Scheduling Analysis of TDMA-Constrained Tasks: Illustration with Software Radio Protocols

Shuai Li*†, Stéphane Rubini†, Frank Singhoff†, Michel Bourdellès*
*Thales Communications & Security, 4 av. des Louvresses, 92622 Gennevilliers, France
*Email: {first-name}.{last-name}@fr.thalesgroup.com
†Lab-STICC/UMR 6285, UBO, UEB, 20 av. Le Gorgeu, 29200 Brest, France
†Email: {last-name}@univ-brest.fr

***Abstract*—In this paper a new task model is proposed for scheduling analysis of dependent tasks in radio stations that embed a TDMA communication protocol. TDMA is a channel access protocol that allows several stations to communicate in a same network, by dividing time into several time slots. Tasks handling the TDMA radio protocol are scheduled in a manner to be compliant with the TDMA configuration: task parameters such as execution times, deadlines and release times are constrained by TDMA slots. The periodic task model, commonly used in scheduling analysis, is inefficient for the accurate specification of such systems, resulting in pessimistic scheduling analysis results. To encompass this issue, this paper proposes a new task model called Dependent General Multiframe (DGMF). This model extends the existing GMF model with precedence dependency and shared resource synchronization. We show how to perform scheduling analysis with DGMF by transforming it into a transaction model and using a schedulability test we proposed. In this paper we experiment on "software radio protocols" from Thales Communications & Security, which are representative of the system we want to analyze. Experimental results show an improvement of system schedulability using the proposed analysis technique, compared to existing ones (GMF and periodic tasks). The new task model thus provides a technique to model and analyze TDMA systems with less pessimistic results.**

I. Introduction

This paper proposes to improve scheduling analysis of systems with Time Division Multiple Access (TDMA) [3] communications. In this kind of system, the TDMA communication protocol has an impact on task semantics which leads to pessimistic scheduling analysis with the periodic task model [15]. Furthermore, we analyze systems where tasks have dependencies, which, if not considered, may lead to wrong scheduling analysis results. To specify more accurately tasks in such systems, a new task model is proposed: the Dependent General Multiframe (DGMF) model, which extends the General Multiframe (GMF) [2] model with task dependencies. This task model is applied to TDMA Software Radio Protocols (SRP) [19], systems for which the mentioned analysis issues apply.

An SRP is a software implementing a communication protocol embedded in radio stations that are part of a (mobile ad-hoc) wireless network. It is a real-time software that runs on an execution platform (e.g. operating system and hardware) and it controls how the radio's physical equipment behaves in terms of data transmission/reception over the air. An SRP is a multi-tasked system with time-constrained tasks accessing the execution platform's processors according to a scheduling policy. Tasks may have dependencies through precedence dependencies (e.g. communications) and shared resource synchronizations.

TDMA is a channel access protocol, in which a TDMA frame is synchronized between all stations. The frame is divided into several contiguous time slots of different types and durations. When critical tasks are released by slots, task scheduling depends on the way the TDMA frame is configured. As an example, a task released by a slot has its execution time that depends on the slot's type, its deadline that depends on the slot's duration, and its release time that depends on the slot's start time.

Scheduling analysis [23] is a typical technique used to verify that tasks meet their time constraints. Scheduling analysis techniques assume that simplified task models are used to model the analyzed system. The periodic task model [15] is an example of such simplified task model. In [14], we modeled and performed scheduling analysis of a TDMA SRP system with the periodic model. It was shown that this model leads to pessimistic scheduling analysis results in this context.

In this paper a new task model is proposed, to increase accuracy of scheduling analysis of TDMA oriented systems. Our model is based on the GMF model. This model was proposed to improve scheduling analysis of video decoders/encoders where task parameters depend on incoming video frames. A TDMA SRP's behavior is similar to this kind of application. Unfortunately GMF does not allow to specify task dependencies (i.e. precedence and shared resource) and SRPs have such kind of constraint. We thus propose a new GMF model called DGMF, to extend GMF with tasks dependencies.

The rest of the paper is organized as follows: in section II, we present the TDMA SRP system, i.e. our system scope and assumptions. In section III the DGMF task model is proposed. Section IV shows how to perform scheduling analysis of DGMF tasks by transforming them to transactions. In section V, experimental results show the DGMF to transaction transformation's correctness, performance, and how it is applied to a TDMA SRP from Thales. In section VI this paper's approach is compared to related work. We conclude and discuss future work in section VII.

II. Software Radio Protocol

In this paper, we consider SRP embedded in a radio station which is part of a larger physical system (e.g. helicopter, vehicular transport). An SRP is a software that implements a communication protocol. Most of the time, these radio stations communicate in a mobile ad-hoc wireless network. An SRP observes events occurring in the network (e.g. stations appearing, disappearing), transmits/receives messages and reroutes them to other stations when necessary. When a message is received, the SRP can send it to a user system or an equipment that may control the physical system in which the SRP is embedded. We assume that the effects of non-determinism in wireless networks, on scheduling analysis of a single station, are negligible.

TDMA is a common communication protocol in SRPs. In the following sections, we present a TDMA SRP through its system view and then its software and execution platform view.

A. System View

From a system point of view, a SRP is divided into several layers. Fig. 1 shows an example of such layers.

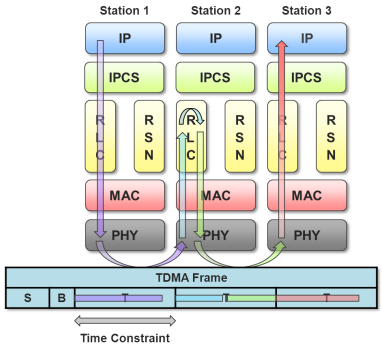

Fig. 1. TDMA SRP System View Example

In Fig. 1, the *IPCS* layer interfaces with the *IP* stack of the user system above. The *RLC* layer handles translation between IP packets and radio protocol packets. It also reroutes incoming packets if necessary (e.g. a received packet's destination is a neighbor). The *RSN* layer handles network topology and address updates (e.g. address of neighbor stations in the network appearing/disappearing). When an SRP uses TDMA, the *MAC* layer handles the TDMA protocol by preparing/receiving protocol packets for/from the *PHY* layer that sends them over the air.

In Fig. 1, control and data flows pass through the different layers. The flows are constrained by the TDMA frame. A TDMA frame is divided into several time slots of different types, durations, and modes. For example in Fig. 1, the TDMA frame has three kinds of slot: *Service* (*S*) for synchronization between stations; *Broadcast* (*B*) for observation/signaling of/on the network; *Traffic* (*T*) for effective data transmission/reception. Slots of different types do not have the same duration (e.g. a *B* slot is shorter than a *S* and *T* slot). Slots can either be in *Tx* (transmission), *Rx* (reception), or *Idle* mode. A TDMA configuration defines the combination of slots (type and mode) in a TDMA frame. A TDMA frame is repeated after it finishes, with possibly a different configuration. We assume that in a TDMA configuration, only the slot modes change from one TDMA frame to the next.

B. Software and Execution Platform

Fig. 2 shows an example of the software and execution platform architecture of an SRP.

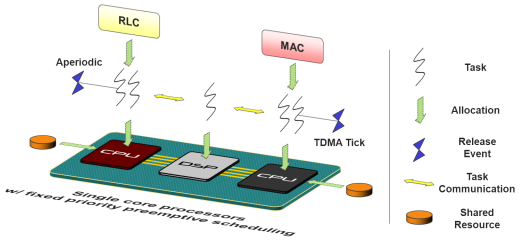

Fig. 2. Software and Execution Platform Architecture Example

From Fig. 2, we see that the layers are implemented by tasks allocated on processors. The tasks thus handle the flows that pass through layers. Tasks are scheduled by a **fixed priority preemptive policy**. Tasks may have **precedence dependency** and used **shared resources** protected by a protocol [24]. Tasks that handle the TDMA protocol have **hard deadlines** and are constrained by the TDMA frame. For example a task must be released by a TDMA "tick" indicating the start of a slot; a task has an execution time that depends on a specific slot; and a task must finish before some next slot. These tasks are thus the ones that interest us for scheduling analysis.

III. Dependent General Multiframe

Dependent General Multiframe (DGMF) is a GMF task model extended with task precedence dependency and shared resource synchronization.

GMF is well suited for TDMA SRPs where task parameters depend on TDMA slots. Indeed a GMF task is a vector of frames (not to be confused with "TDMA frame") with different parameters. This task model suits tasks released by slots of a TDMA frame, having different task parameters at each slot. However, we have to extend GMF with task dependencies because TDMA SRPs have such requirements.

In the following sections, a DGMF task is defined. An example of modeling a TDMA SRP's tasks with DGMF is then shown. We finish by discussing about scheduling analysis for DGMF.

A. DGMF Definitions

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 has parameters:

- E_i^j [2] is the Worst Case Execution Time (WCET) of F_i^j.
- D_i^j [2] is the relative deadline of F_i^j.
- P_i^j [2] is min-separation of F_i^j, i.e. the minimum time separating the release of of F_i^j and the release of F_i^{j+1}.

- $[U]_i^j$ is a set of (R, S, B) tuples denoting shared resource critical sections. F_i^j allocates resource R after it has run S time units of its execution time, and then locks the resource during the next B time units of its execution time.
- $[F_p^q]_i^j$ is a set of predecessor frames, i.e. frames from any other DGMF task that must finish before F_i^j can be released. A predecessor frame is denoted F_p^q. To avoid communication buffer overflows, F_p^q can be in $[F_p^q]_i^j$ only if G_i and G_p have the same *DGMF-Period* (defined below).
- $prio(F_i^j)$ is the priority of F_i^j.
- $proc(F_i^j)$ is the processor on which F_i^j is allocated on.

For GMF task G_i with N_i frames, we define the *DGMF-Period* of G_i as: $T_{G_i} = \sum_{j=1}^{N_i} P_i^j$

Frames are released cyclically [2]: frames are released in the order defined by the vector and any released frame $F_i^{j+k\times N_i}$ ($k > 0$) has parameters equal to frame F_i^j. The first frame to be released by a DGMF task G_i is always the first frame in its vector, denoted F_i^1.

In the GMF model, time between events was only set between frames of a same task, through the P_i^j parameter. We introduce another parameter to allow further capturing of time between events. For example a task can be specified to be released for the first time only two slots after the beginning of the TDMA frame. We call r_i^1, the release time of first frame F_i^1. We call r_i, the release time of G_i and we thus have $r_i = r_i^1$. For any F_i^j, with $j > 1$, their release time is: $r_i^j = r_i^1 + \sum_{h=1}^{j-1} P_i^h$. Finally for any F_i^j, we call value $r_i^j + D_i^j$ the global deadline of F_i^j.

A DGMF task set may have the following property:

Property 1 (Unique Predecessor): Let F_i^j be a frame of a task G_i, in a DGMF task 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, if $j > 1$. The set of frames that precede F_i^j is the set $[F_p^q]_i^j$ and F_i^{j-1} (if $j > 1$). A DGMF task 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, among frames that precede 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. Formally the *Unique Predecessor* property is defined as:

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

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

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

We assume that TDMA SRPs are modeled with DGMF task sets having the *Unique Predecessor* property.

B. DGMF Example

Consider the DGMF task set in Table I, modeling tasks constrained by a TDMA frame.

TABLE I
DGMF TASK SET

	E_i^j	D_i^j	P_i^j	$[U]_i^j$	$[F_p^q]_i^j$
G_1; $r_1 = 0$					
F_1^1	1	4	1		F_2^1
F_1^2	1	3	1		
F_1^3	1	2	6		
F_1^4	1	4	4		F_2^2
F_1^5	4	8	8	(R, 1, 3)	F_2^3
G_2; $r_2 = 0$					
F_2^1	1	4	8		F_{Tick}
F_2^2	1	4	4		
F_2^3	1	4	4		
F_2^4	2	4	4	(R, 0, 1)	
G_3; $r_3 = 4$					
F_3^1	1	2	2		F_4^1
F_3^2	1	2	18		F_4^2
G_4; $r_4 = 4$					
F_4^1	1	2	2		F_{Tick}
F_4^2	1	2	18		
$Tick$; $r_{Tick} = 0$					
F_{Tick}	0	$+\infty$	20		

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*, and F_{Tick}, which is allocated on *CPU3*. Task *Tick* is similar to the idea of the "ghost root task" introduced in [22]. Its only purpose is to represent the first TDMA tick that starts the whole TDMA frame. Task *Tick* ensures that all tasks constrained by the TDMA frame are part of the same precedence dependency graph. Fig. 3 shows an example of a schedule produced by the task set, over 20 time units, and the TDMA frame of 1 S slot, 2 B slots, and 3 T slots. 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 to specify the time between releases of G_2, G_4, and the TDMA slots. Note that precedence dependencies are respected and F_4^2 is blocked by F_1^5 during 1 time unit, due to a shared resource.

C. Applicability of GMF Scheduling Analysis on DGMF

Scheduling analysis techniques exist for independent GMF tasks. Let us see if they can be applied to DGMF tasks.

In [2] a processor utilization based feasibility test for GMF tasks is proposed. The test runs in polynomial time and is restricted to independent tasks running on a uniprocessor system under a preemptive Earliest Deadline First (EDF) scheduling policy. In [26] a response time based schedulability test for GMF tasks is proposed. This test assumes that tasks are independent and run on a uniprocessor system with a preemptive FP scheduling policy. Furthermore the authors also assume the relative deadline of a frame to be smaller than the release time of the next frame.

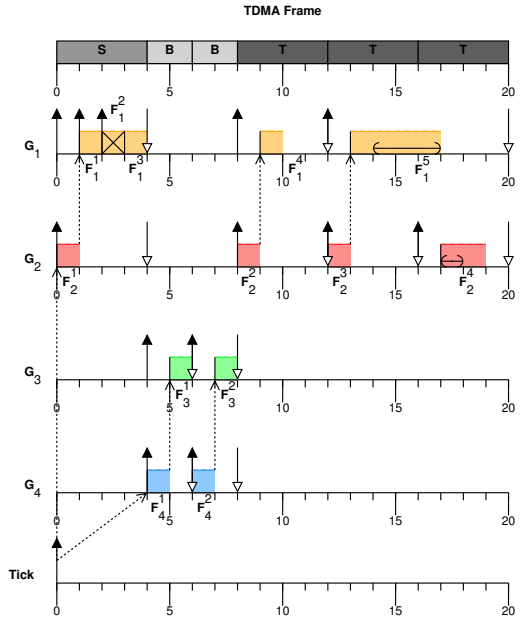

Fig. 3. DGMF Tasks in a TDMA SRP: Up arrows are frame releases; Down arrows are frame relative deadlines; Dashed arrows are precedence dependencies; Curved arrows are shared resource allocations; Crossed frame executes on different processor

Obviously original analysis techniques for GMF tasks cannot be directly applied to DGMF tasks due to task dependencies. For example in the original tests [2], [26], frames can be released simultaneously as long as they belong to different GMF tasks. This no longer holds true when precedence dependencies are defined between frames.

The following section shows how the transaction task model is used to analyze schedulability of DGMF tasks.

IV. DGMF Scheduling Analysis Using Transactions

The transaction model [28] was originally proposed to model distributed systems and compute end-to-end response times. This model allows specification of precedence dependency and shared resource synchronization.

In [21] a GMF to transaction transformation algorithm is proposed. Authors in [21] argue that scheduling analysis techniques for transactions can be applied to GMF tasks. We propose to use transactions, and their associated scheduling analysis techniques, to perform scheduling analysis of DGMF tasks. The transformation algorithm in [21] is thus extended for DGMF.

In the following sections the transaction model is defined. We then show how to transform DGMF tasks to transactions. Afterwards, a typical transformation example is presented. From this example we choose a scheduling analysis technique applicable to transactions resulting from the transformation.

A. Transaction Definitions

From [20], a transaction (denoted Γ_i) is a group of tasks (denoted 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. In this paper, we call r_i the release time of a transaction Γ_i, i.e. of the first job of Γ_i.

A job of a task in a transaction is released after/by the event that releases the 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/at t_0. Each task has parameters:

- C_i^j is the WCET. We assume the Best Case Execution Time (BCET) of a task is equal to its WCET.
- O_i^j is the offset, i.e. τ_i^j is released at least O_i^j units of time after t_0. Value $r_i^j = r_i + O_i^j$ is called the release time of τ_i^j.
- d_i^j is the relative deadline, i.e. the response time of τ_i^j must be smaller than $O_i^j + d_i^j$. Value $O_i^j + d_i^j$ is called the global deadline [20] of τ_i^j. Value $t_0 + O_i^j + d_i^j$ is called the absolute deadline of a job of τ_i^j.
- J_i^j is the maximum jitter, i.e. τ_i^j is released in $[t_0 + O_i^j; t_0 + O_i^j + J_i^j]$.
- B_i^j is the maximum shared resource blocking time [24].
- $prio(\tau_i^j)$ is the priority.
- $proc(\tau_i^j)$ is the processor on which τ_i^j is allocated on.

Tasks in a transaction are related by precedence dependencies [20]. A precedence dependency between two tasks, denoted $\tau_{ip} \prec \tau_{ij}$, is a constraint that means that a job p of τ_{ip} must finish 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}).

Tasks may access shared resources in critical sections [24]. In this paper, a critical section is denoted (τ, R, S, B) where τ is the task accessing the resource R, S is the resource allocation time, and B is the resource blocking time.

B. DGMF To Transaction

The DGMF to transaction transformation aims at expressing parameters and dependencies in the DGMF model as ones in the transaction model. The transformation has three major steps:

1 Independent DGMF to Transaction: Consider DGMF tasks independent and transform to transactions.
2 Add Shared Resource Synchronizations: Express critical sections in the resulting transaction set.
3 Add Precedence Dependencies: Model precedence dependencies in transaction model [20].

In the following sections each step is explained in detail. We will also see that the transformation already starts assessing schedulability.

1) Independent DGMF to Transaction: **Step 1** consists in transforming each DGMF task to a transaction by considering DGMF tasks as independent. Algorithm IV-B1.1 shows the original algorithm proposed by [21], that we extend for DGMF tasks. The idea behind the algorithm is to transform frames F_i^j of a DGMF task G_i into tasks τ_i^j of a transaction Γ_i. Parameters in the transaction model, like WCET (C_i^j), relative deadline (d_i^j) and priority ($prio(\tau_i^j)$), are computed from parameters E_i^j, D_i^j and $prio(F_i^j)$ from the DGMF model.

To transform the min-separation between two frames in the DGMF model, offsets (O_i^j) are used in the transaction model. The offset of a task τ_i^j is computed by summing the P_i^h of frames F_i^h preceding F_i^j in the vector of G_i. In our 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 IV-B1.1 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_i^j in Γ_i
9:
10:         C_i^j ← E_i^j
11:         d_i^j ← D_i^j
12:         J_i^j ← 0
13:         B_i^j ← 0
14:         prio(τ_i^j) ← prio(F_i^j)
15:         proc(τ_i^j) ← proc(F_i^j)
16:         if j = 1 then
17:             O_i^j ← 0
18:         else
19:             O_i^j ← Σ_{h=1}^{j-1} P_i^h
20:         end if
21:     end for
22: end for
```

Proof of Algorithm IV-B1.1: Algorithm IV-B1.1 in is based on the algorithm in [21]. The algorithm is proven by construction. ∎

2) Add Shared Resource Synchronizations: In **Step 2**, if a critical section is defined for a frame, then the task, corresponding to the frame after transformation, must also define the critical section. If there is a critical section (R, S, B) in $[U]_i^j$, then a critical section (τ_i^j, R, S, B) must be specified in the transaction set resulting from **Step 1**. When all frame critical sections have been transformed, B_i^j of each τ_i^j is computed [24].

*Proof of **Step 2**:* Task τ_i^j is the result of the transformation of F_i^j, thus by construction we must have (τ_i^j, R, S, B) if we have $(R, S, B) \in [U]_i^j$. ∎

3) Add Precedence Dependencies: The goal of **Step 3** is to add precedence dependencies to the transaction set, with respect to how they are modeled in the transaction model (i.e. with offsets and jitters [20]). **Step 3** is divided into three sub-steps:

- **3.a** Express Precedence Dependency Constraints: In the transaction set, express precedence dependency constraints from the DGMF set.
- **3.b** Model Precedence Dependency in the Transaction Model: Modifications of releases and offsets so precedence dependencies in the transaction set are modeled according to [20].
- **3.c** Reduce Precedence Dependencies: Simplify the transaction set by reducing number of precedence dependency constraints.

The following paragraphs present each of these sub-steps.

a) Express Precedence Dependency Constraints: We define two kinds of precedence dependency in the DGMF model: intra and inter. We call intra dependency a precedence dependency that is implicitly expressed between frames of a same DGMF task. 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 adding a precedence dependency between tasks, representing successive frames, if they are part of a same transaction resulting from **Step 1** (i.e. $\tau_i^j \prec \tau_i^{j+1}$ with $j < N_i$). This also ensures that these tasks are part of the same precedence dependency graph, which is important for determining the transaction's characteristics, as we will see later. Inter dependencies must also be expressed in the transaction set resulting from **Step 1**. If a frame F_p^q of task τ_p^q is in the set of predecessor frames $[F_p^q]_i^j$ of task τ_i^j, then a precedence dependency $\tau_p^q \prec \tau_i^j$ is added.

*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_i^j (resp. τ_i^{j+1}) is the result of the transformation of F_i^j (resp. F_i^{j+1}), thus by construction we must have $\tau_i^j \prec \tau_i^{j+1}$. The same proof is given for $\tau_p^q \prec \tau_i^j$, resulting from the transformation of $F_p^q \in [F_p^q]_i^j$. ∎

b) Model Precedence Dependency in the Transaction Model: In the transaction model, precedence dependencies should be modeled with the method in [20]. This is done in **Step 3.b** with three algorithms: **(ALG1)** Task Release Time Modification; **(ALG2)** Transaction Merge; and **(ALG3)** Transaction Release Time Modification.

(ALG1) Task Release Time Modification

In **ALG1** the release time r_i^j of each task τ_i^j, in the transaction set, is modified according to precedence dependencies. This enforces that the release time of τ_i^j is larger than the latest completion time $(r_p^q + C_p^q)$ of a predecessor τ_p^q of τ_i^j. Task release times are changed by modifying offsets because $r_i^j = r_i + O_i^j$, where r_i is the release time of Γ_i. The release time modification algorithm is shown in Algorithm IV-B3.1. Since the release time of τ_p^q may also be modified when the algorithm runs, release time modifications are made until no more of them occur. Note that when the offset O_i^j of τ_i^j is increased, its relative deadline (d_i^j) is shortened and then compared to its WCET (C_i^j) to verify if the relative deadline is sure to be missed.

Proof of Algorithm IV-B3.1: Let us assume $\tau_p^q \prec \tau_i^j$. The earliest release time of τ_i^j is r_i^j. We remind that it is assumed that the BCET of τ_p^q is equal to its WCET. According to [4], $\tau_p^q \prec \tau_i^j \Rightarrow r_p^q + C_p^q \leq r_i^j$ is true. The implication is false only if $\neg(r_p^q + C_p^q \leq r_i^j) \Leftrightarrow r_p^q + C_p^q > r_i^j$. Therefore if we have $\tau_p^q \prec \tau_i^j$ then we cannot have $r_p^q + C_p^q > r_i^j$. Thus, for all $\tau_p^q \prec \tau_i^j$, r_i^j must be modified to satisfy $r_p^q + C_p^q \leq r_i^j$, if $\tau_p^q \prec \tau_i^j$ and $r_p^q + C_p^q > r_i^j$. Since $r_i^j = r_i + O_i^j$, the offset O_i^j is increased to increase r_i^j. Relative deadline d_i^j is relative to

Algorithm IV-B3.1 Task Release Time Modification

```
repeat
    NoChanges ← true

    for each τ_p^q ≺ τ_i^j do
        if r_p^q + C_p^q > r_i^j then
            NoChanges ← false

            diff ← r_p^q + C_p^q − r_i^j
            O_i^j ← O_i^j + diff
            d_i^j ← d_i^j − diff
            r_i^j ← r_i + O_i^j

            if d_i^j < C_i^j then
                STOP (Deadline Missed)
            end if
        end if
    end for
until NoChanges
```

O_i^j, thus d_i^j must be decreased by the amount O_i^j is increased. ∎

(ALG2) Transaction Merge

Up until now, the transformation algorithm produces separate transactions even if they contain tasks that have precedence dependencies with other tasks from other transactions. This does not respect the modeling of precedence dependencies in [20]. Indeed two tasks with a precedence dependency should be in the same transaction and they should be delayed by a same event that releases the transaction. Two transactions are thus "merged" into one single transaction if there exists a task in one that has a precedence dependency with a task in the other:

$$\exists \tau_p^q, \tau_i^j \mid (\Gamma_i \neq \Gamma_p) \wedge \left(\tau_p^q \prec \tau_i^j \vee \tau_i^j \prec \tau_p^q\right)$$

Algorithm IV-B3.2 will merge transactions two by two until there is no more transaction to merge.

Algorithm IV-B3.2 Transaction Merge

```
for each τ_p^q ≺ τ_i^j do
    if Γ_p ≠ Γ_i then
        for each task τ_i^j in Γ_i do
            Assign τ_i^j to Γ_p
        end for
    end if
end for
```

Proof of Algorithm IV-B3.2: We remind that tasks of a transaction are related by precedence dependencies and a task in a transaction is released after/by the periodic event that releases the transaction. Let us consider two tasks τ_i^j and τ_p^q, with $\tau_p^q \prec \tau_i^j$. Task τ_i^j (resp. τ_p^q) is originally a frame F_i^j (resp. F_p^q). We have $F_p^q \in [F_p^q]_i^j \Rightarrow T_{G_i} = T_{G_p}$. G_i (resp. G_p) is transformed into Γ_i (resp. Γ_p) with period T_i (resp. T_p). We then have $T_i = T_{G_i} = T_{G_p} = T_p$. Thus τ_i^j and τ_p^q are released after/by periodic events of period $T_i = T_p$. Since $\tau_p^q \prec \tau_i^j$, τ_i^j is released after τ_p^q. Thus τ_i^j is released after the periodic event after/by which τ_p^q is released. Therefore τ_i^j and τ_p^q are released after/by the same periodic event, that releases transaction Γ_p. Both tasks then belong to Γ_p. ∎

(ALG3) Transaction Release Time Modification

After merging two transactions into Γ_m, the offset O_m^j of a task τ_m^j (originally denoted 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 IV-B3.3.

Algorithm IV-B3.3 starts by finding the earliest (minimum) task release time in a merged transaction Γ_m (we remind that 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 be modified to be relative to r_m.

Note that when transactions are merged and all of them have at least one task released at $t = 0$, then Algorithm IV-B3.3 produces the same merged transaction. We will see that this is the case for the transaction resulting from the transformation of our DGMF task set example (Section III-B), which was used to model tasks constrained by a TDMA frame.

Algorithm IV-B3.3 Transaction Release Time Modification

```
for each merged transaction Γ_m do
    r_m ← +∞
    for each τ_m^j in Γ_m, originally in Γ_o do
        r_m ← min(r_m, r_o + O_m^j)
    end for
    for each τ_m^j in Γ_m, originally in Γ_o do
        O_m^j ← r_o + O_m^j − r_m
    end for
end for
```

Proof of Algorithm IV-B3.3: 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. ∎

c) Reduce Precedence Dependencies: In **Step 3.c**, we notice that the transaction set can be simplified by reducing the number of precedence dependency constraints. This could not be done in **Step 3.a**, before **Step 3.b**, because we did not yet know the latest completion time among those of predecessors of a task. Now that offsets have been modified, we can reduce some precedence dependencies. Reducing precedence dependencies has the effect of reducing the number of predecessors/successors of a task.

Algorithm IV-B3.4 loops through tasks with more than 1 predecessor. For a specific task τ_i^j, the algorithm reduces predecessors τ_i^q that have a global deadline (i.e. $O_i^q + d_i^q$) smaller than the offset O_i^j of τ_i^j.

Algorithm IV-B3.4 Reduce Precedence Dependencies

```
for each task τ_i^j with multiple predecessors do
    for each τ_i^q ≺ τ_i^j do
        if O_i^q + d_i^q < O_i^j then
            Remove τ_i^q ≺ τ_i^j
        end if
        if τ_i^j has only one predecessor then
            break
        end if
    end for
end for
```

Proof of Algorithm IV-B3.4: Let us assume $\tau_i^q \prec \tau_i^j$ and $O_i^q + d_i^q < O_i^j$. By definition $t_0 + O_i^j$ is the earliest release time of a job of τ_i^j, 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_i^q is $t_0 + O_i^q + d_i^q$. The job of τ_i^q must finish before $t_0+O_i^q+d_i^q$ and τ_i^j is released at earliest after $t_0+O_i^q+d_i^q$ since $O_i^q+d_i^q < O_i^j$. Thus the precedence dependency constraint $\tau_i^q \prec \tau_i^j$ is already encoded in the relative deadline d_i^q of τ_i^q. Tasks in a transaction are related by precedence dependencies so τ_i^j must have at least one predecessor. ■

C. Transformation Example

The DGMF task set in section III-B is transformed into a transaction Γ_1 of period $T_1 = 20$. Tasks of transaction Γ_1 are defined with parameters shown in Table II (PCP [24] is assumed for computation of B_i^j). Tasks are all allocated on *CPU1* except τ_1^2, which is allocated on *CPU2*. Fig. 4 shows the precedence dependency graph of tasks. An example of a schedule over 20 time units is shown in Fig. 5.

TABLE II
TRANSACTION FROM DGMF TRANSFORMATION

	C_i^j	O_i^j	d_i^j	J_i^j	B_i^j	$prio(\tau_i^j)$
τ_1^1	1	1	3	0	0	1
τ_1^2	1	2	2	0	0	1
τ_1^3	1	3	1	0	0	1
τ_1^4	1	9	3	0	0	1
τ_1^5	4	13	7	0	0	1
τ_2^1	1	0	4	0	0	2
τ_2^2	1	8	4	0	0	2
τ_2^3	1	12	4	0	0	2
τ_2^4	2	16	4	0	3	2
τ_3^1	1	5	1	0	0	1
τ_3^2	1	7	1	0	0	1
τ_4^1	1	4	2	0	0	2
τ_4^2	1	6	2	0	0	2
$Tick$	0	0	$+\infty$	0	0	0
Critical Sections						
(τ_1^3 R, 1, 3), (τ_2^4 R, 0, 1)						

From the task parameters in Table II, the precedence dependency graph in Fig. 4, and the schedule in Fig. 5 we see that a transaction resulting from the transformation of DGMF tasks (modeling tasks constrained by a TDMA frame) has the following characteristics:

- Tree-shaped [22]: A tree-shaped transaction is one where each task may have zero or several successor tasks. Each task may have at most one predecessor. There is an unique "root task" in the transaction that does not have any predecessor.
- Tasks may be non-immediate, i.e. a task does not necessarily release immediately its successors.

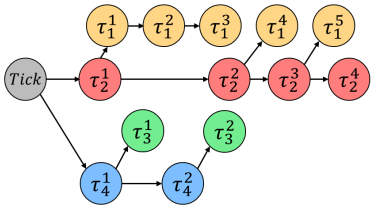

Fig. 4. Tasks Precedence Dependency Graph

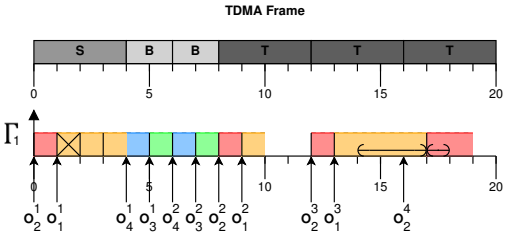

Fig. 5. Transaction Resulted from DGMF Tasks Transformation: Up arrows below timeline are offsets; curved arrows are shared resource allocations; tasks execute on same processor except the crossed task

From these characteristics, we will now propose a suitable scheduling analysis technique for this kind of task set modeling our TDMA system.

D. Assessing Schedulability of Resulting Transactions

To enforce schedulability of the resulting transactions, we use the schedulability test in [13], based on [22]. The schedulability test is applicable to tree-shaped transactions with *non-immediate tasks*. A *non-immediate task* is one that is not necessarily immediately released by its predecessor. We obtain tree-shaped transactions with *non-immediate tasks* from our DGMF to transaction transformation algorithm, if the DGMF task set has the *Unique Predecessor* property (we remind that DGMF task sets in this paper are assumed to have the property):

Theorem 1: A DGMF task 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 task set have the *Unique Predecessor* property. A frame F_i^j with inter and intra dependencies is transformed into a task τ_i^j with multiple predecessor tasks. Task τ_i^j has several predecessor tasks that correspond to frames that precede F_i^j. At most one frame F_x^y that precedes 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 task τ_i^y (resulting from F_x^y and assigned to the same transaction as τ_i^j) that precedes τ_i^j can have a global deadline greater than the offset of τ_i^j (i.e. only one τ_i^y can have $O_i^y + d_i^y \geq O_i^j$). Algorithm IV-B3.4 removes a precedence dependency $\tau_i^y \prec \tau_i^j$ if $O_i^j > O_i^y + d_i^y$. Since there is at most one predecessor τ_i^y

of τ_i^j, such that $O_i^y + d_i^y \geq O_i^j$, all other predecessors will be reduced until task τ_i^j has at most one predecessor. ■

Note that the release time r_i of a transaction Γ_i is not used by schedulability tests such as the one in [13]. This has no impact on the analysis since offsets of tasks in Γ_i are relative to r_i.

V. Experiments

Our proposition is implemented in Cheddar [6], a GPL-licensed open-source real-time scheduling analysis tool. Experiments are conducted on DGMF tasks. These experiments verify the correctness of the transformation, study the performance of the transformation, and evaluate the proposed DGMF analysis technique, compared to existing scheduling analysis techniques, when applied to a real TDMA SRP from Thales.

A. Transformation Evaluation

To verify the transformation correctness and its time performance, simulation is conducted.

1) Transformation Correctness: By using an architecture generator in 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 DGMF-Period between 10 to 50, and 50% of DGMF tasks with the same DGMF-Period.

From the combination of these varying parameters, 25600 DGMF architecture models are generated. Each of them is transformed to an architecture model with transactions. Both DGMF and transaction models are then simulated over the schedulability interval in [5] and schedules are compared.

For each architecture, we observed that the schedule of the DGMF model is strictly the same as the schedule of the resulting transaction model. Along with the proofs, this experiment enforces the transformation correctness.

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 IV-B3.1, which has the same complexity as the algorithm in [4].

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 @ 2.40GHz processor.

Fig. 6 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 resource dependencies to 0. Note that it does not matter which dependency parameter (precedence or shared resource) varies to verify the influence of n_D, since all dependencies are iterated through once in the Cheddar implementation 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. From Fig. 6 we see that the transformation duration is polynomial when the number of precedence dependencies vary. This result is consistent with the complexity, which is $O(n_D^2)$ when n_D is the varying parameter.

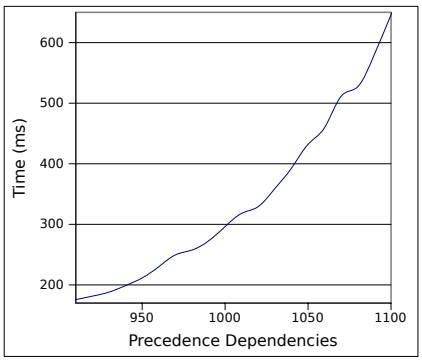

Fig. 6. Transformation Duration by Number of Precedence Dependencies

Fig. 7 shows the transformation duration by the number of frames. The number of precedence dependencies is set to 0 (i.e. no intra dependencies either) and the other parameters remains the same. From Fig. 7 we see 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 is $O(n_F)$ 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 task set. Thus the time complexity of the implementation is $O(n_F^2)$.

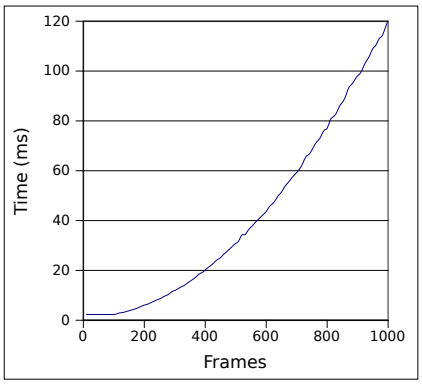

Fig. 7. Transformation Duration by Number of Frames

Overall we see that a system with no dependency, 1000 frames, and 100 DGMF tasks, takes about 120ms to be transformed on the PC used for the experiment. A system with 1100 precedence dependencies, 1000 frames, and 100 DGMF tasks, takes less than 650ms to be transformed. The transformation duration is acceptable for our needs. Indeed for a typical TDMA frame handled by Thales, with 13 slots

(1S, 4B, 8T) and 10 critical tasks, there would be a maximum of 130 frames (13×10), and 237 precedence dependencies ($10 \times 13 - 10 + 9 \times 13$) if all tasks are part of a same end-to-end flow released at each slot.

B. Experiment on a TDMA SRP from Thales

We now apply the DGMF task model to the modeling and scheduling analysis of a real TDMA SRP. The results given by DGMF analysis are compared to results given by GMF Worst Case Response Time (WCRT) analysis [26] and periodic task WCRT analysis [9]. A Cheddar model of the full case-study (8 DGMF tasks, 44 frames) can be downloaded at the Cheddar website[1]. For the sake of space, let us consider a simple TDMA frame with two slots shown in Fig. 8.

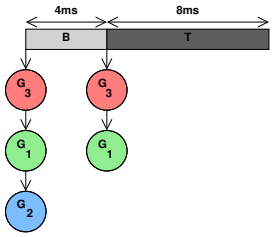

Fig. 8. Experiment TDMA SRP

In our system there are three tasks: G_1, G_2 and G_3. Task G_3 is released at the beginning of each slot. After it finishes execution, it releases G_1. G_1 releases G_2 at the first B slot but not at the T slot. G_3 and G_1, when released at a slot, must finish before the end time of the slot. G_2 when released at the B slot, must finish before the end time of slot T. Tasks are scheduled by a preemptive fixed priority scheduler and run on a single processor. PCP is used for shared resource synchronization. Tasks have parameters shown in Table III. Task priorities are in highest priority first order (e.g. a priority level 3 task has a higher priority than a priority level 1 task). Execution times come from a real SRP application. Time units are in μs.

To compare the DGMF analysis with the GMF WCRT analysis and the periodic task WCRT analysis, we also model these tasks in the GMF and periodic models. Since shared resource synchronization cannot be modeled in the GMF model, we do not consider them. If the GMF WCRT analysis is still more pessimistic without shared resources, then pessimism will not be improved with shared resources. As for the periodic model, the tasks were considered as sporadic, and then modeled as periodic. Their longest frame execution time is taken as their WCET. Their smallest min-separation time between two frames is taken as their period.

Computed WCRTs are shown in Table IV. Headers represent the task model. A response time with "/" means a deadline was missed.

[1]http://beru.univ-brest.fr/svn/CHEDDAR/trunk/project_examples/wcdops+_nimp/tdma_mac/

TABLE III
EXPERIMENT TASK SET

(D)GMF					
		E_i^j	D_i^j	P_i^j	$[F_p^q]_i^j$
G_1, $prio(G_1) = 1$	F_1^1	955	4000	4000	F_3^1
	F_1^2	1874	8000	8000	F_3^2
G_2, $prio(G_2) = 2$	F_2^1	5722	12000	12000	F_1^1
G_3, $prio(G_3) = 3$	F_3^1	986	4000	4000	
	F_3^2	986	8000	8000	
Periodic Model					
		C_i	d_i	T_i	p_i
G_1		1874	4000	4000	1
G_2		5722	12000	12000	2
G_3		986	4000	4000	3

TABLE IV
EXPERIMENT RESPONSE TIMES

		DGMF WCRT	GMF WCRT	Periodic WCRT
G_1	F_1^1	1941	/	/
	F_1^2	6523	/	
G_2	F_2^1	8649	7694	7694
G_3	F_3^1	986	986	986
	F_3^2	986	986	

From the WCRTs we see that DGMF analysis determines that no deadlines are missed. This is not the case for the other two analysis techniques.

For G_2, GMF WCRT analysis gives a lower WCRT the DGMF analysis but this value is underestimated. Indeed GMF WCRT analysis considers that F_2^1 is only interfered by F_3^1 and F_3^2, without considering the fact that F_2^1 is released after F_1^1. In conclusion DGMF analysis determines a schedulable system, and precedence dependencies must be considered by the analysis, to not underestimate WCRTs.

VI. RELATED WORK

Several works are related either to scheduling analysis of TDMA communication systems, transactions used in this paper, or the DGMF task model.

TDMA systems are networked systems for which there exist formal methods, like network calculus [12], to bound end-to-end response times of messages in the network. TDMA systems are also a special case of time-triggered systems [11] for which there exists feasibility tests [17]. Both approaches do not consider shared resources and the event-triggered [11] aspect of SRPs, through precedence dependencies. Transactions used in this paper were initially proposed by [28] to compute message end-to-end response times in a distributed system where tasks communicate through a TDMA bus. [28] thus models the holistic behavior of the TDMA bus to assess schedulability. The author's approach is based on the classic periodic task model, in which task parameter values do not vary and tasks do not have real precedence dependencies (i.e. their offset and jitter are static). In our approach, we need to model depending tasks that have variable parameters (e.g. execution time) due to functional properties related to a TDMA frame.

Schedulability tests that make use of precedence dependency conflicts [20], or execution-time dependencies [16], have been proposed for linear transactions (tasks can have at most one successor and predecessor). These tests cannot be applied to transactions resulting from DGMF transformation, since they are tree-shaped. In [18] the authors propose a model for multi-event synchronization. Their approach is to transform a system with tasks having multiple predecessors and successors, into a set of linear transactions and then use tests for linear transactions. This approach cannot be applied to transactions resulting from DGMF transformation, since these transactions are tree-shaped and they have non-immediate tasks. Tests for tree-shaped [22], [8] and graph-shaped [10] transactions have been proposed. These tests cannot be applied to transactions resulting from DGMF transformation since these transactions have non-immediate tasks.

In [27] the authors propose non-cyclic GMF tasks to model behavior of tasks in software radio modems. Their work does not consider TDMA constrained tasks and task dependencies. In [7] the authors propose an optimal resource sharing protocol for GMF tasks. In our work, we also focus on precedence dependencies that is not handled by [7]. Non-cyclic GMF is further generalized with digraph kind of task models [1], [25]. They are applicable to uniprocessor systems scheduled by EDF, without shared resources. Furthermore the feasibility tests are based on processor utilization. Contrary to DGMF, the digraph task models do not satisfy the requirements of our system: partitioned multi-processor system scheduled by a fixed-priority policy with both precedence and shared resources. Furthermore our analysis technique is a schedulability test based on response times.

VII. Conclusion

In this paper we expected to improve scheduling analysis of systems with TDMA communications and task dependencies. A TDMA Software Radio Protocol was used as an illustration. The DGMF task model was proposed to model a TDMA system. To assess schedulability of DGMF tasks, we proposed to transform DGMF tasks to tree-shaped transactions with non-immediate tasks for which we previously proposed a schedulability test. Experiments were done on a TDMA SRP from Thales and showed that the DGMF model gives less pessimistic schedulability results than both the GMF model and the periodic model.

In the future we will propose extensions of digraph task models, with task dependencies and response-time based schedulability tests.

References

[1] S. Baruah. The non-cyclic recurring real-time task model. In *Proc. 31st IEEE Real-Time Syst. Symp.*, 2010.

[2] S. Baruah, D. Chen, S. Gorinsky, and A. Mok. Generalized multiframe tasks. *Real-Time Syst.*, 17(1):5–22, Jul, 1999.

[3] T. S. Chan. Time-division multiple access. In *Handbook of Computer Networks*, pages 769–778. John Wiley & Sons, Inc., Hoboken, USA, 2011.

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

[5] A. Choquet-Geniet and E. Grolleau. Minimal schedulability interval for real-time systems of periodic tasks with offsets. *Theoretical Comput. Sci.*, 310:117–134, Jan, 2004.

[6] P. Dissaux, O. Marc, S. Rubini, C. Fotsing, V. Gaudel, F. Singhoff, A. Plantec, V. Nguyen Hong, and H. N. Nam Tran. The SMART project: Multi-agent scheduling simulation of real-time architectures. In *Proc. 7th European Congr. Embedded Real Time Software and Syst.*, Toulouse, France, 2014.

[7] P. Ekberg, N. Guan, M. Stigge, and W. Yi. An optimal resource sharing protocol for generalized multiframe tasks. In *Nordic Workshop on Programming Theory*, 2011.

[8] R. Henia and R. Ernst. Improved offset-analysis using multiple timing-references. In *Proc. Conf. Design Automation and Test in Europe 2006*, pages 450–455, Munich, Germany, 2006.

[9] M. Joseph and P. Pandya. Finding response times in a real-time system. *Comput. J.*, 29(5):390–395, 1986.

[10] J. Kany and S. Madsen. Design optimisation of fault-tolerant event-triggered embedded systems. Master's thesis, Tech. Univ. of Denmark, Lyngby, Denmark, 2007.

[11] H. Kopetz. Event-triggered versus time-triggered real-time systems. In *Operating Systems of the 90s and Beyond*, volume 563 of *Lecture Notes in Computer Science*, pages 86–101. Springer Berlin Heidelberg, 1991.

[12] J.-Y. Le Boudec and P. Thiran. *Network calculus: a theory of deterministic queuing systems for the internet*, volume 2050. Springer, 2001.

[13] S. Li, F. Singhoff, S. Rubini, and M. Bourdellès. Extending schedulability tests of tree-shaped transactions for tdma radio protocols. In *Proc. 19th IEEE Intl. Conf. Emerging Technology & Factory Automation*, 2014.

[14] S. Li, F. Singhoff, S. Rubini, and M. Bourdellès. Applicability of real-time schedulability analysis on a software radio protocol. *ACM SIGAda Ada Lett.*, 32(3):81–94, Dec, 2012.

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

[16] J. Maki-Turja and M. Sjodin. Response-time analysis for transactions with execution-time dependencies. In *Proc. 19th Int. Conf. Real-Time and Network Syst.*, pages 139–146, Nantes, France, 2011.

[17] N. Malcolm and W. Zhao. The timed-token protocol for real-time communications. *Comput.*, 27(1):35–41, Jan. 1994.

[18] A. Mehiaoui, S. Tucci-Piergiovanni, C. Mraidha, and J. Babau. Extending response-time analysis for the automatic synthesis of functional graphs into fixed-priority distributed systems. In *Proc. 9th IEEE Intl. Symp. on Industrial Embedded Syst.*, 2014.

[19] J. Mitola. The software radio architecture. *IEEE Commun. Magazine*, 33(5):26–38, 1995.

[20] J. Palencia and M. Harbour. Exploiting precedence relations in the schedulability analysis of distributed real-time systems. In *Proc. 20th IEEE Real-Time Syst. Symp.*, pages 328–339, Phoenix, USA, 1999.

[21] A. Rahni. *Contributions à la validation d'ordonnancement temps réel en présence de transactions sous priorités fixes et EDF*. PhD thesis, Univ. Poitiers, Poitiers, France, 2008.

[22] O. Redell. Analysis of tree-shaped transactions in distributed real time systems. In *Proc. 16th Euromicro Conf. Real-Time Syst.*, pages 239–248, Catania, Italy, 2004.

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

[24] L. Sha, R. Rajkumar, and J. Lehoczky. Priority inheritance protocols: an approach to real-time synchronization. *IEEE Trans. Comput.*, 39(9):1175–1185, Sep, 1990.

[25] M. Stigge, P. Ekberg, N. Guan, and W. Yi. The non-cyclic recurring real-time task model. In *Proc. 17th IEEE Real-Time and Embedded Technology and Applications Symp.*, 2011.

[26] H. Takada and K. Sakamura. Schedulability of generalized multiframe task sets under static priority assignment. In *Proc. 4th Intl. Workshop on Real-Time Computing Syst. Applicat.*, pages 80–86, Taipei, Taiwan, 1997.

[27] N. Tchidjo Moyo, E. Nicollet, F. Lafaye, and C. Moy. On schedulability analysis of non-cyclic generalized multiframe tasks. In *Proceedings of the 2010 22nd Euromicro Conference on Real-Time Systems*, pages 271–278. IEEE, 2010.

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