Scheduling analysis of tasks constrained by TDMA: Application to software radio protocols

Shuai Li, Frank Singhoff, Stéphane Rubini, Michel Bourdellès

Scheduling Analysis of Tasks Constrained by TDMA: Application to Software Radio Protocols
Author's version

Shuai Li+*, Frank Singhoff*, Stéphane Rubini*, Michel Bourdellès+
*Lab-STICC UMR 6285, Université de Bretagne Occidentale, France, {last-name}@univ-brest.fr
+Thales Communications & Security, France, {first-name}.{last-name}@thalesgroup.com

July 20, 2017

Abstract:

The work, presented in this article, aims at performing scheduling analysis of Time Division Multiple Access (TDMA) based communication systems. Products called software radio protocols, developed at Thales Communications & Security, are used as case-studies.

Such systems 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 configuration. 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.

We propose a task model called Dependent General Multiframe (DGMF) to capture the specificities of such a system for scheduling analysis. The DGMF task model describes, in particular, the different jobs of a task, and task dependencies. To analyze DGMF tasks, we propose their transformation to real-time transactions and we also propose a new schedulability test for such transactions.

The general analysis method is implemented in a tool that can be used by software engineers and architects. Experimental results show that our proposi-

tions 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.

These results are important for engineering work, in order to limit the over-dimensioning of resources. Through tooling, we also automated the analysis. These are advantages for engineers in the more and more competitive market of software radios.

Keywords: Real-time Embedded System, Scheduling Analysis, Task Dependency, Software Radio Protocol, TDMA, General Multiframe, Transaction

1 Introduction

In this article, we propose a scheduling analysis method for communication systems that use the Time Division Multiple Access (TDMA) channel access method. Our method is applied to real industrial systems, developed at Thales Communications & Security, called software radio protocols.

1.1 Context

Radios are communication systems. For communication to be established, some radio protocols need to be defined and implemented. Traditionally radios are implemented as dedicated hardware components. Trends in the last two decades have seen the emergence of software radios [1], 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.

Software radio protocols are typical real-time systems. In such systems, activities are implemented by software tasks that are time-constrained. The tasks may also have precedence dependencies and use shared resources. In this article, we consider tasks that are scheduled by a partitioned Fixed Priority (FP) preemptive scheduling policy.

The particularity of a software radio protocol real-time system is that it may be impacted by TDMA, a common method to access the shared communication medium between several radio stations. In TDMA [2], time is divided into several time slots and at each time slot, a radio either transmits its data, or receives some data. This method impacts the software of the system, because the tasks have an execution time, deadline, and release time that depend on the time slots. TDMA is also one reason, among others, that there are hard deadlines in such systems. We must thus analyze the scheduling of these systems.

Scheduling analysis [3, 4] is the method used to verify that tasks will meet their deadline, when they are scheduled on processors. Scheduling analysis can potentially be complex, due to the dependencies between the tasks for example. Some methods, part of scheduling analysis, are schedulability tests [5]. 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 Worst Case Response Time (WCRT) of

tasks. The WCRT of a task is compared to its deadline to assess its schedulability.

Schedulability tests are associated with task models that abstract the architecture of a real-time system for the analysis. There are several task models in the literature that consider more or less dependencies between tasks, and characteristics of the execution platform that hosts the tasks.

1.2 Overview of Contributions

As we will see, current task models and schedulability tests are not applicable to a software radio protocol that uses TDMA to access the shared communication medium. For example some task models may consider the impact of TDMA on tasks, but they do not consider task dependencies. Therefore, in this article we propose a scheduling analysis method adapted for the characteristics of real software radio systems developed at Thales. Our analysis method is based on several of our contributions. The next paragraphs describe our contributions and give an overview of our approach.

Extension of the General Multiframe Task Model This article proposes the Dependent General Multiframe (DGMF) task model, an extension of the General Multiframe (GMF) task model [6]. 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. The DGMF task model extends GMF with precedence dependencies and shared resources. Contrary to GMF, the DGMF task model can also be applicable to a multiprocessor system with partitioned scheduling.

DGMF to Transaction The analysis method for DGMF consists in exploiting another kind of model called transactions [7]. A transaction is a group of tasks related by precedence dependency. This article proposes an algorithm to transform DGMF tasks to transactions. The transformation solves the issue of the semantic gap between the two models.

Extension of Schedulability Test for Transactions The transactions that are the results of the transformation have what we call non-immediate tasks, i.e. tasks which are not necessarily released immediately by their predecessor. Thus this article proposes the WCDSOP+NIM test, which is an extension of the WCDOPS+ test in [8]. Our test is used to assess schedulability of transactions with non-immediate tasks. Thanks to WCDSOP+NIM, we are able to perform scheduling analysis on transactions produced from DGMF models, and thus to perform scheduling analysis on DGMF models themselves.

Available Tool The GMF, DGMF, transaction models, and their analysis methods, are all implemented in Cheddar [9]. Cheddar is an open-source

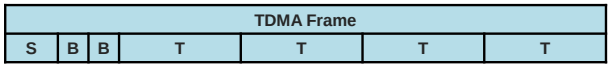

Figure 1: TDMA Frame

scheduling analysis tool, available for both the research and industrial communities.

1.3 Article Organization

The rest of this article is divided into five sections that are organized as follows. In Section 2, the TDMA-based software radio protocol system is presented. This section defines the assumptions of our work. In Section 3, we discuss the applicability of task models related to our work. In Section 4, the DGMF task model, and its analysis method, and experiments (both simulation results and case-study results) are exposed. Section 5 is on our WCDOPS+NIM schedulability test. Again, experiments on our test are shown in the same section, with both simulation results and case-study results. Finally, we conclude in Section 6 by discussing some future works.

2 Software Radio Protocol

In the following sections, we first present the concept of a radio protocol and the impact of TDMA on the protocol. Afterwards the software and execution platform of our system is presented, and we relate the architecture of the system to two typical architecture designs. Finally, we summarize the requirements to consider for our scheduling analysis work.

2.1 Radio Protocols Based On Time-Division Multiplexing

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 exchanged between users, but also how communication is synchronized between all users. Several protocols can cooperate within a same radio to form the protocol stack [10].

The functionalities of a radio protocol may be impacted by the method used by the radio to access the communication medium. TDMA [2] 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 1 shows a typical frame.

Slots may be of different types. For example in Figure 1, the TDMA frame has three types of slot: *Service* (S) for synchronization between stations; *Beacon* (B) for observation/signaling the network; *Traffic* (T) for payload 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 1, 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. 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. 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.

When TDMA is used to access the communication medium, it impacts the functionalities of the radio protocol and it introduces time constraints. For example the operation to fetch data packets to be transmitted in a Tx slot is time-constrained. The TDMA method is a particular case of the timed-token protocol for real-time communications [11]. Indeed the same approach of dividing time into time slots, and allocating slots to entities, is used in both methods. Since these methods introduce time constraints, tasks that implement the system may also have time constraints.

The next section exposes the software and execution platform architecture of one possible implementation of a radio protocol.

2.2 Software and Execution Platform Architecture

The previous section showed how a radio protocol is designed. This section presents one possible implementation by Thales, called a software radio protocol [1]. The implementation is described through its software and execution platform architecture.

Traditionally functionalities of a radio protocol are implemented as dedicated hardware (e.g. ASIC, 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). The software entities are implemented by some entities in the execution platform: OS and hardware. Figure 2 shows an example of these entities.

Figure 2, shows several protocols (called RLC and MAC) implemented by *tasks* allocated on *processors*. The system is a partitioned multiprocessor system: tasks are allocated on a processor, and they do not migrate. On a processor, tasks are scheduled by a preemptive FP scheduling policy. The priorities of tasks are assigned arbitrarily according to domain expertise.

They may have precedence dependency (e.g. communication through semaphores signaling). They may also use shared resources. Shared resources are assumed local [12], 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 proto-

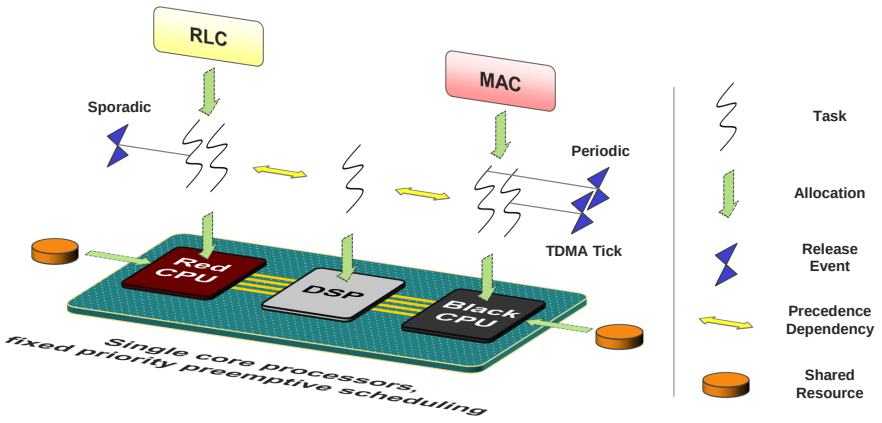

Figure 2: Architecture for Scheduling Analysis: Red CPU is a GPP handling ciphered data, Black CPU is a GPP handling deciphered data, DSP means Digital Signal Processor

col that prevents unbounded priority inversion for uniprocessor. Examples of such resource access protocols are PIP and PCP [13]. Their implementation can typically be found in execution platforms with the VxWorks OS. This OS is present in some products developed by Thales.

Some tasks may be released sporadically. For example some tasks 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 of a Medium Access Control (MAC) protocol [10]. In this article, activities of MAC are divided in time due to the usage of TDMA to access the communication medium. 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.

The software architecture of the system is conform to some architectures found in the literature. Indeed, two software architectures for RTES have been studied extensively: event-triggered [14] and time-triggered [15]. A software radio protocol is both a time-triggered and an event-triggered system. Indeed, some tasks are 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.

2.3 Requirements for Scheduling Analysis

From the description of its architecture in Section 2.2, characteristics of a software radio protocol, useful for scheduling analysis, are summarized in Table 1. In this article, it is expected that the proposed scheduling analysis method covers all of the characteristics in Table 1.

In the next section we study the applicability of scheduling analysis methods of the literature, when considering the requirements exposed in this section.

3 Applicability of Related Works

In this section, we discuss problems encountered when applying existing task models of the literature, to analyze the scheduling of a software radio protocol.

A task model is said applicable to a characteristic of the system, if it is possible to model the characteristic, and if there exists a schedulability test for the task model that considers the characteristic. Table 2 shows the applicability of some task models to each characteristic. These task models come from the literature and they are the fundamental periodic task model [3], the transaction model [7] that expresses precedence dependent tasks, the multiframe task models [6] that express jobs of a task as a vector, and the DAG task models [5] that express jobs of a task as a graph.

The applicability of the task models is discussed in the following paragraphs. Either the non-applicability of some task models on a characteristic is justified, or the applicability of all task models on a characteristic is shown.

TDMA release Seminal schedulability tests in [3, 16] for the fundamental periodic and sporadic task models assume a synchronous system, where tasks are all released at a unique critical instant. Tasks constrained by TDMA are not synchronous since they may be released at different pre-defined times.

The transaction model, that generalizes the fundamental periodic and sporadic task models, focuses on asynchronous releases of tasks. 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. In [7], Tindell analyzes asynchronous tasks 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 Tindell 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.

Some other schedulability tests for transactions in [17, 8] also consider that there are some periodic tasks, and some tasks released immediately by their predecessor, instead of pre-defined times. These results cannot be applied on the targeted systems of this article.

Table 1: Requirements for Scheduling Analysis

Name	**Description**
TDMA release	Tasks may be released at predefined times, corresponding to the 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.

Table 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

Finally some works have been done on systems that are both event-triggered and time-triggered, like a software radio protocol. In [18], 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 a sporadic event [3, 7, 6, 19].

Arbitrary deadline Some of the schedulability tests for the periodic and sporadic task models assume $D_i \leq T_i$ [3, 16]. Thus the deadline is constrained.

Schedulability tests [6, 20] for GMF assume that the *l-MAD* property or the *Frame Separation* property hold. The *l-MAD* property constrains the deadline of a frame to be less than, or equal to, the deadline of the next frame. The *Frame Separation* property constrains the deadline of a frame to be less than the release of the next frame. These properties thus constrain the deadline of frames and thus tasks.

Arbitrary priority There exists at least one schedulability test for each task model that supports this characteristic [16, 21, 20, 5].

Precedence dependency Analysis of the periodic and sporadic task models did not fully handle precedence dependencies, until they were generalized by the transaction model [21].

Some works [22, 23, 7, 24] 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 a Earliest-Deadline First (EDF) policy with dynamic priorities instead of fixed one. Therefore they

are not applicable due to the arbitrary priorities, arbitrary deadlines, and FP scheduling policy of a software radio protocol.

Other works [25] 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 task models assume independent tasks [6]. The DAG task models support precedence dependencies between jobs of a task. They do not explicitly handle precedence dependencies between the tasks themselves. For example the authors in [26] assume independent DAG tasks.

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 tasks [6, 26]. In [27] 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 [6, 5, 26], among those present in Table 2.

Preemptive FP policy on uniprocessor All task models have schedulability tests for the preemptive FP scheduling policy [3, 7, 6, 26].

Partitioned multiprocessor The periodic, sporadic task models, and the multiframe task models have schedulability tests for uniprocessor systems [3, 6]. The DAG task models has tests for uniprocessor systems [5], or global multiprocessor systems [26], i.e. where tasks may migrate to other processors. Again, these task models are not applicable to our system.

In conclusion, the results of Table 2 show that none of the task models support all characteristics of the software radio protocol to analyze. The following two sections expose our contributions to solve this problem: the DGMF task model, its transformation to a transaction model, and our extended schedulability test for transactions.

4 Dependent General Multiframe

To analyze the schedulability of a TDMA-based software radio protocol, this article proposes the Dependent General Multiframe (DGMF) task model. The following sections first define this task model. Then its analysis method is proposed. Finally some experimental results and evaluation are exposed.

4.1 DGMF, an Extension of GMF

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 TDMA slots. On the other hand, 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.

4.1.1 DGMF Definitions and Properties

A DGMF task G_i is a vector composed of N_i frames F_i^j, with $1 \leq j \leq N_i$. Each frame is a job of the same task G_i. Frames have some parameters inherited from GMF:

- E_i^j is the Worst Case Execution Time (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 [6] inherited from the original task model. For DGMF task G_i with N_i frames, the GMF period P_i of G_i is:

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

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 tasks 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 $\forall F_p^q \in [F_p^q]_i^j, F_p^q \rightarrow F_i^j$. 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 $F_i^{j-1} \rightarrow F_i^j$.
- $proc(F_i^j)$ 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 $proc(F_i^j) = proc(F_x^y)$.
- $prio(F_i^j)$ 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 [6]. 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 several properties. Let use define the *Unique Predecessor* property and the *Cycle Separation* property.

Property 1 (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 $pred(F_i^j)$ such that:

$$pred(F_i^j) = \begin{cases} [F_p^q]_i^j \cup \{F_i^{j-1}\} & \text{with } j > 1 \\ [F_p^q]_i^j & \text{otherwise.} \end{cases} \tag{2}$$

Formally the Unique Predecessor property is then defined as follows:

$$\exists_{\leq 1} F_x^y \in pred(F_i^j)', r_x^y + d_x^y \geq \max(\max_{F_l^h \in pred(F_i^j)} (r_l^h + E_l^h), r_i^j) \tag{3}$$

where $\exists_{\leq 1}$ means "there exists at most one", and the set $pred(F_i^j)'$ is defined as:

$$pred(F_i^j)' = pred(F_i^j) \setminus \{F_x^y \in pred(F_i^j) \mid \exists F_k^l \in pred(F_i^j), F_x^y \rightarrow F_k^l\} \tag{4}$$

Table 3: 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 2 (Cycle Separation)**.** *A DGMF task G_i is said to respect the Cycle Separation property if:*

$$D_i^{N_i} \leq r_i + P_i \tag{5}$$

The *Unique Predecessor* and *Cycle Separation* properties simplify the analysis method of DGMF so both properties are assumed for a DGMF tasks set. Experiments, presented later in this article, 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.

4.1.2 DGMF Example

Consider the DGMF tasks set in Table 3, 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 3 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.

4.1.3 Applicability of GMF Analysis Methods on DGMF

Schedulability tests exist for GMF tasks. In this section a test is reminded and its applicability to DGMF is discussed.

In [20] 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 FP preemptive scheduling policy. Furthermore the authors also assume that the *Frame Separation* property holds: the relative deadline of a frame is less than or equal to its min-separation. Obviously this test cannot be applied to DGMF tasks for the following reasons:

- DGMF tasks have task dependencies
- DGMF frames may be allocated on different processors.

The next section proposes a scheduling analysis method for DGMF by exploiting the transaction model.

4.2 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 in this article: 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) can be expressed.
- Partitioned multiprocessor systems are considered.
- A independent GMF to transaction transformation is proposed in [28] 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 [7], not individual jobs of a task. The multiframe task models, on the other hand, express individual job parameters.

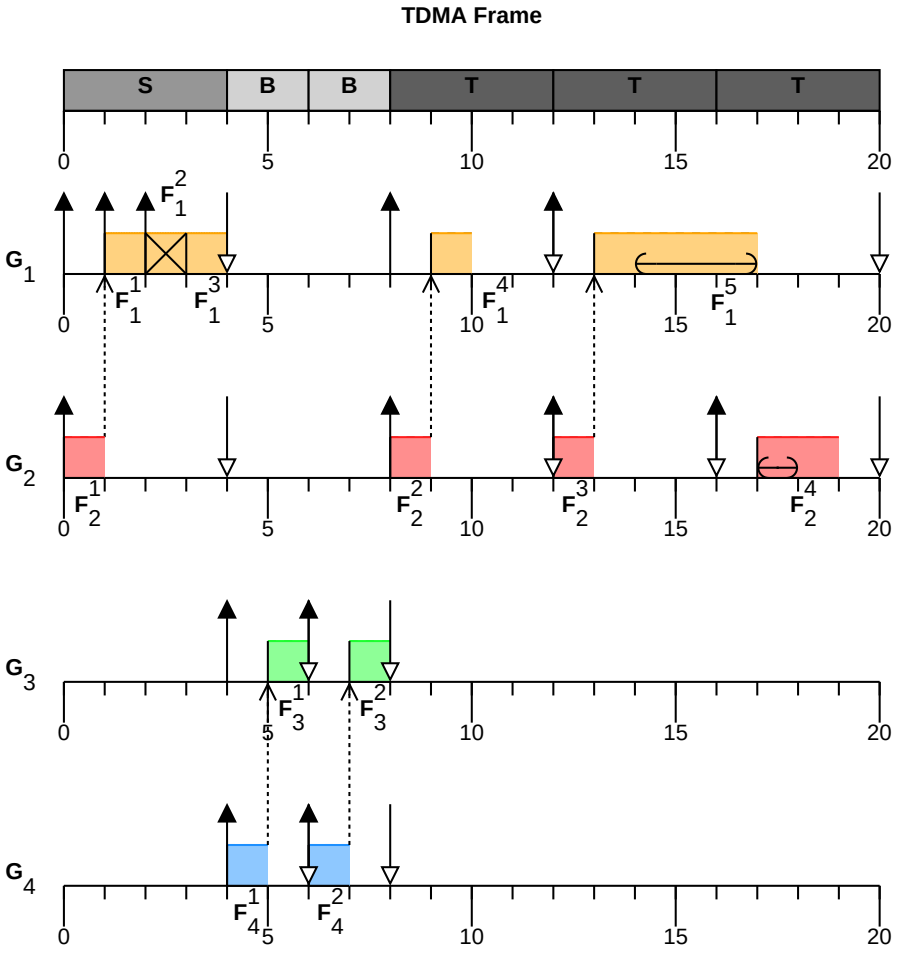

Figure 3: Example of Schedule of DGMF Tasks: Up arrows are frame releases; Down arrows are frame relative deadlines; Dashed arrows are precedence dependencies; Curved arrows are shared resource critical sections where R is used; Crossed frame executes on a different processor

In the next sections, the transaction model is first reminded. Then the transformation algorithm in [28] is extended for DGMF tasks. Afterwards the DGMF tasks set in Section 4.1.2 is transformed to a transactions set. Finally schedulability tests for transactions are discussed to choose one suitable for transactions resulting from DGMF transformation.

4.2.1 Transaction Definitions

According to [29], "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 [21], 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 [29, 7, 21, 17, 8].

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_{ij}^b is the Best Case Execution Time (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 [21]. 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 [17] 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 Worst Case Blocking Time (WCBT) [13] due to shared resources.
- $prio(\tau_{ij})$ is the fixed priority.
- $proc(\tau_{ij})$ is the processor on which τ_{ij} is allocated on.
- R_{ij}^w is the global WCRT, which is the WCRT relative to the release of the transaction [21]. 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_{ij}^b is the global Best Case Response Time (BCRT), which is the BCRT relative to the release of the transaction [21].

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 units of time of its execution time.

Tasks in a transaction are related by precedence dependencies [17]. 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. This article handles tree-shaped transactions.

Definition 1 (Tree-Shaped Transaction [8])**.** *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 $pred(\tau_{ij})$) that releases it upon completion. A task τ_{ij} may have several immediate successors (denoted by $succ(\tau_{ij})$) that it releases upon completion. The root task τ_{i1} has no predecessor.*

4.2.2 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 as 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.

Independent DGMF to Transaction **Step** 1 consists in transforming each DGMF task to a transaction by considering DGMF tasks as independent. Algorithm 4.2.2.1 shows the original algorithm proposed in [28] 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 ($prio(\tau_{ij})$), are computed from parameters E_i^j, D_i^j and $prio(F_i^j)$ 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 4.2.2.1 Independent DGMF to Transaction

```
1:  for each DGMF task G_i do
2:      Create transaction Γ_i
3:
4:      T_i ← Σ_{j=1}^{N_i} P_i^j
5:      Γ_i.r_i ← G_i.r_i
6:
7:      for each F_i^j in G_i do
8:          Create task τ_ij in Γ_i
9:
10:         C_ij ← E_i^j
11:         d_ij ← D_i^j
12:         J_ij ← 0
13:         B_ij ← 0
14:         prio(τ_ij) ← prio(F_i^j)
15:         proc(τ_ij) ← proc(F_i^j)
16:         if j = 1 then
17:             O_ij ← 0
18:         else
19:             O_ij ← Σ_{h=1}^{j-1} P_i^h
20:         end if
21:     end for
22: end for
```

Express Shared Resource Critical Sections In **Step** 2, the goal is to express critical sections of tasks τ_{ij} and then to 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

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.

In **Step** 2.**B**, the goal is to compute B_{ij} of a task τ_{ij}, assuming B_{ij} of a task is bounded [13, 12]. 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 accounted for the computation of B_{ij}. Otherwise, it may lead to a pessimistic WCBT.

Consider a task τ_{xy} that shares 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:

$$proc(\tau_{xy}) = proc(\tau_{ij}) \wedge \tag{6}$$
$$\neg SG(\tau_{xy}, \tau_{ij}) \wedge \tag{7}$$
$$\neg \exists (\tau_{kl}, R, S', B') \mid (SG(\tau_{kl}, \tau_{ij}) \wedge B' > B_{max}) \tag{8}$$

The three conditions have the following meaning:

1. Condition 1 (Equation 6) means that τ_{xy} must be on the same processor as τ_{ij}.

2. Condition 2 (Equation 7) means that both tasks must not come from frames which are part of a same DGMF task.

3. Finally, for condition 3 (Equation 8) let us 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 which 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 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 $\beta_{ij,R}$. The equation in [13] to compute the WCBT with PIP becomes:

$$B_{ij} = \sum_{R} \max_{\tau_{xy} \in \beta_{ij,R}} (\text{Critical section duration of } \tau_{xy}) \tag{9}$$

where R denotes a shared resource.

The equation in [13] to compute the WCBT with PCP becomes:

$$B_{ij} = \max_{\tau_{xy} \in \beta_{ij,R}, R} (D_{xy,R} \mid prio(\tau_{xy}) < prio(\tau_{ij}), C(R) \leq prio(\tau_{ij})) \tag{10}$$

where R denotes a shared resource, $C(R)$ the ceiling priority [13] of R, and $D_{xy,R}$ the duration of the longest critical section of task τ_{xy} using shared resource R.

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 [21]. 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.

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 $F_p^q \in [F_p^q]_i^j$. □

In the transaction model, precedence dependencies should be modeled with offsets. This is done in **Step** 3.**B** with three algorithms: Task Release Time Modification; Transaction Merge; and Transaction Release Time Modification.

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 4.2.2.2. Since the release time of τ_{pq} may also be modified by this algorithm, 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 4.2.2.2. Let us assume $\tau_{pq} \prec \tau_{ij}$. The earliest release time of τ_{ij} is r_{ij}. According to [22], $\tau_{pq} \prec \tau_{ij} \Rightarrow r_{pq} + C_{pq} \leq r_{ij}$ is true. The implication is false if $\neg(r_{pq} + C_{pq} \leq r_{ij})$, otherwise said $r_{pq} + C_{pq} > r_{ij}$. Therefore if we

Algorithm 4.2.2.2 Task Release Time Modification

```
1:  repeat
2:      NoChanges ← true
3:
4:      for each τ_pq ≺ τ_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 ← O_ij + diff
10:             d_ij ← d_ij − diff
11:             r_ij ← 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
```

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 $r_{pq} + C_{pq} \leq r_{ij}$, 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. □

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 is not compliant to the modeling of precedence dependencies in [21]. Indeed two tasks with a precedence dependency should be in the same transaction and they should be delayed from the same event that releases the transaction. Two transactions are thus merged into one single transaction if there exists a task in the first transaction that has a precedence dependency with a task in the other transaction:

$$\exists \tau_{pq}, \tau_{ij} \mid (\Gamma_i \neq \Gamma_p) \wedge (\tau_{pq} \prec \tau_{ij} \vee \tau_{ij} \prec \tau_{pq}) \tag{11}$$

Merging two transactions consist in obtaining a final single transaction, containing the tasks of both. Algorithm 4.2.2.3 merges transactions two by two until there is no more transaction to merge.

Algorithm 4.2.2.3 Transaction Merge

```
1: for each τ_pq ≺ τ_ij do
2:     if Γ_p ≠ Γ_i then
3:         for each task τ_ij in Γ_i do
4:             Assign τ_ij to Γ_p
5:         end for
6:     end if
7: end for
```

Proof of Algorithm 4.2.2.3. 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. □

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 4.2.2.4.

Algorithm 4.2.2.4 Transaction Release Time Modification

```
1: for each merged transaction Γ_m do
2:     r_m ← +∞
3:     for each τ_m^j in Γ_m, originally in Γ_o do
4:         r_m ← min(r_m, r_o + O_m^j)
5:     end for
6:     for each τ_m^j in Γ_m, originally in Γ_o do
7:         O_m^j ← r_o + O_m^j − r_m
8:     end for
9: end for
```

Proof of Algorithm 4.2.2.4. 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_m^j of a task τ_m^j in Γ_m, otherwise the definition of a transaction is contradicted. A task τ_m^j should be released at $r_m^j = r_o + O_m^j$. Once r_m is computed, when task offsets have not been modified yet, it is possible to have $r_o + O_m^j \neq r_m + O_m^j$. If we assign $r_m^j \leftarrow r_m + O_m^j$ then τ_m^j may not be released at $r_o + O_m^j$. This contradicts the fact that τ_m^j should be released at $r_m^j = r_o + O_m^j$. Therefore O_m^j must be shortened to be relative to r_m: $O_m^j \leftarrow r_o + O_m^j - r_m$. Since $r_m = \min_{\tau_m^j \in \Gamma_m} (r_o + O_m^j)$, the minimum value of $r_o + O_m^j - r_m$ is 0 and thus the assignment $O_m^j \leftarrow r_o + O_m^j - r_m$ will never assign a negative value to O_m^j. □

Algorithm 4.2.2.4 starts by finding the earliest (minimum) task release time in a merged transaction Γ_m (as a reminder, O_m^j is still relative to r_o at this moment). The earliest task release time becomes r_m. The offset O_m^j of each task τ_m^j is then 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 $t = 0$, Algorithm 4.2.2.4 produces the same merged transaction. This is the case for the transaction resulting from the transformation of the DGMF tasks set example (Section 4.1.2), which was used to model tasks constrained by a TDMA frame.

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 4.2.2.5 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 [30]. 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.

Algorithm 4.2.2.5 Reduce Precedence Dependencies

```
1:  for each task τ_ij with multiple predecessors do
2:      for each τ_iq ≺ τ_ij do
3:          if O_iq + d_iq < O_ij
4:              or ∃τ'_iq | τ'_iq ≺ τ_ij ∧ τ_iq ≺ τ'_iq then
5:              Remove τ_iq ≺ τ_ij
6:          end if
7:          if τ_ij has only one predecessor then
8:              break
9:          end if
10:     end for
11: end for
```

Proof of Algorithm 4.2.2.5. 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 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.

4.2.3 Transformation Example

The DGMF tasks set in Section 4.1.2 is transformed into a transaction Γ_1 of period $T_1 = 20$. Tasks of transaction Γ_1 are defined with parameters shown in Table 4. PCP [13] is assumed for computation of B_{ij}. Tasks are allocated on *cpu1* except τ_{12}, which is allocated on *cpu2*. Figure 4 shows the precedence dependency graph of tasks. An example of a schedule over 20 time units is shown in Figure 5. Notice that the schedule of tasks in Figure 5 is the same as the schedule of frames in Figure 3.

Table 4: 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)							

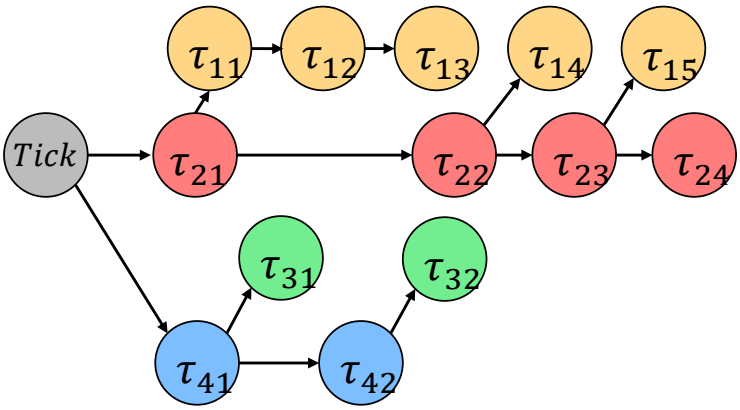

Figure 4: Tasks Precedence Dependency Graph

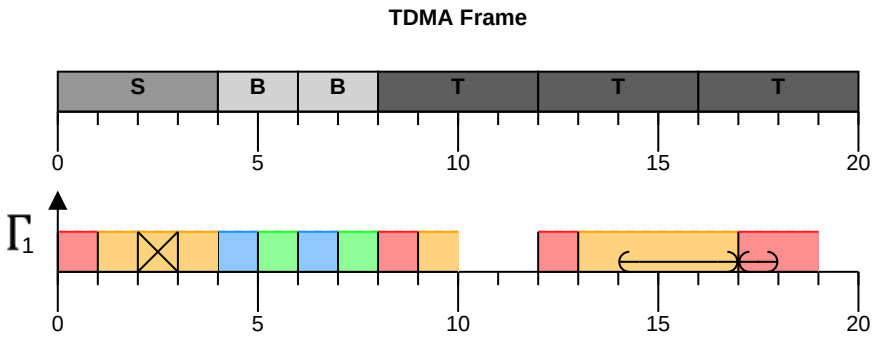

Figure 5: Transaction from DGMF Tasks Transformation: Curved arrows are shared resource critical sections; Tasks execute on same processor except the crossed task

Task *Tick* represents a ghost root task (defined below) to simplify the analysis, as proposed by [8]. In the specific example, *Tick* can represent the first TDMA tick that starts the whole TDMA frame. Therefore the 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 4. Conform to its definition, *Tick* is allocated alone on a processor (*cpu3*). 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 *cpu3*.

Definition 2 (Ghost Root Task)**.** *A ghost task is allocated alone on a processor and is 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 a processor. In a transaction, a ghost root task is a ghost task that precedes all other tasks, and does not have any predecessor.*

The next section determines the characteristics of the transaction given by the transformation example presented in this section, and in the general case. A suitable schedulability test is also discussed.

4.2.4 Assessing Schedulability of Resulting Transactions

The task parameters in Table 4, the precedence dependency graph in Figure 4, and the schedule in Figure 5 show that the transaction presented in the last section has the following characteristics:

- Tree-shaped
- Tasks may be non-immediate as defined below.

Definition 3 (Non-Immediateness)**.** *A task τ_{ix} and its immediate successor task τ_{iy} are said to be non-immediate tasks if $\tau_{ix} = pred(\tau_{iy}) \wedge O_{iy} > O_{ix} + C^b_{ix}$. 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 1. *A DGMF tasks set with the Unique Predecessor property (Property 1) 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 4.2.2.5 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 1) of F_i^j. From now, 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. $O_{iy} + d_{iy} \geq O_{ij}$). Algorithm 4.2.2.5 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 $O_{iy} + d_{iy} \geq O_{ij}$, 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 4.2.3 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. This section (Section 4) does not focus on proposing a schedulability test for tree-shaped transactions with non-immediate tasks. The test will be the subject of Section 5.

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 practice, 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 met, 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.

The following section shows some experiments that evaluate the DGMF modeling and the transformation of DGMF tasks to transactions.

4.3 Experiment and Evaluation

The DGMF scheduling analysis method is implemented in the Cheddar scheduling analysis tool. Therefore the DGMF and GMF task models, transaction model, transformation of DGMF tasks to transactions, and some schedulability tests for transactions are implemented in this tool [1].

In this section, 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.

4.3.1 Transformation Correctness

The transformation correctness is evaluated by scheduling simulation performed on randomly generated system architecture models. Cheddar provides a module that is able to generate random architecture models. The generator is updated so DGMF tasks can be produced.

The generator respects some parameters defined by the user. The user may define the number of entities in the model, i.e. processors, DGMF tasks, frames, shared resources, critical sections, and precedence dependencies. The user may also define the scheduling policy, the number of DGMF tasks with the same GMF period (called "synced DGMF tasks"), and the GMF period for such tasks. The generator avoids inconsistencies in the model (e.g. DGMF tasks with no frames). It also produces precedence dependencies in a way that avoids deadlocks during the scheduling simulation.

By using an architecture generator of Cheddar, DGMF task sets are randomly generated. The varying generator parameters are: 2 to 5 DGMF tasks, as many frames as tasks and up to 10 for each number of tasks, 1 to 3 shared resources, as many critical sections as frames, as many precedence dependencies as frames, a GMF Period between 10 to 50, and 50% of DGMF tasks with the same GMF Period. From these parameters, 540 DGMF architecture models are generated. Each of them is transformed to an architecture model with transactions.

[1] All sources available at `http://beru.univ-brest.fr/svn/CHEDDAR/trunk/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/`

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 output transactions is less than the number of input DGMF tasks, since some transactions are merged into a same one, during the transformation.

Both DGMF and transaction models are then simulated in the feasibility interval proposed in [31]. Schedules are then compared. Each DGMF schedule is strictly similar to the corresponding transaction schedule. Along with the proofs in Section 4.2.2, this experiment enforces the transformation correctness and its implementation correctness in Cheddar.

4.3.2 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 4.2.2.2, which has the same complexity as the algorithm in [22]. 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 6 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 6 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 7 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 can not be noticed in Figure 7 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 6. 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.

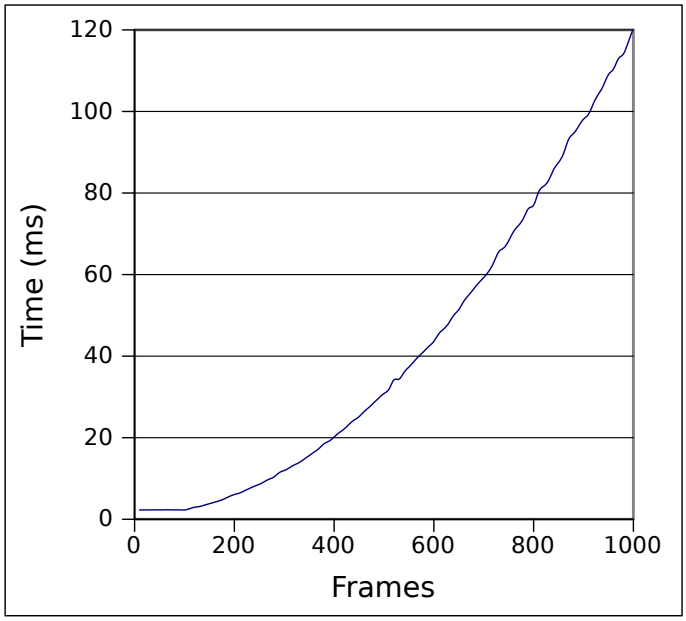

Figure 6: Transformation Duration by Number of Frames

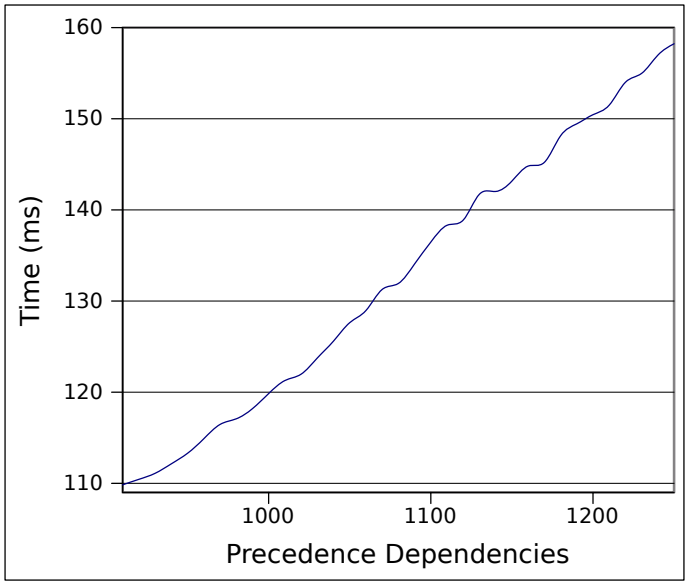

Figure 7: Transformation Duration by Number of Precedence Dependencies

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 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.

4.3.3 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 [32] 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 8.

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 [8]. The transaction is illustrated in Figure 9.

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 $FC\_S1\_1 \prec FC\_S1\_2 \prec FC\_S1\_3$).

Furthermore, it may seem 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 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,

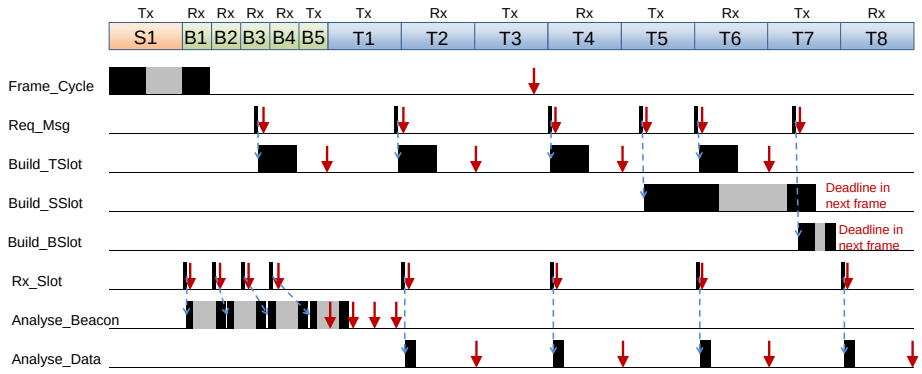

Figure 8: TDMA Frame and Threads of Real Case-Study: Line = Instances of a thread; Down arrow = Deadline; Dashed arrow = Precedence; Black = Exec on *GPP1*; Gray = Exec on *GPP2*; 36 jobs in total from 8 threads on *GPP1* and 4 threads on *GPP2*; Sizes not proportional to time values

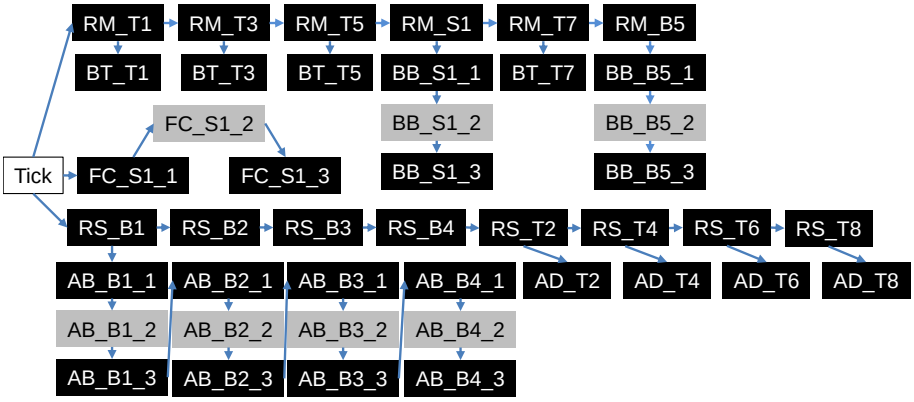

Figure 9: Tree-Shaped Transaction of a Real Case-Study: Black tasks on *GPP1*, gray tasks on *GPP2*, Tick task is ghost root task; Task nomenclature is [Name]_[Slot]_[Part (Optional)]; Abbreviated names are RM = *Request_Msg*, BT = *Build_TSlot*, BB = *Build_BSlot*, BS = *Build_SSlot*, FC = *Frame_Cycle*, RS = *Rx_Slot*, AB = *Analyse_Beacon*, AD = *Analyse_Data*

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.

In the next section, we propose a schedulability test for tree-shaped transactions with non-immediate tasks. Such type of transactions can be the result of DGMF transformation.

5 Schedulability Analysis of Tree-Shaped Transactions With Non-Immediate Tasks

Schedulability tests for tree-shaped transactions have been proposed by [8] but they do not handle non-immediate tasks. Non-immediate tasks need specific schedulability analysis. Therefore, in this section, 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 [8]. The proposed test completes the general DGMF analysis method presented in the previous section.

In the following sections, first the consequence of non-immediateness is shown. Then the WCDOPS+NIM test is exposed. First, experiments show some simulations that compare the original WCDOPS+ test to our test. Finally, our test is applied to a real case-study from Thales.

5.1 Applicability of WCDOPS+ on Non-immediate Tasks

Based on a software radio protocol architecture, we have shown in [33, 34] that if WCDOPS+ is applied directly to non-immediate tasks, then interference is underestimated. Indeed, the test does not consider that there are tasks that are not released immediately, therefore it omits some combinations of tasks that may interfere together some other task. This means WCRT computation is optimistic.

We proposed a possible solution to this underestimation problem in [33, 34]. We proposed to model 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 updated due to the existence of non-immediate tasks.

Definition 4 (Direct Predecessor and Successor in Tree-shaped Transaction)**.** *A task τ_{ij} is said to have one **direct** predecessor, denoted by $pred(\tau_{ij})$, and a set of **direct** successors, denoted by $succ(\tau_{ij})$. A task τ_{ix} is $pred(\tau_{ij})$ (resp. in*

succ(τ_{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,* *pred*(τ_{i1}) *is undefined.*

The concept of ghost tasks is introduced in [8] (Definition 2). The concept of intermediate tasks is proposed in [35]. An intermediate task represents some extra execution time or message transmission time. A ghost intermediate task can be defined as follows:

Definition 5 (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:*

$$\begin{aligned}
&C_{ixy} = C^b_{ixy} = O_{iy} - (O_{ix} + C^b_{ix}) \\
&O_{ixy} = O_{ix} + C^b_{ix} \\
&J_{ixy} = R^w_{ix} - O_{ixy} \\
&D_{ixy} = \infty \\
&B_{ixy} = 0 \\
&prio(\tau_{ixy}) = 1 \\
&\forall \tau_{kl}, proc(\tau_{ixy}) \neq proc(\tau_{kl})
\end{aligned} \tag{12}$$

Unfortunately if we model non-immediateness with ghost intermediate tasks, WCRT computation becomes pessimistic [33, 34]. Consider a task τ_{ix} and its non-immediate direct successor τ_{iy}. Even if the response time of τ_{ix} increases, it does not necessarily finish after the earliest release time of τ_{iy}. Task τ_{ix} then has no impact on the response time of τ_{iy}. With a ghost intermediate task τ_{ixy} between the two, due to the precedence dependencies, any increase to the response time of τ_{ix} will increase the response time of τ_{ixy} and thus τ_{iy} [33, 34]. Therefore the WCRT of non-immediate tasks may be overestimated if we model ghost intermediate tasks.

In the case where τ_{iy} is released immediately by τ_{ix} (i.e. case where τ_{ix} finishes after the earliest release of τ_{iy}), then τ_{iy} executes instead of lower priority tasks. Any higher priority tasks released by the lower priority tasks will not interfere τ_{iy}. Consider now that τ_{iy} is preceded by a ghost intermediate task τ_{ixy}. While τ_{ixy} is executing, the lower priority tasks can execute and release the higher priority tasks before τ_{ixy} finishes. These higher priority tasks may then interfere τ_{iy}. The response time of τ_{iy} is then impacted by both τ_{ix} and the interferences, although it is incorrect as shown in [33, 34]. This result shows again how the WCRT of a non-immediate task may be overestimated if we model ghost intermediate tasks.

In conclusion, two problems are observed when applying WCDOPS+ to transactions with non-immediate tasks. First, if applied directly, tasks interference may be underestimated, and thus WCRT computation is optimistic. Second, by modeling non-immediate tasks with ghost intermediate tasks, response

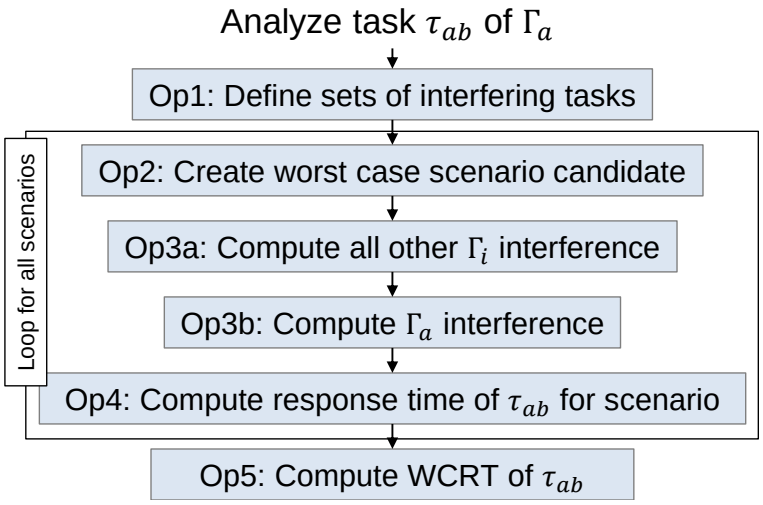

Figure 10: WCDOPS+NIM Overview: Circles indicate key operations

time and tasks interference are overestimated. WCRT computation is then pessimistic. In the following section our test proposes to solve these problems by considering the effects of non-immediateness directly.

5.2 A Schedulability Test for Non-Immediate Tasks

To consider the effects of non-immediateness directly, we propose the Worst Case Dynamic Offset with Priority Schemes Plus for Non-Immediate tasks (WCDOPS+NIM) schedulability test, which extends the original WCDOPS+ test. The general approach of the WCDOPS+NIM algorithm is the same as the WCDOPS+ algorithm. The algorithm is illustrated in Figure 10. The system to analyze has some transactions Γ_i. It is assumed that the WCRT of τ_{ab}, belonging to Γ_a, is computed.

Due to the complexity of the mathematical equations and space limitations, we will now only summarize the crucial steps of algorithm. A fully detailed explanation the WCDOPS+NIM test, and its differences with the original test, can be found in [33, 34]. We also provided theorems and proofs to justify the changes we made.

Op1: Define sets of interfering tasks The initial step is to define some task sets to help the analysis of τ_{ab}. For example some tasks of higher priority than τ_{ab} should be considered to interfere τ_{ab} as one single task if they are related by precedence dependencies [8].

In order to identify the correct set of tasks, and thus not underestimate the interferences on τ_{ab}, we can consider that non-immediate tasks are preceded by ghost intermediate tasks. We model these ghost intermediate tasks only to

compute the correct tasks sets in this step. They are not considered in the next steps.

Op2: Create worst case scenario candidate In this step, we create a worst case scenario candidate to compute the response time of of τ_{ab} within the scenario. The worst case scenario is when some higher priority task τ_{ac} (also in Γ_a) starts the busy period of τ_{ab}. As a reminder, 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 [36]. To get the worst scenario, we then start a busy period with all tasks τ_{ac}, hence a worst case scenario candidate. Task τ_{ac} is chosen by considering tasks that interfere as one single task, computed in the previous step.

When there is a non-immediate higher priority task τ_{ac}, then we consider that it can start the busy period. We also check if the non-immediate task τ_{ac} can experience jitter, due to its predecessor finishing after the earliest release time of τ_{ac}. Indeed, a non-immediate task should not experience jitter if it starts the busy period, and its predecessor task is of higher priority than the analyzed τ_{ab} task [33, 34].

Op3a, Op3b: Compute Γ_i and Γ_a interference In this step, we compute the interferences from of transactions Γ_i to τ_{ab}, including Γ_a. When interferences are computed, execution conflicts are considered. Not all sets of tasks defined in Op1 can interfere together. They are then in execution conflict. This is due to precedence dependencies, task priorities, and processor allocations [8].

When there are non-immediate tasks, we check which non-immediate task must be released immediately in the scenario. This will impact the detection of execution conflicts. Indeed, a non-immediate task released immediately by a precedent task should be in the set of the precedent task. Since execution conflicts are detected correctly, the the interferences are not overestimated [33, 34].

From an implementation point of view, the algorithm computes interferences by exploring the tree-shaped transaction with a depth-first search. Modifications are needed in the exploration algorithm, due to non-immediate tasks [33, 34].

Op4: Compute response time of τ_{ab} for scenario In this step, the response time of τ_{ab} is computed for the worst case scenario candidate. We take the longest response time of one of its jobs, since several jobs of τ_{ab} can be released in the busy period of the scenario. A response time is computed by considering the execution time of τ_{ab}, the interferences it experiences, the jitter it experiences (especially due to preceding tasks), and the WCBT it experiences due to shared resources [8].

If the longest response time comes from the scenario where $\tau_{ac} = \tau_{ab}$ starts the busy period, then τ_{ab} could have not experienced jitter, thanks to step Op2. Therefore the response time of τ_{ab} is not overestimated [33, 34].

Op5: Compute WCRT of τ_{ab} In this step, the WCRT of τ_{ab} is computed by taking the longest response time computed from one of the worst case scenario candidates.

The WCDOPS+NIM test completes the scheduling analysis method for DGMF tasks. The following section shows some experiments and results.

5.3 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 a real case-study from Thales. The following sections present these experiments. In each section the experimental setup is exposed, then experimental results are presented and discussed.

5.3.1 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. 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 [8] so both tests can be compared. The Cheddar generator is updated for the simulations.

The generator produces system architecture models composed of 4 processors all with a preemptive FP scheduling policy. In the simulations, all processors have a utilization factor that varies between 10% to 70%. A model also has 10 transactions with 10 tasks per transaction. A transaction has a period between 10 and 100000 units of time.

Initially a task has the same priority as its direct predecessor in the transaction. It is also allocated on the same processor as its direct predecessor. The direct predecessor is chosen randomly. Both priority and processor parameters can vary. The probability to choose a random priority for a task, after its default priority is set, is 0, 0.25, or 0.50, depending on the user-defined simulation parameters. There is a probability of 0.25 to choose a random processor for a task, after its default processor is set. If a parameter varies, a random priority (resp. a random processor) is chosen for a task.

Tasks are immediate initially. When its offset is computed, a generated task can become non-immediate randomly. The probability to increase a task's offset, after its default offset is set, is 0.25 or 0.50, depending on the user-defined simulation parameters. If a task becomes non-immediate, its offset is increased by a random value between 0 and 1000. WCDOPS+ is applied with ghost intermediate tasks added to model non-immediateness.

Simulations are conducted by making different parameters of the generator vary. For each set of parameters of the generator, 5 system architecture models

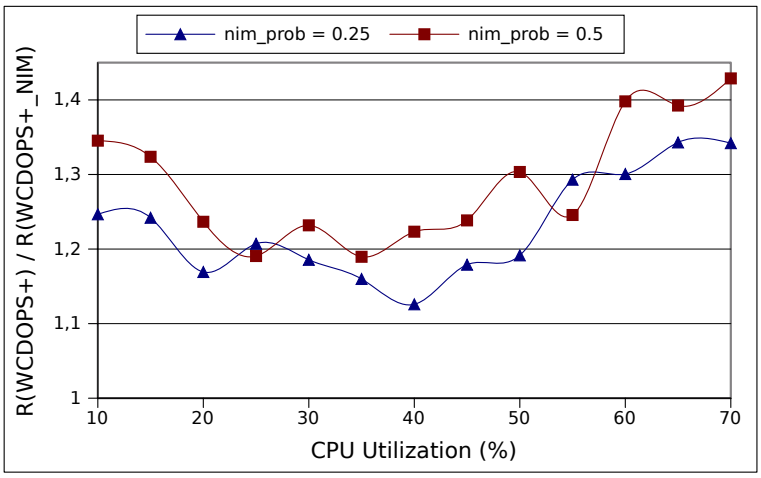

Figure 11: Comparison between WCDOPS+ and WCDOPS+NIM by Processor Utilization and Offset Increase Probability: *nim_prob* denotes the probability to increase a task's offset

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 11 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 12 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 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 [8], 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+.

5.3.2 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

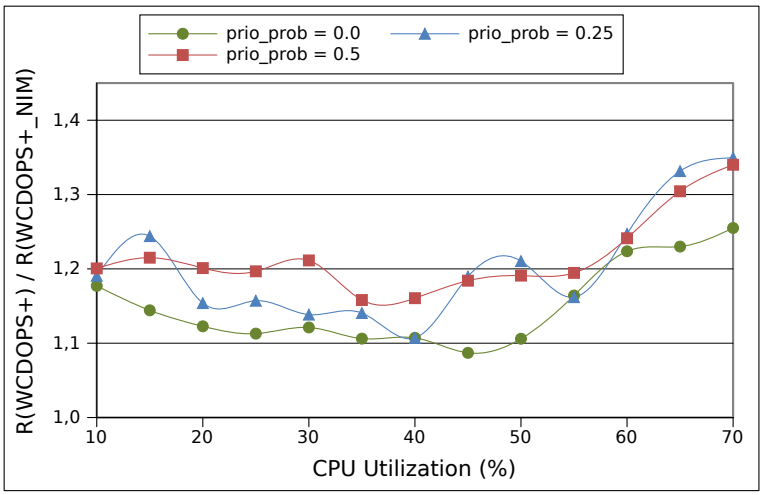

Figure 12: Comparison between WCDOPS+ and WCDOPS+NIM by Processor Utilization and Random Priority Probability: *prio_prob* denotes the probability to choose a random priority

transformations. The schedulability 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. A case-study is used to evaluate our proposed analysis method. Besides evaluating the advantages of the proposed analysis method, this experiment also determines its scalability.

The case-study is the real software radio protocol that was presented in Section 4.3.3. 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, was transformed to a tree-shaped transaction of 44 tasks with non-immediate tasks, shown in Figure 9 of Section 4.3.3.

Currently the analysis approach used at Thales is similar to the approach of the Joseph & Pandya test in [16] (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. 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

For the problem of precedence dependency between tasks on different processors, in the case-study, the priority assignment in [23] 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 than 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 by 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 5. From results in Table 5, a ratio of response times, given by both tests, is computed for each task. In average, this ratio is 8.89 so in average the J&P schedulabiltiy 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 can be 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 5.3.1, for a low 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 will decrease the slot durations, which means an increase in processor utilization.

6 Conclusion

6.1 Summary

The work presented in this article contributed to scheduling analysis of real-time radio software systems. This article focused on communication systems using

Table 5: Task WCRTs of MAC Layer Case-Study: R_{RM} is WCRT given by [16]; $R_{WCDOPS+NIM}$ 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

TDMA to access the shared communication medium. We took software radio protocols developed at Thales Communications & Security as case-studies.

Software radio protocols have some characteristics to consider for scheduling analysis. Among these characteristics, the system has tasks released by TDMA ticks, and execution times and deadlines that depend on the TDMA slots. The tasks are also dependent, through precedence dependencies and shared resources. They execute on a partitioned multiprocessor execution platform, with preemptive fixed priority scheduling.

For scheduling analysis of a software radio protocol, we solved the issue of applicability of task models and analysis methods. Indeed, existing task models of the literature are not applicable to all characteristics of a software radio protocol.

The solution proposed in this article, is to abstract the architecture of a software radio protocol with the DGMF task model. Our task model fulfills requirements for scheduling analysis of a software radio protocol based on TDMA. Indeed, our task model expresses 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 transactions. Then an adapted schedulability test, proposed in this article, is applied to the transactions. Our schedulability test is called WCDOPS+NIM, and it extends the WCDOPS+ test of the literature. 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.

The proposed solution was evaluated through several experiments. The modeling and transformation of DGMF tasks to transactions was evaluated both by simulation and with 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 than the models generated in the simulation. Indeed, the case-study was modeled with 8 DGMF tasks, 43 frames and 14 precedence dependencies.

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 test. 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 scheduling analysis of a software radio protocol based on TDMA. Experimental results also show that the solution is scalable to systems developed at Thales Communications & Security.

6.2 Future Works

In the future, we wish to improve the DGMF analysis method, by extending the WCDOPS+NIM schedulability test. 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.

The WCDOPS+NIM test is currently applicable to tree-shaped transactions. There exists an extension of WCDOPS+ for graph-shaped transactions in [37]. In the future, the WCDOPS+NIM test can be adapted for graph-shaped transactions with non-immediate tasks.

Works can also be done on exploiting the DGMF task model with higher level architecture models. In [34], we proposed an experimental architecture model of a software radio protocol in UML MARTE [38] that can be used for schedulability analysis with DGMF but The MARTE model and its transformation to

DGMF, have to be formalized in the future.

Finally, we should also investigate how DGMF can exploit other architecture models described with languages such as AADL [39], and EAST-ADL [40]. This would also investigate furthermore the applicability of DGMF to domains such as avionic and automotive.

References

[1] J. Mitola, The software radio architecture, IEEE Communications Magazine 33 (5) (1995) 26–38.

[2] T. Chan, Time-division multiple access, in: Handbook of Computer Networks, John Wiley & Sons, Hoboken, 2011, pp. 769–778.

[3] C. Liu, J. Layland, Scheduling algorithms for multiprogramming in a hard-real-time environment, Journal of the ACM 20 (1) (Jan, 1973) 46–61.

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

[5] S. Baruah, Dynamic- and static-priority scheduling of recurring real-time tasks, Real-Time Systems 24 (1) (Jan, 2003) 93–128.

[6] S. Baruah, D. Chen, S. Gorinsky, A. Mok, Generalized multiframe tasks, Real-Time Systems 17 (1) (Jul, 1999) 5–22.

[7] K. Tindell, J. Clark, Holistic schedulability analysis for distributed hard real-time systems, Microprocessing and Microprogramming 40 (2-3) (Apr, 1994) 117–134.

[8] O. Redell, Analysis of tree-shaped transactions in distributed real time systems, in: Proceedings of the 16th Euromicro Conference on Real-Time Systems, Catania, Italy, 2004, pp. 239–248.

[9] F. Singhoff, A. Plantec, P. Dissaux, J. Legrand, Investigating the usability of real-time scheduling theory with the cheddar project, Real-Time Systems 43 (3) (Jun, 2009) 259–295.

[10] H. Zimmermann, OSI reference model–the ISO model of architecture for open systems interconnection, IEEE Transactions on Communications 28 (4) (1980) 425–432.

[11] N. Malcolm, W. Zhao, The timed-token protocol for real-time communications, Computer 27 (1) (Jan, 1994) 35–41.

[12] R. Rajkumar, Real-time synchronization protocols for shared memory multiprocessors, in: Proceedings of the 10th International Conference on Distributed Computing Systems, Paris, France, 1990.

[13] L. Sha, R. Rajkumar, J. Lehoczky, Priority inheritance protocols: an approach to real-time synchronization, IEEE Transactions on Computers 39 (9) (1990) 1175–1185.

[14] H. Kopetz, Event-triggered versus time-triggered real-time systems, in: Operating Systems of the 90s and Beyond, Vol. 563 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 1991, pp. 86–101.

[15] H. Kopetz, The time-triggered model of computation, in: Proceedings of the 19th IEEE Real-Time Systems Symposium, Madrid, Spain, 1998.

[16] M. Joseph, P. Pandya, Finding response times in a real-time system, The Computer Journal 29 (5) (1986) 390–395.

[17] J. Palencia, M. Harbour, Exploiting precedence relations in the schedulability analysis of distributed real-time systems, in: Proceedings of the 20th IEEE Real-Time Systems Symposium, Phoenix, AZ, 1999, pp. 328–339.

[18] E. Wandeler, 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.

[19] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, L. Stougie, 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.

[20] H. Takada, 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, Taipei, Taiwan, 1997, pp. 80–86.

[21] J. Palencia, M. Harbour, Schedulability analysis for tasks with static and dynamic offsets, in: Proceedings of the 19th IEEE Real-Time Systems Symposium, Madrid, Spain, 1998.

[22] H. Chetto, M. Silly, T. Bouchentouf, Dynamic scheduling of real-time tasks under precedence constraints, Real-Time Systems 2 (3) (1990) 181–194.

[23] N. Audsley, K. Tindell, 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.

[24] J. Forget, F. Boniol, E. Grolleau, D. Lesens, 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.

[25] M. Forget, E. Grolleau, C. Pagetti, 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.

[26] V. Bonifaci, A. Marchetti-Spaccamela, S. Stiller, A. Wiese, Feasibility analysis in the sporadic DAG task model, in: Proceedings of the 25th Euromicro Conference on Real-Time Systems, Paris, France, 2013.

[27] P. Ekberg, N. Guan, M. Stigge, W. Yi, An optimal resource sharing protocol for generalized multiframe tasks, in: Nordic Workshop on Programming Theory, Vasteras, Sweden, 2011.

[28] A. Rahni, Contributions à la validation d'ordonnancement temps réel en présence de transactions sous priorités fixes et EDF, Ph.D. thesis, Université de Poitiers, Poitiers, France (2008).

[29] K. Tindell, Adding time-offsets to schedulability analysis, Tech. rep., University of York (1994).

[30] 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, Vol. 411, Berlin, Germany, 1990.

[31] A. Choquet-Geniet, E. Grolleau, Minimal schedulability interval for real-time systems of periodic tasks with offsets, Theoretical Computer Science 310 (Jan, 2004) 117–134.

[32] A. Burns, A. Wellings, Real-time systems and programming languages: Ada 95, real-time Java, and real-time POSIX, Pearson Education, 2001.

[33] S. Li, F. Singhoff, S. Rubini, 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.

[34] S. Li, Scheduling analysis of tasks constrained by time-division multiplexing: application to software radio protocols, Ph.D. thesis, Universit de Bretagne Occidentale, Brest, France (Nov, 2014).

[35] J. Garcia, J. Gutierrez, 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.

[36] J. Lehoczky, Fixed priority scheduling of periodic task sets with arbitrary deadlines, in: Procedings of the 11th Real-Time Systems Symposium, Lake Buena Vista, USA, 1990, pp. 201–209.

[37] J. Kany, S. Madsen, Design optimisation of fault-tolerant event-triggered embedded systems, Master's thesis, Technical University of Denmark, Lyngby, Denmark (2007).

[38] M. Z. Iqbal, S. Ali, T. Yue, L. Briand, Applying UML/MARTE on industrial projects: challenges, experiences, and guidelines, Software & Systems Modeling (Mar, 2014) 1–19.

[39] P. H. Feiler, D. P. Gluch, J. J. Hudak, The architecture analysis & design language (AADL): an introduction, Tech. rep., Software Engineering Institute, Pittsburgh (2006).

[40] V. Debruyne, F. Simonot-Lion, Y. Trinquet, EAST-ADL - an architecture description language, in: Architecture Description Languages, Vol. 176, Springer-Verlag, New York, USA, 2005, pp. 181–195.