rubus-ICE: preliminary findings, issues and experiences. In *Proc. 32nd IEEE Real-Time Syst. Symp., WIP Session*, Vienna, Austria, 2011.

[102] D. Oh and T. Baker. Utilization bounds for n-processor rate monotone scheduling with static priority assignment. *Real-Time Systems*, 15(2):183–192, Sept, 1998.

[103] OMG. UML. Specification 2.4.1, OMG, Aug, 2011. `http://www.omg.org/spec/UML`.

[104] E. Oracle. Papyrus website. `http://www.eclipse.org/papyrus/`.

[105] J. Palencia and M. Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *Proceedings of the 19th IEEE Real-Time Systems Symposium*, Madrid, Spain, 1998.

[106] J. Palencia and M. Harbour. Exploiting precedence relations in the schedulability analysis of distributed real-time systems. In *Proceedings of the 20th IEEE Real-Time Systems Symposium*, pages 328–339, Phoenix, AZ, 1999.

[107] J. Palencia and M. Harbour. Offset-based response time analysis of distributed systems scheduled under EDF. In *Proceedings of the 15th Euromicro Conference on Real-Time Systems*, Porto, Portugal, 2003.

[108] R. Pellizzoni and G. Lipari. A new sufficient feasibility test for asynchronous real-time periodic task sets. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems*, Catania, Italy, 2004.

[109] R. Pellizzoni and G. Lipari. Feasibility analysis of real-time periodic tasks with offsets. *Real-Time Systems*, 30(1):105–128, May, 2005.

[110] J. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice Hall, New Jersey, USA, 1981.

[111] A. Plantec and F. Singhoff. Refactoring of an ada 95 library with a meta CASE tool. *ACM SIGAda Ada Letters*, 26(3):61–70, Nov, 2006.

[112] PrismTech. Spectra CX website. `http://www.prismtech.com/spectra/products/spectra-cx`.

[113] I. Quadri and S. Li. PRESTO: Improvements of industrial real-time embedded systems design and development. In *OMG Workshop on Real-Time, Embedded and Entreprise-Scale Time-Critical Systems*, Paris, France, 2012.

[114] I. R. Quadri, A. Sadovykh, and L. S. Indrusiak. MADES: A SysML/MARTE high level methodology for real-time and embedded systems. In *Proceedings of the 10th Embedded Realtime Software and Systems Conference*, Toulouse, France, 2012.

[115] T. Qureshi, D. J. Chen, M. Persson, and T. Martin. On integrating EAST-ADL and UPPAAL for embedded system architecture verification. *Embedded Systems Development*, 20:85–99, 2014.

[116] A. Rahni. *Contributions à la validation d'ordonnancement temps réel en présence de transactions sous priorités fixes et EDF*. PhD thesis, Université de Poitiers, Poitiers, France, 2008.

[117] R. Rajkumar. Real-time synchronization protocols for shared memory multiprocessors. In *Proceedings of the 10th International Conference on Distributed Computing Systems*, Paris, France, 1990.

[118] O. Redell. Analysis of tree-shaped transactions in distributed real time systems. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems*, pages 239–248, Catania, Italy, 2004.

[119] X. Renault, F. Kordon, and J. Hugues. Adapting models to model checkers, a case study : Analysing AADL using time or colored petri nets. In *Proceedings of the 20th IEEE International Workshop on Rapid System Prototyping*, Paris, France, 2009.

[120] J. Rufino, P. Verissimo, G. Arroz, C. Almeida, and L. Rodrigues. Fault-tolerant broadcasts in CAN. In *Proceedings of the 28th Annual Internation Symposium on Fault-Tolerant Computing*, Munich, Germany, 1998.

[121] J. Rumbaugh, I. Jacobson, and G. Brooch. *Unified Modeling Language Reference Manual*. Pearson Higher Education, 2004.

[122] D. Schmidt. Guest editor's introduction: Model-Driven engineering. *Computer*, 39(2):25–31, 2006.

[123] D. C. Schmidt. Evaluating architectures for multithreaded object request brokers. *Communications of the ACM*, 41(10):54–60, 1998.

[124] SEI Carnegie Mellon University. OSATE website. `http://www.aadl.info/aadl/currentsite/tool/osate.html`.

[125] S. Sendall and W. Kozaczynski. Model transformation: the heart and soul of model-driven software development. *IEEE Software*, 20(5):42–45, Sep, 2003.

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

[127] L. Sha, R. Rajkumar, and J. Lehoczky. Priority inheritance protocols: an approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, 1990.

[128] F. Singhoff and A. Plantec. Towards user-level extensibility of an ada library: an experiment with cheddar. In *Proceedings of the 12th International Conference on Reliable Software Technologies*, Berlin, Germany, 2007.

[129] F. Singhoff, A. Plantec, P. Dissaux, and J. Legrand. Investigating the usability of real-time scheduling theory with the cheddar project. *Real-Time Systems*, 43(3):259–295, Jun, 2009.

[130] Softeam. Modelio website. `http://www.modelio.org/`.

[131] I. Software. Rational Software Architect website. `http://www-03.ibm.com/software/products/en/ratisoftarch`.

[132] O. Sokolsky, I. Lee, and D. Clarke. Schedulability analysis of aadl models. In *Proceedings of the 20th International Parallel and Distributed Processing Symposium*, Rhodes Island, Greece, 2006.

[133] B. Sprunt, L. Sha, and J. Lehoczky. Aperiodic task scheduling for hard-real-time systems. *Real-Time Systems*, 1(1):27–60, Jun, 1989.

[134] M. Spuri. Analysis of deadline scheduled real-time systems. Technical report, INRIA Rocquencourt, 1996.

[135] J. Stankovic. Misconceptions about real-time computing: A serious problem for next-generation systems. *IEEE Computer*, 21(10):10–19, Oct, 1988.

[136] M. Stigge, P. Ekberg, N. Guan, and W. Yi. The digraph real-time task model. In *Proceedings of the 17th IEEE Real-Time and Embedded Technology and Applications Symposium*, Chicago, USA, 2011.

[137] J. Strosnider, J. Lehoczky, and L. Sha. The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments. *IEEE Transactions on Computers*, 44(1):73–91, Jan, 1995.

[138] H. Takada and K. Sakamura. Schedulability of generalized multiframe task sets under static priority assignment. In *Proceedings of the 4th International Workshop on Real-Time Computing Systems and Applications*, pages 80–86, Taipei, Taiwan, 1997.

[139] A. Tanenbaum. *Modern Operating Systems*. Prentice Hall Press, Upper Saddle River, USA, 3rd edition, 2007.

[140] N. Tchidjo Moyo, E. Nicollet, F. Lafaye, and C. Moy. On schedulability analysis of non-cyclic generalized multiframe tasks. In *Proceedings of the 2010 22nd Euromicro Conference on Real-Time Systems*, pages 271–278, 2010.

[141] K. Tindell. Adding time-offsets to schedulability analysis. Technical report, University of York, 1994.

[142] K. Tindell, A. Burns, and A. Wellings. Calculating controller area network (CAN) message response times. *Control Engineering Practice*, 3(6):1163–1169, Aug, 1995.

[143] K. Tindell and J. Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and Microprogramming*, 40(2-3):117–134, Apr, 1994.

[144] J.-P. Tolvanen and S. Kelly. MetaEdit+: defining and using integrated domain-specific modeling languages. In *Proceedings of the 24th Conference Companion on Object Oriented Programming, Systems, Languages, and Applications*, Orlando, Florida, 2009.

[145] R. Urunuela, A. Déplanche, and Y. Trinquet. STORM a simulation tool for real-time multiprocessor scheduling evaluation. In *Proc. 2010 IEEE Conf. Emerging Technologies and Factory Automation*, pages 1–8, Bilbao, Spain, 2010.

[146] S. Vinoski. CORBA: integrating diverse applications within distributed heterogeneous environments. *IEEE Communications Magazine*, 35(2):46–55, Feb, 1997.

[147] E. Wandeler and L. Thiele. Optimal TDMA time slot and cycle length allocation for hard real-time systems. In *Proceedings of the 11th Asia and South Pacific Design Automation Conference*, Yokohama, Japan, 2006.

[148] N. Wang, D. C. Schmidt, and C. O'Ryan. An overview of the CORBA component model. In *Comopnent-Based Software Engineering: Putting the Pieces Together*. Addison-Wesley, 2001.

[149] R. Wilhelm, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, P. Stenström, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, and R. Heckmann. The worst-case execution-time problem - overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems*, 7(3):1–53, 2008.

[150] J. Xu and D. Parnas. On satisfying timing constraints in hard-real-time systems. *IEEE Transactions on Software Engineering*, 19(1):70–84, Jan, 1993.

[151] O. Yassine. *Model-based Framework for Using Advanced Scheduling Theory in Real-Time Systems Design*. PhD thesis, Ecole Nationale Sup'erieure de M'ecanique et d'A'erotechnique, Poitiers, France, 2013.

[152] H. Zimmermann. OSI reference model–the ISO model of architecture for open systems interconnection. *IEEE Transactions on Communications*, 28(4):425–432, 1980.