Efficient mapping of runnables to tasks for embedded AUTOSAR applications

Fouad Khenfri, Khaled Chaaban, Maryline Chetto

Efficient mapping of runnables to tasks for embedded AUTOSAR applications

Fouad Khenfri, Khaled Chaaban and Maryline Chetto

Abstract—An AUTOSAR-based embedded software consists of a set of inter-connected Software Components (SWCs). Each SWC contains a set of runnables which are small code-fragments that should be mapped to Operating System (OS) tasks. This process of mapping runnables to tasks may affect both the system real-time schedulability and performance. This paper proposes fast and efficient algorithms for mapping runnables to tasks. For a given embedded software containing a defined set of runnables, proposed algorithms define the number of required tasks to schedule the set of runnables, tasks priority assignment, and execution order of runnables inside each task. Experimental studies have been carried out to assess the proposed solutions. They outline both the computational efficiency, and the performance of proposed algorithms in comparison with other existing methods.

Keywords—Real-time models, Scheduling, Proofs of real-time guarantees, AUTOSAR, Automotive Electronics, Embedded Computing.

I. Introduction

Automotive manufacturers, suppliers, and tools developers have created AUTOSAR (AUTomotive Open System ARchitecture) as a worldwide industry standard for automotive Electrical/Electronic (E/E) architectures [1]. AUTOSAR provides a set of concepts and defines a methodology for automotive software development. Key features of this standard are modularity and reconfigurability. AUTOSAR permits the functional reuse of software components (SWCs) from different suppliers. It guarantees interoperability of SWCs through standardized interfaces.

An AUTOSAR-based embedded software consists of a set of inter-connected SWCs. A SWC contains one or more runnables that should be mapped to tasks. In industrial practices, most of runnables are triggered periodically and a preemptive fixed-priority scheduler is used to schedule tasks. This process of mapping runnables to tasks is thus a crucial step of AUTOSAR software design methodology, since it may affect both system performance and real-time behavior. At this stage, system designer needs to determine tasks number required to schedule runnables, tasks priority assignment, execution order of runnables inside a given task, and runnables offsets assignments (Figure 1).

A simple solution will be to assign each runnable to a separate task. However, this solution becomes quickly infeasible given the huge number of runnables to be scheduled against a limited number of tasks that can be supported by an embedded OS.

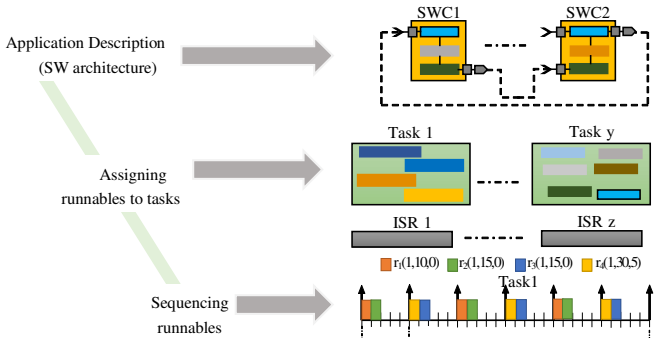


Fig. 1. Runnable mapping process in AUTOSAR.

According to AUTOSAR standard, task period could be determined using greatest common divisor (GCD) of all periods and offsets of its runnables [2]. Accordingly, we can apply three methods to map runnables to tasks, namely: Periodic Solution (PS), Multiple Periodic Solution (MPS), and Arbitrary Periodic Solution (APS).

According to PS method, runnables having same period are mapped to same task, and runnables offsets are set to zero (Figure 2). In this case, task period is equal to its runnables periods and the solution produces a number of tasks equal at least to number of distinct periods of runnables. According to MPS method, runnables with periods that are multiple of the shortest period, are mapped to same task, and runnables offsets are set to zero. Task period in this case is the least common multiple (LCM) period of task runnables. This methods produces lower number of tasks in comparison to PS method. In APS method, runnables with different periods may be mapped to same task. Task period in this case corresponds to GCD of runnables periods mapped to same task. Runnable offset may vary between 0 and runnable period. According to AUTOSAR, a runnable offset must be a multiple of its task period. This solution provides lower tasks number compared to PS and MPS methods.

Today, this step of embedded OS configuration and runables to task mapping is performed manually using system designer's empirical knowledge. This is usually error-prone and cost-consuming. Further, system architectural complexity leads to a large design decision space which is difficult to be explored without using an analytical method or a design tool.

This paper is an extension work from a previous paper [7]. The objective of that paper was purely focused on the APS

Fouad Khenfri is with Embedded Systems and Energy for Transportation (S2ET) department, ESTACA Engineering school, Paris, France, e-mail: fouad.khenfri@estaca.fr.

Khaled Chaaban is with computer science and information systems college, Umm Al-QURA University, Makkah, Saudi Arabia. e-mail: aochaaban@uqu.edu.sa

Maryline Chetto is with IRCCyN research laboratory of Nantes University, Nantes, France e-mail: maryline.chetto@univ-nantes.fr

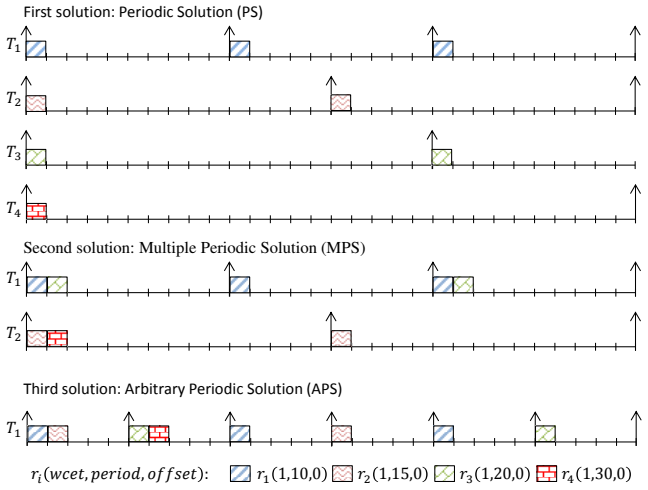


Fig. 2. Different methods to map runnables to tasks.

algorithm. From the weak points discovered in the previous algorithm, mainly the slow execution time, the high buffering of a task, which leads to a great amount of memory consumption. The current paper deals with the three algorithms "APS, MPS, PS" all together, in a way that the said disadvantages are treated.This paper proposes fast and efficient algorithms permitting to map runnables to tasks while preserving real-time system schedulability. The proposed solution determines tasks number, task priorities assignment, execution order of runnables inside a task, and runnable offsets assignment.

The rest of paper is organized as follows: Section II presents some related works. Section III describes the system model. Section IV details the developed algorithms. Section V reports and discusses obtained experiments results. Section VI concludes work and identifies potential perspectives.

II. Related works

Several recent and relevant research works addressing task mapping process for AUTOSAR embedded software have been considered by this paper [13], [15], [10], [3], [5], [6], [14],[16], [12] [8]. In [13] and [15], none of the three proposed methods (PS, MPS, and APS) is considered. In [13], authors define several runnable to task mapping rules based on inter-SWCs communication pattern. This mapping does not rely on the optimization theory, and the timing schedulability is not considered. In [15], authors consider that the runnable to task mapping and the task priority assignment are given *a priori*. A mixed integer linear programming (MILP) is used to formalize the problem of execution order of runnables inside a task, and the problem of guaranteeing data consistency (choose between several protection mechanisms) to minimize the stack memory space.

A PS method is considered in [10], [3], and [5]. In [10], a genetic algorithm guarantees the data consistency in each ECU while minimizing memory usage. Runnables are supposed to have deadlines equal to periods, and Rate-Monotonic (RM) scheduling algorithm is used for task priority assignment. An exact schedulability test is used as a constraint inequality of the optimization algorithm. In [3], a greedy algorithm aims to gather the largest number of runnables into same task. The algorithm uses either exact or sufficient schedulability tests for each clustering step. Authors consider that deadlines of runnables are lower than or equal to periods. In [5], authors address the problem of migrating the AUTOSAR applications from single core to multi-core using RM scheduling algorithm.

An MPS method with exact schedulability tests is considered in [6], [14] and [16]. In [6], a simulated annealing algorithm permits to optimize stack memory usage while the sequencing and preemption "threshold" of runnables are calculated heuristically. In [14], authors propose an optimization approach for AUTOSAR architecture synthesis, including runnables to tasks mapping process. It uses genetic algorithms and mixed integer linear programming techniques. Optimization criteria consider end-to-end timing responses and memory consumption. In [16], authors address the problems of mapping runnables to tasks to minimize stack usage for mixed-criticality systems with preemption threshold scheduling.

To our best knowledge, [12] is the unique research work addressed APS method for a multi-core architecture with a limitation (hypothesis) of only one task per core. The authors in [8] use least-loaded (LL) algorithm proposed by [12] to scheduling a mixed harmonic runnable set to one task per core.

III. System model

This section presents system model and defines terminologies used throughout the paper.

A. Runnable model

We consider a set of independent periodic runnables that should be assigned to fully preemptable fixed-priority tasks. A runnable, say r_i, is the smallest code fragment to be executed in the context of a task, say τ_j. A runnable r_i is characterized by a worst-case execution time (WCET) ϵ_i, an activation period p_i between two consecutive releases, and a relative deadline d_i with $d_i \leq p_i$. After mapping r_i to τ_j, r_i will be characterized by an offset o_i and an execution order k_i inside the task τ_j. The value of o_i and k_i must be tuned by the designer to satisfy real-time schedulability constraints.

B. Task model

In order to model an AUTOSAR task, we adapt multiframe task model which is introduced by Monk and Chen [11], as a generalization of the well-known periodic task model of Liu and Layland (or simply L&L task) [9]. A multiframe task τ_j consists of a sequence of N_j frames characterized by a 3-tuple $(\vec{E_j}, T_j, D_j)$, where $\vec{E_j} = [C_0, C_1, ..., C_s, ..., C_{N_j-1}]$ is a vector of N_j execution times $(N_j \geq 1)$, T_j is task period, and D_j is relative deadline of each frame. An execution time C_s of the s^{th} frame must be less than or equal to task period; if this condition is not verified, the behavior becomes undefined [2], [11].

To determine the three timing parameters $(\vec{E_j}, T_j, D_j)$ of task τ_j, we consider a set of runnables $\Lambda_j = \{r_1, ..., r_i, ..., r_n\}$

mapped to the same task τ_j. Task period T_j is calculated by the Greatest Common Divisor (GCD) of all runnable periods and offsets [2] mapped to this task. We assume that relative deadline (or simply deadline) of a task is the shortest deadline of runnables mapped to this task. The N_j frames of task τ_j repeat cyclically with major cycle π_j, which is equal to the least common multiple (LCM) of runnable periods. Thus, number of frames N_j can be determined by dividing the major cycle by task period ($N_j = \pi_j / T_j$). Therefore, each execution time C_s of s^{th} frame ($s = 0, 1, ..., N_j - 1$) is given by the following equation:

$$C_s = \sum_{i=1}^{n} x_i(s) * \epsilon_i \tag{1}$$

function $x_i(s)$ indicates whether the i^{th} runnable is present or not in the s^{th} frame as follows:

$$x_i(s) = \begin{cases} 1 & if\ a_i(s) = s \\ 0 & if\ a_i(s) \neq s \end{cases} \tag{2}$$

with

$$a_i(s) = \left\lfloor \frac{s}{\gamma_i} \right\rfloor \gamma_i + \delta_i \tag{3}$$

$a_i(s)$ returns a frame location of the i^{th} runnable with reference to the s^{th} frame ($a_i(s) \leq s$). $\delta_i = o_i / T_j$ and $\gamma_i = p_i / T_j$ are respectively the first position and the repetition factor of the i^{th} runnable in the task τ_j, where $((n\gamma_i + \delta_i)\ mod\ N)$ is the position of i^{th} runnable in the frames of this task, with $(n = 0, 1, 2, ...)$.

We note that the maximum execution time $\overline{E_j} = max_{s=0}^{N-1}\{C_s\}$ in the multiframe task corresponds to the WCET in the L&L task. Thus, any multiframe task $(\vec{E_j}, T_j, D_j)$ can be written as an L&L task $(\overline{E_j}, T_j, D_j)$.

Example. *Let us consider a motivating example with four periodic runnables $r_i(o_i, \epsilon_i, p_i, d_i, k_i)$ having the following parameters: $r_1(0, 1, 10, 8, 1)$, $r_2(5, 1, 15, 10, 2)$, $r_3(0, 1, 15, 12, 3)$ and $r_4(25, 1, 30, 19, 4)$. All runnables are mapped to one task τ_1. Task period T_1 is $5\ ms$ since $gcd(10, 15, 15, 30, 0, 5, 0, 25) = 5$. The major cycle π_1 is $30\ ms$ since $lcm(10, 15, 15, 30) = 30$. Thus, the number of frames is 6. Task deadline D_1 is $8\ ms$ since $\min\{8, 10, 12, 19\} = 8$. Table I depicts numerical values to illustrate the computation of execution time C_s of the s^{th} frame using Equation (1).*

TABLE I
COMPUTING WCET FOR THE s^{th} FRAME.

s^{th} frame	a_i(s)				$x_i(s)$				C_s
	r_1	r_2	r_3	r_4	r_1	r_2	r_3	r_4	
0	0	1	0	5	1	0	1	0	2
1	0	1	0	5	0	1	0	0	1
2	2	1	0	5	1	0	0	0	1
3	2	4	3	5	0	0	1	0	1
4	4	4	3	5	1	1	0	0	2
5	4	4	3	5	0	0	0	1	1

Figure 3 shows the multiframe task created by 4 runnables $\tau_1 = ((2, 1, 1, 1, 2, 1), 5, 8)$. This task contains 6 frames which repeat periodically every $30\ ms$. In each frame, the task τ_1 executes several runnables which are defined a priori.

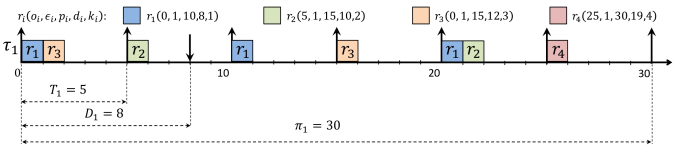


Fig. 3. Mapping runnables to tasks under AUTOSAR standard.

The following section describes a new efficient and fast heuristic algorithm inspired by Audsley's algorithm. It maps runnables to tasks while preserving real-time system schedulability and determines simultaneously the number of tasks required to schedule a set of runnables, tasks priorities assignment, as well as the sequencing of runnables inside a task.

IV. MAPPING RUNNABLES TO TASKS

Let us consider a system composed of a set of runnables $\Delta = \{r_1, ..., r_i, ..., r_n\}$. The problem consists of finding a mapping function that guarantees the system schedulability of Δ while using a minimum number of tasks. The following definition explains this function.

Definition 1. *A mapping function f from a set of runnables Δ to a set of tasks Γ, denoted by $f : \Delta \rightarrow \Gamma$, is a rule that associates each runnable r_i in Δ to a unique task τ_j in Γ. We denote the mapping of the subset of runnables $\Lambda_j \subseteq \Delta$ to task τ_j by $f(\Lambda_j) = \tau_j$. The inverse mapping function of tasks to runnables is denoted by $f^{-1}(\tau_j) = \Lambda_j$. we say that f exists if these conditions are verified:*

1) $\cap_{j=1}^{m} \Lambda_j = \emptyset$
2) $\cup_{j=1}^{m} \Lambda_j = \Delta$
3) $\forall \tau_j \in \Gamma : \tau_j$ *is timely feasible (τ_j must complete before its deadline)*

Conditions 1 and 2 reflects the surjective nature of function f, i.e., each runnable in Δ can be mapped to only one task of Γ. The function f may map multiple runnables of Δ to one task of Γ. The third condition signifies that Δ is schedulable if, and only if any task τ_j of Γ is feasible (or schedulable).

The mapping problem tries to determine function f that guarantees the schedulability of Δ with a minimal number of tasks. The following theorem helps in solving the mapping problem. We assume that runnables can have virtual priorities.

Theorem 1 (Mapping feasibility)**.** *Let Δ be a set of runnables with all their offsets set to zero, and τ_j be a task. We say that there is a set of runnables $\Pi_j \subseteq \Delta$ that can be mapped in the same task τ_j if and only if each runnable $r_i \in \Pi_j$ is feasible (schedulable) by assigning r_i to a lower priority level 1, where the remaining runnables $(\Delta - \{r_i\})$ are assigned to higher priorities in any order.*

Proof of runnables mapping theorem: We consider two runnables r_1 and r_2 with deadlines d_1 and d_2 respectively.

Let us assume that under the lowest priority level 1, the two runnables are feasible. Hence, there exist two distinct feasible priority assignment functions φ_1 and φ_2:

- case 1: runnable r_1 receives a priority which is higher than the priority of r_2, i.e., $\varphi_1(r_1) = 2$ and $\varphi_1(r_2) = 1$.
- case 2: runnable r_2 receives a priority which is higher than the priority of r_1, i.e., $\varphi_2(r_1) = 1$ and $\varphi_2(r_2) = 2$.

The response time of runnable r_1 with the lowest priority ($\varphi_1(r_1) = 1$) is equal to that of r_2 with the same priority ($\varphi_2(r_2) = 1$). Thus, r_1, and r_2 must complete execution before the shortest deadline in the two cases. Hence, the two runnables can be mapped into one task with a deadline equal to $\min\{d_1, d_2\}$. Therefore, if a set of runnables is feasible when all are receiving the lowest priority, then this set can be mapped to only one task.

Let us consider that two runnables are not schedulable when receiving the lowest priority. Clearly, no priority assignment rule permits to schedule these runnables. Moreover, there is no task for mapping them. We conclude that if a set of runnables $\Pi_j \subseteq \Delta$ can be mapped in one task, then each runnable in Π_j is feasible when assigned to the lowest priority level. ■

The following algorithm implements Definition 1 and Theorem 1 by mapping n runnables of Δ into m tasks of Γ with a time complexity (TC) of of $O\left(mn + n^2/m\right)$. Knowing that $MappingFeasibility$ and $TaskCreation$ algorithms run with a TC of $O\left(n\right)$ and $O\left(n^2\right)$ respectively.

ALGORITHM 1: Mapping runnables to tasks

```
   Input: runnables set Δ
   Output: tasks set Γ
1  Γ ← ∅; Schedulable ← True ;
2  j ← 1 ;                          /* Lowest task priority level */
3  while Δ ≠ ∅ and Schedulable do
4      Π_j ← MappingFeasibility(Δ) ;        /* Algorithm 2 */
5      if Π_j ≠ ∅ then
6          [Λ_j, τ_j] ← TaskCreation(Π_j) ; /* PS,MPS, or APS */
7          Γ ← τ_j; ;                 /* Adding τ_j to task set Γ */
8          Δ ← Δ\Λ_j; ;               /* Update Δ by eliminating
             mapped runnabes set Λ_j */
9          j ← j + 1; ;  /* Incerement priority level j */
10     else
11         Schedulable ← False ; /* Unschedulable system */
12     end
13 end
```

First, the algorithm determines a set of runnables Π_1 which is feasible at the lowest priority level using algorithm 2 (Line 4). If such a set is found, one task, say τ_1 with a lower priority $j = 1$ is created. A set of runnables $\Lambda_1 \subseteq \Pi_1$ is mapped to task τ_1 using PS, MPS or APS task creation algorithms detailed afterwards (Line 6).

Second, if $\Delta \backslash \Lambda_1$ is not empty, the algorithm determines the second set of runnables Π_2 which is feasible at the next lowest priority level. If Π_2 exists (amongst the $|\Delta \backslash \Lambda_1|$ runnables that have not yet been mapped into task (Line 8)), a task τ_2 with priority $j = 2$ is created. The set of runnables $\Lambda_2 \subseteq \Pi_2$ is mapped into task τ_2. Successively, the set of tasks $\Gamma = \{\tau_1, ..., \tau_j, ..., \tau_m\}$ is set up until obtaining an empty set Δ, where $\Lambda_1 \cap \Lambda_2, ..., \cap \Lambda_m$ is empty, and $\Lambda_1 \cup \Lambda_2, ..., \cup \Lambda_m = \Delta$. If at any mapping level j, no set of runnables Π_j can be found ($\Pi_j = \emptyset$), then the whole system is declared as not schedulable (Line 11).

The following subsections details $MappingFeasibility$ and $TaskCreation$ algorithms which are called by the main algorithm 1.

A. Mapping feasibility

This section describes how to implement the mapping feasibility theorem 1. The goal is to find, for each priority level j, a set of runnables $\Pi_j \subseteq \Delta$ that can be mapped to one task, say τ_j. According to theorem 1, the set of runnables Π_j exists if the following feasibility condition is satisfied for all runnables in Π_j:

$$R_i = \epsilon_i + \sum_{k \in \Delta - \{r_i\}} \left\lceil \frac{R_i}{p_k} \right\rceil \epsilon_k \leq d_i \tag{4}$$

R_i is the WCRT (Worst-case Response Time) of runnable r_i when assigned to lowest priority and all runnables release simultaneously. Such release time is known as the critical instant for r_i since it leads to longest response time [9]. Furthermore, all runnables are feasible at the lowest priority level; they have the same WCRT. To reduce algorithm complexity, feasibility test given by inequality (5), efficiently defines the set of runnables to be mapped in one task.

$$\forall r_i \in \Delta : R = \sum_{k \in \Delta} \left\lceil \frac{R}{p_k} \right\rceil \epsilon_k \leq d_i \tag{5}$$

The WCRT R in inequality (5) can be solved by the following recurrence relation:

$$R_{t+1} = \sum_{k \in \Delta} \left\lceil \frac{R_t}{p_k} \right\rceil \epsilon_k \tag{6}$$

where $R_0 = \sum_{k \in \Delta} \epsilon_k$. Equation (6) surely converges if processor utilization factor does not exceed 100%. Unschedulability condition is given by the following condition:

$$R_{t+1} > \max_{k \in \Delta} \{d_k\} \tag{7}$$

Algorithm 2 implements Theorem 1 and includes feasibility test (5) and unschedulability test (7). This algorithm finds a set of runnables feasible to map in the same task from n runnables with a time complexity of $O\left((\max(p_i)/\min(p_i))n\right)$.

B. Task creation

This section presents the second algorithm, namely $TaskCreation$, to create a task τ_j for any set of runnables Π_j given by the $MappingFeasibility$ algorithm. The $TaskCreation$ algorithm selects a set of runnables $\Lambda_j \subseteq \Pi_j$ using one of the mapping methods PS, MPS, or APS. The algorithm assigns offset o_i and execution order k_i for each runnable $r_i \in \Lambda_j$; and simultaneously computes three timing parameters of the task: vector of execution times $\vec{E}_j$, period T_j, and deadline D_j.

In this work, it is assumed that execution order of runnables corresponds to the increasing order of their deadlines, i.e., a runnable with a shortest deadline runs first.

ALGORITHM 2: Mapping feasibility test

Input: $runnable\ sets\ \Delta$
Output: Π_j

$\Pi_j \leftarrow \emptyset$; $R \leftarrow \sum_{k\in\Delta} \epsilon_k$; $D_{max} \leftarrow \max_{k\in\Delta}\{d_k\}$;
$schedulability = continue \leftarrow True$;
while *continue* **do**
  $R_t = \sum_{i\in\Delta} \left\lceil \frac{R}{p_i} \right\rceil \epsilon_i$;
  if $R = R_t$ **then**
    continue ← False;
  end
  if $R_t > D_{max}$ **then**
    continue ← False ; /* Δ is unschedulable */
    $schedulability \leftarrow False$;
  end
  $R \leftarrow R_t$;
end
if $schedulability$ **then**
  for $r_i = 1$ *to* $|\Delta|$ **do**
    if $(R \leq d_i)$ **then**
      $\Pi_j \leftarrow \Pi_j \cup \{r_i\}$
    end
  end
end

*1) **Task creation using PS method**:* The runnables having same periods are mapped to the same task, and all runnable offsets are set equal to zero. In this case, task period T_j is equal to period of its runnables, and WCET $\overline{E_j}$ is the sum of WCET of all runnables. We say that there is a set of runnables $\Lambda_j \subseteq \Pi_j$ mapped to the same task using PS method, if, and only if:

Condition 1: $\forall i \in \Lambda_j, p_i = T_j$
Condition 2: $\overline{E_j} = \sum_{i=1}^{|\Lambda_j|} \epsilon_i \leq T_j$
Condition 3: $R \leq \min_{i=1}^{|\Lambda_j|} d_i$

By using mapping feasibility theorem, second and third conditions are verified since that feasibility test ensure that R is less than or equal to the shortest deadline of runnables. Now, the following algorithm implements the first condition to select a set of runnables Λ_j. In this case, it is sufficient to choose an arbitrary runnable of the set Π_j to find the task period T_j. Here, we consider the last runnable of Π_j sorted in ascending deadline order. Thus, the period of task j is the period of this runnable. Algorithm 3 creates a task from n runnables with a time complexity of $O(n^2)$ using insertion sort algorithm.

ALGORITHM 3: Task creation using PS method

Input: runnable set Π_j
Output: Λ_j and $\tau_j\left((\overline{E_j}), T_j, D_j\right)$

Sort Π_j in ascending deadline order;
$\Lambda_j \leftarrow \emptyset$; $\overline{E_j} \leftarrow 0$; $T_j \leftarrow p_{|\Pi_j|}$; $D_j \leftarrow d_{|\Pi_j|}$; $k \leftarrow 0$;
for $i \leftarrow 1$ *to* $|\Pi_j|$ **do**
  if $T_j = p_i$ **then**
    $\Lambda_j \leftarrow \Lambda_j \cup \{r_i\}$;
    $k \leftarrow k + 1$;
    $k_i \leftarrow k$; /* Execution order of i^{th} runnable */
    $o_i \leftarrow 0$; /* Offset of i^{th} runnable */
    $D_j \leftarrow \min(D_j, d_i)$; /* Deadline of j^{th} task */
    $\overline{E_j} \leftarrow \overline{E_j} + \epsilon_i$; /* WCET of j^{th} task */
  end
end

*2) **Task creation using MPS method**:* Runnables having multiple periods are mapped to the same task, and all runnable offsets are set equal to zero. In this case, task period T_j is the shortest period of its runnables, and WCET $\overline{E}$ is the sum of WCET of all runnables. We say that there is a set of runnables $\Lambda_j \subseteq \Pi_j$ mapped to the same task using MPS method, if, and only if:

Condition 1: $\forall i \in \Lambda_j : mod(p_i, T_j) = 0$
Condition 2: $\overline{E} = \sum_{i=1}^{|\Lambda_j|} \epsilon_i \leq T_j$
Condition 3: $R \leq \min_{i=1}^{|\Lambda_j|} d_i$

Second and third conditions are verified since that feasibility test ensure R is less than or equal to the shortest deadline of runnables. Algorithm 4 implements only first condition to select a set of runnables Λ_j. Task period is the shortest period of runnables, which are multiple of a runnable period T (refers to first loop in Algorithm 4). This period T can be arbitrarily chosen from runnable set Π_j. Algorithm 4 chooses the period T as the period of runnable with longest deadline. Algorithm 4 creates a task from n runnables with a time complexity of $O(n^2)$.

ALGORITHM 4: Task creation using MPS method

Input: runnable set Π_j
Output: $\wedge_j$ and $\tau_j\left(\vec{E_j}, T_j, D_j\right)$

Sort Π_j in ascending deadline order;
$\Lambda_j \leftarrow \emptyset$; $C_{\tau_j} \leftarrow 0$; $D_j \leftarrow d_{|\Pi_j|}$; $T \leftarrow p_{|\Pi_j|}$;$T_j \leftarrow T$; $k \leftarrow 0$;
for $i \leftarrow 1$ *to* $|\Pi_j|$ **do**
  if $mod\,(T, p_i) = 0$ **then**
    $T_j \leftarrow \min\{T_j, p_i\}$; /* Period of j^{th} task */
  end
end
for $i \leftarrow 1$ *to* $|\Pi_j|$ **do**
  if $mod\,(T_j, p_i) = 0$ **then**
    $\Lambda_j \leftarrow \Lambda_j \cup \{r_i\}$;
    $k \leftarrow k + 1$;
    $k_i \leftarrow k$; /* Execution order of i^{th} runnable */
    $\pi_j \leftarrow \max(\pi_j, p_i)$; /* Major cycle of j^{th} task */
    $o_i \leftarrow 0$; /* Offset of i^{th} runnable */
    $D_j \leftarrow \min(D_j, d_i)$; /* Deadline of j^{th} task */
    $\overline{E_j} \leftarrow \overline{E_j} + \epsilon_i$; /* WCET of j^{th} task */
  end
end
Compute the N_j execution times of $\vec{E_j}$ by using the Eq. 1 with $N_j = \pi_j / T_j$;

*3) **Task creation using APS method**:* Runnables with different periods are mapped to the same task. Runnable offsets are not forced to set to zero. In this case, task period T_j is the GCD of runnable periods. We say that there is a set of runnables $\Lambda_j \subseteq \Pi_j$ mapped in the same task for APS if, and only if:

Condition 1: $\forall i \in \Lambda_j : T_j = gcd_{i=1}^{|\Lambda_j|} p_i > 1$
Condition 2: $\overline{E} = \sum_{s=0}^{N-1} C_s \leq T_j$
Condition 3: $R \leq \min_{i=1}^{|\Lambda_j|} d_i$

Third condition is verified given that feasibility test ensures that R is less than or equal to shortest deadline of runnables. We note that task deadline using APS method can be greater than task period. Thus, first condition allows avoiding highest buffering of a task instance. Considering a realistic case of five runnables $r_1(0, 0.5, 15, 15, 1)$, $r_2(1, 0.5, 18, 18, 2)$, $r_3(2, 0.5, 25, 25, 3)$, $r_4(3, 0.5, 35, 35, 4)$, $r_5(4, 0.5, 55, 55, 5)$ mapped in the same task τ_2. Figure 4 shows that higher priority task τ_1 delays several executions of τ_2. After τ_1 completes, τ_2, with its input buffers filled, runs in burst mode.

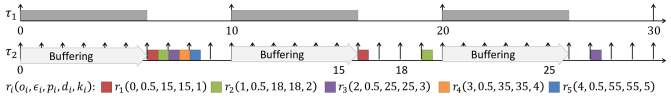


Fig. 4. Example of buffering case.

The number of instances of any task τ_j that needs to be buffered is bounded by $\lceil R_{\tau_j}/T_j \rceil$, where R_{τ_j} and T_j are respectively WCRT and period of τ_j. In a larger response time with $T_j = 1$, there may be hundreds or thousands of task instances. To avoid this, first condition allows creating subsets of a set of runnable periods, where one of these subsets must be mapped to the same task. To do this, we propose the following algorithm named Bucket Select Algorithm (BSA).

ALGORITHM 5: Bucket Select Algorithm

```
   Input: runnable set Π_j and list of k first primes number L
   Output: a new runnable set ∧_j and task period T_j
1  create k buckets B;
2  create three lists T_b, and P_b of k length initialized to 0;
3  for i ← 1 to |Π_j| do
4      for j ← 1 to k do
5          if mod(p_i, L[j]) = 0 then
6              Insert p_i into bucket B[j];
7              T_b[j] ← gcd(T_b[j], p_i);
8              P_b[j] is the first prime number, where T_b[j] is divisible by it;
9          end
10     end
11 end
12 T_j ← 0 ;                                   /* Period of task */
13 for j ← 1 to k do
14     if L[j] = P_b[j] and T_j < T_b[j] then
15         T_j ← T_b[j];
16         ∧_j ← B[j] ;        /* Select the bucket that has a
             great period value */
17     end
18 end
```

Bucket Select Algorithm (BSA) works by partitioning a set of runnable periods of a cardinality n into a number of buckets using a divisibility test by k prime numbers. Therefore, each bucket contains a subset of runnable periods characterized by two timing parameters: T_b is the GCD of runnable periods, and P_b is the minimum prime number, where T_b is divisible by it. The Bucket Select given by the Algorithm 5 and described in Figure 5 selects one subset with a time complexity of $O(kn)$.

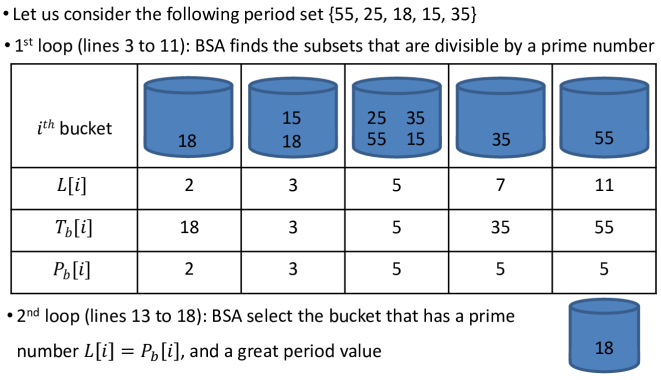


Fig. 5. Example of Bucket Select Algorithm (BSA).

To verify second condition, which is the overload constraint of task, we need to find for each runnable an offset. This last must be a multiple of task period and should be less than runnable period. Each runnable has $\gamma_i = p_i/T_j$ possibilities for mapping in task. Consequently, we need to search an optimal solution between $\prod_{i=1}^{n}(\gamma_i)$ solutions for n runnables. To find an acceptable solution, authors in [12] developed two algorithms, which are Least Loaded (LL) for harmonic periods and Lowest Peak (LP) for non-harmonic periods. These algorithms assume that WCETs of runnables are small compared to the period of task that is fixed at $5ms$. We are interested in non-harmonic cases, and we will adapt LP algorithm. LP algorithm, described from line 4 to 15, chooses the runnable offset through a larger window of frames T_w, which is equal to the LCM of runnables periods already scheduled at the current state of the algorithm. Runnable offset $o_i = \delta_i * T_j$ is selected from the first γ_i frames, and gives the lowest peak (LP) over T_w (or T_w/T_j frames), knowing that the schedule repeats the runnable in cycle afterward. The peak execution time $\overline{E_j} = max_{s=0}^{T_w/T_j} C_s$ must be less than or equal to task period T_j (line 9). The following algorithm uses BSA algorithm and adapts LP algorithm to create a task from n runnables using APS method with a TC of $O((k+N)n+n^2)$, where N_j is the frame number of a multiframe task , and k is the first prime numbers.

ALGORITHM 6: Task creation using APS method

```
   Input: runnable set Π_j
   Output: Λ_j and τ_j(E_j, T_j, D_j)
1  Λ_j ← ∅; D_j ← ∞;
2  let L be the list of k first prime numbers;
3  [∧_j, T_j] ← BucketSelectAlgorithm(Π_j, L);
4  Sort ∧_j by increased period order ;
5  T_w ← p_1; π_j ← p_1;
6  for i in ∧_j do
7      T_w ← lcm(π_j, p_i);
8      In the first γ_i frames, find the first position δ_i of the runnable r_i which
         minimize the highest load in the T_w/T_j first frames;
9      if E_j ≤ T_j then
10         Λ_j ← Λ_j ∪ {r_i};
11         o_i ← δ_i * T_j ;          /* Offset of i^th runnable */
12         D_j ← min{D_j, d_i} ;      /* Deadline of j^th task */
13         π_j ← T_w ;                /* Major cycle of j^th task */
14     end
15 end
16 Sort Λ_j in ascending deadline order to define the execution order of runnables;
17 Compute the N_j execution times of E_j by using Eq.1 with N_j = π_j/T_j;
```

We note that the TC of all algorithms described above can be reduced by using the merge sort algorithm with a logarithmic time complexity.

V. Experimentation

Several experimentations have been carried out to evaluate Algorithm 1 with the application of PS, MPS, and APS methods. To well assess proposed solutions, two other heuristic algorithms using PS are implemented.

First algorithm is named RMS (Rate Monotonic Scheduling) which tries to map n runnables to m tasks, where m is the number of different periods of runnables set, in two stages:

- in the first stage, runnables with same periods are grouped into same task with a time complexity of $O(mn)$;
- in the second stage, an insertion sort algorithm is used to determine task priorities by assigning highest priorities to

the shortest deadline of runnables mapped to a task, with a time complexity of $O(m^2)$.

Time complexity of this RMS-based mapping algorithm is of $O(m^2 + mn)$ without considering the schedulability test. In this experimentation, an exact schedulability test (i.e., each task's WCRT is less than its relative deadline) is used to check system schedulability.

Second algorithm is named GBFS and is developed by [3]. It has a polynomial complexity of $O(n^4)$.

To assess the proposed solutions under different execution scenarios, a set of runnables is randomly generated using UUnifast algorithm developed by [4]. This distributes a total utilization factor U to smaller utilization factors u_i for n runnables. For each runnable r_i, deadline d_i is determined by the following equation:

$$d_i = (p_i - \epsilon_i) rand(a, b) + \epsilon_i \tag{8}$$

where $rand(a, b)$ generates values from a uniform distribution on the interval $[a, b]$ with $0 \leq a \leq b$. In the case of $a = b = 1$, deadline is equal to period. WCET ϵ_i is equal to the product of smaller utilization factor u_i by the period p_i, which is randomly chosen from a set of periods.

All algorithms have been compared based on four evaluation criteria: scheduling success rate, execution time, number of tasks, and the average response time.

A. Scheduling success rate

In this section, scheduling success rate is compared for all algorithms. Following definition clarifies this criteria:

Definition 2 (Success rate)**.** *Let us consider a set* $\Sigma = \{\Delta_1, \Delta_2, ..., \Delta_i, ..., \Delta_N\}$ *composed of* N *subsets of runnables* Δ_i*, and* A *a scheduling algorithm. We say that* Δ_i *is schedulable by scheduling algorithm* A*, if all runnables from* Δ_i *meet their deadlines, denoted by* $A(\Delta_i) = 1$*, and if not we denote it by* $A(\Delta_i) = 0$*. As a result, success rate* SR *of scheduling algorithm* A *on* Σ *is defined by:*

$$SR = 100 * \frac{\sum_{i=1}^{N} A(\Delta_i)}{N} \tag{9}$$

Figure 6 shows success rate of all algorithms depending on deadline interval. For each interval $[a, b]$, we run all algorithms on 1000 runnables sets with a cardinality of 100 and a utilization factor of 90%. Periods used to randomly generate each set of runnables are choosen from following set $T = \{5, 10, 15, 20, 25, 30, 40, 45, 50, 60, 75, 80, 90, 100, 125\}$.

For all deadline intervals, algorithms PS, MPS, and APS have obtained the same success rate. However, our proposed algorithms outperform RMS and GBFS in terms of scheduling success rate.

B. Run-time analysis

This subsection provides run-time analysis of algorithms by executing each algorithm on different sets of runnables with a cardinality that varies from 10 to 10000. For each cardinality value, ten sets of runnables are randomly generated with a deadline on request (i.e., deadline equals period), a utilization

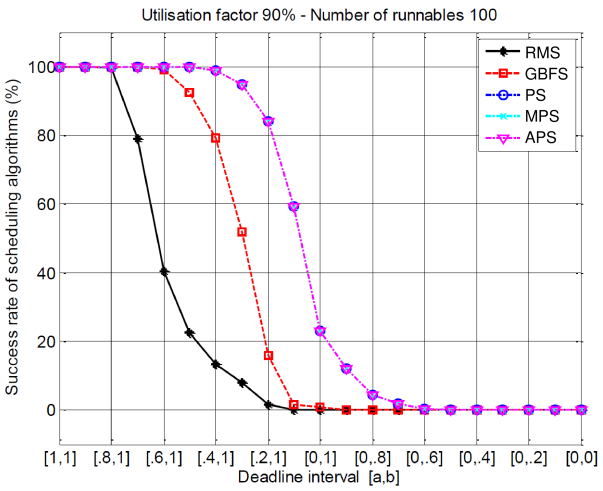


Fig. 6. Success rate of scheduling algorithms depending on deadline.

factor of 60%, and an identical periods set used in the previous subsection. All algorithms are successively run on all sets of runnables, and the average execution time is computed for each one.

Figure 7 shows obtained experimental results. We note that execution time of algorithm GBFS explodes very quickly. However, other algorithms have a reasonable execution time. Time complexity of our three algorithms obtained by this test is similar to the one obtained in section IV $O\left(mn + n^2/m\right)$. For a small number of tasks m, we get an important time complexity. As a result, time complexity of APS method is more important than MPS and PS.

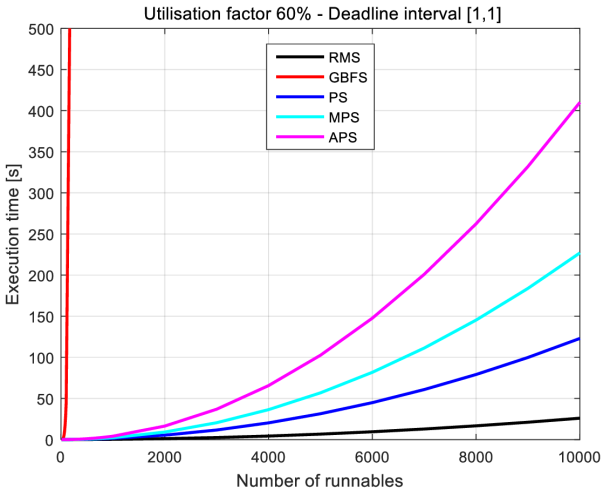


Fig. 7. Execution time for different algorithms.

C. Number of tasks

In this section, we compare numbers of tasks obtained by all algorithms (RMS, GBFS, PS, MPS, and APS) and for two different configurations. The first configuration presents the evolution of task number depending on the number of periods. The second configuration shows the influence of stringent

deadlines on the number of tasks. For all configurations, a utilization factor of 60% is used with a set of 100 runnables.

For the first configuration, we generate randomly 10 sets of runnables with a deadline on request for each set of periods given in the Table II. We then successively run five algorithms

TABLE II
FIVE SETS OF PERIODS USED.

# period	set of periods
5	{10, 20, 40, 80, 160}
10	{10, 20, 40, 80, 160, 15, 30, 45, 60, 90}
15	{10, 20, 40, 80, 160, 15, 30, 45, 60, 90, 25, 50, 75, 100, 125}
20	{10, 20, 40, 80, 160, 15, 30, 45, 60, 90, 25, 50, 75, 100, 125, 35, 70, 105, 140, 175}
25	{10, 20, 40, 80, 160, 15, 30, 45, 60, 90, 25, 50, 75, 100, 125, 35, 70, 105, 140, 175, 55, 110, 165, 220, 275}

on all runnables sets and then we determine the maximum number of tasks among the 10 sets of runnables (Figure 8). As expected, algorithms based on the PS solution (RMS, GBFS, and PS) deliver a number of task equals to number of different periods. For MPS and APS solutions, obtained number of tasks is much smaller than the number of periods.

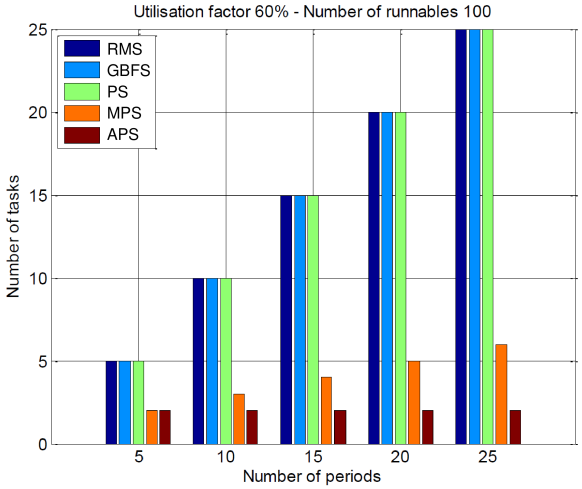


Fig. 8. Number of tasks depending on the number of periods.

In the second configuration, we run algorithms on 10 sets of runnables randomly generated for a set of periods of cardinality 20 (see table II). For each deadline interval $[a, b]$ and each algorithm, we take the maximum number of tasks among the 10 sets of runnables.

Figure 9 shows the influence of deadline on the number of tasks. Note that our algorithms find the required number of tasks while guaranteeing system schedulability even in very stringent deadlines situations. In particular, APS solution delivers a minimum number of tasks and has a very low variance by tightening the deadline. As a result, the maximum number of tasks does not exceed the cardinality of the set of periods. On the other hand, in the solution PS, the variance is higher, and the number of tasks may be twice the number of periods. This has a major influence on the system stack memory. For example, suppose that all runnables consume the same size of stack memory of 512 $bytes$. The stack of task is the maximum size of all runnables mapped to this task. Thus, stack memory used to implement PS solution for a deadline interval $[0, 0.5]$ is 47 $tasks * 512$ $bytes \approx 24$ kb. However, in APS solution, it is 8 $tasks * 512$ $bytes \approx 4$ kb.

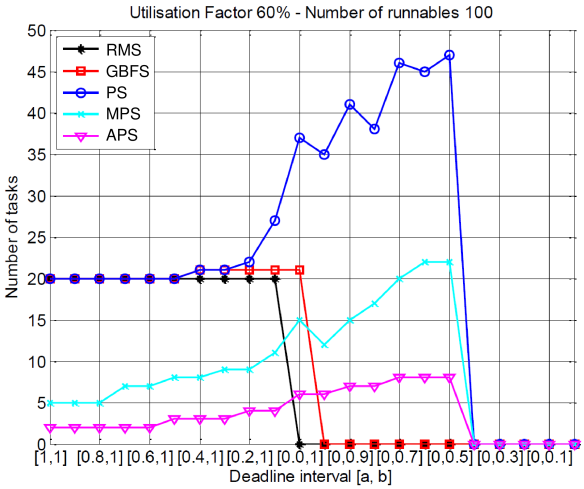


Fig. 9. Number of tasks depending on the deadline.

D. Average response time

This section evaluates average response time of algorithms depending on deadline. We use the following equation to calculate rate of average response time TRm for each set of runnables with a cardinality n:

$$TRm = 100 * \frac{\sum_{i=1}^{n} \frac{R_i}{d_i}}{n} \quad (10)$$

where R_i and Di are respectively worst response time and relative deadline of runnable r_i. We consider 10 sets of runnables. Each set contains 100 runnables generated randomly with a utilization factor of 60% and a set of periods $T = \{10, 15, 25, 35, 20, 30, 50, 70, 40, 45, 75, 105, 80, 60, 100, 140, 160, 90, 125, 175\}$.

For each deadline interval $[a, b]$, we run all algorithms on 10 sets of runnables and we determine the maximum of average response time. Obtained results are shown in Figure 10.

We note that algorithm APS finds a mapping of runnables to tasks that give better response times than those found by other heuristics. This is due to the tuning of runnables offsets. Nevertheless, MPS algorithm gives highest response times although it uses fewer tasks compared to PS, RMS, and GBFS algorithms due to the lack of use of runnables offsets. Further, we note that deadlines start to be tightened after the interval $[0.2, 1]$, where our algorithms continue to be schedulable with a gradual increase in response time up to interval $[0, 0.5]$.

VI. CONCLUSION

For AUTOSAR software design methodology, mapping of runnables to tasks represents a crucial design decision. This may affect both system performance and real-time execution. Several research works have addressed this problem ranging

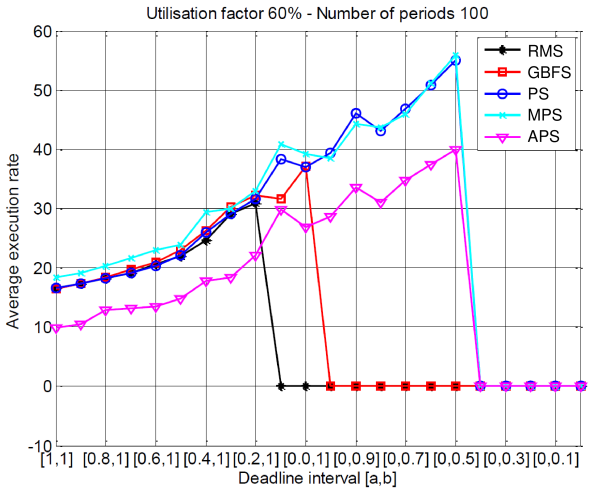


Fig. 10. Average execution rate of runnables depending on deadline.

from a simplest solution of mapping runnables having same periods (PS) or multiple periods (MPS) to same task, up to more advanced solution with mapping runnables having arbitrary periods (APS) to same task. Three different algorithms have been developed in this work whose main goal is construction of a minimum set of tasks on which a set of runnables is mapped. Task creation process goes through two stages: First, a feasibility test of mapping is applied to search a set of runnables that is feasible to map in the same task. If this set is empty, the system is declared as not schedulable. Second, a task is created by selection of a set of runnables given by the first step, using one of three main methods: PS, MPS, or APS.

Developped algorithms increase system schedulability by 23.81% when compared to conventional RMS algorithm, and by 14.28% when compared to GBFS algorithm. APS-based algorithm reduces significantly the number of tasks and the average response time in comparison to other algorithms. Further, in case of tight deadlines situation, task number increases up to three times the number of tasks where deadlines are relaxed.

REFERENCES

[1] Autosar consortium web page. http://www.autosar.org. accessed: 03/12/2015.

[2] Specification of rte autosar release 4.2.2. http://www.autosar.org/fileadmin/files/releases/4-2/software-architecture/rte/standard/AUTOSAR/_SWS/_RTE.pdf. accessed: 03/12/2015.

[3] Antoine Bertout, Julien Forget, and Richard Olejnik. A heuristic to minimize the cardinality of a real-time task set by automated task clustering. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, SAC '14, pages 1431–1436, New York, NY, USA, 2014. ACM.

[4] Enrico Bini and Giorgio C Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005.

[5] Priyanshi Gupta, N. P. Singh, and Geetha Srinivasan. An efficient approach for mapping autosar runnables in multi-core automotive systems to minimize communication cost. *2019 Innovations in Power and Advanced Computing Technologies (i-PACT)*, 1:1–4, 2019.

[6] Zeng Haibo, M. Di Natale, and Zhu Qi. Optimizing stack memory requirements for real-time embedded applications. In *Emerging Technologies and Factory Automation (ETFA), 2012 IEEE 17th Conference on*, pages 1–8, 2012.

[7] F. Khenfri, K. Chaaban, and C. Maryline. A novel heuristic algorithm for mapping autosar runnables to tasks. In *Pervasive and Embedded Computing and Communication Systems (PECCS), 2015 5th International Conference on*, pages 239–246, 2015.

[8] Kyung-Jung Lee, Jae-Woo Kim, H. Chang, and Hyun-Sik Ahn. Mixed harmonic runnable scheduling for automotive software on multi-core processors. *International Journal of Automotive Technology*, 19:323–330, 04 2018.

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

[10] Zhang Ming and Gu Zonghua. Optimization issues in mapping autosar components to distributed multithreaded implementations. In *Rapid System Prototyping (RSP), 2011 22nd IEEE International Symposium on*, pages 23–29, 2011.

[11] A. K. Mok and D. Chen. A multiframe model for real-time tasks. *IEEE Transactions on Software Engineering*, 23(10):635–645, Oct 1997.

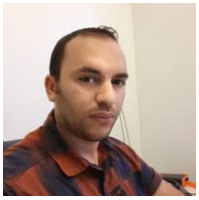
[12] A. Monot, N. Navet, B. Bavoux, and F. Simonot-Lion. Multisource software on multicore automotive ecus combining runnable sequencing with task scheduling. *Industrial Electronics, IEEE Transactions on*, 59(10):3934–3942, 2012.

[13] Long Rongshen, Li Hong, Peng Wei, Zhang Yi, and Zhao Minde. An approach to optimize intra-ecu communication based on mapping of autosar runnable entities. In *Embedded Software and Systems, 2009 ICESS '09 International Conference on*, pages 138–143, 2009.

[14] E. Wozniak, A. Mehiaoui, C. Mraidha, S. Tucci-Piergiovanni, and S. Gerard. An optimization approach for the synthesis of autosar architectures. In *Emerging Technologies and Factory Automation (ETFA), 2013 IEEE 18th Conference on*, pages 1–10, 2013.

[15] Haibo Zeng and M. Di Natale. Efficient implementation of autosar components with minimal memory usage. In *Industrial Embedded Systems (SIES), 2012 7th IEEE International Symposium on*, pages 130–137, June 2012.

[16] Qingling Zhao, Zonghua Gu, and Haibo Zeng. Design optimization for autosar models with preemption thresholds and mixed-criticality scheduling. *Journal of Systems Architecture*, 72:61 – 68, 2017. Design Automation for Embedded Ubiquitous Computing Systems.



Fouad Khenfri received an engineering degree from the University of Biskra, Biskra, Algeria, in 2008, and a M.Sc. degree in electrical and computer engineering from the Polytechnic School of Algiers, Algeria, in 2011, and a Ph.D. degree in computer science and automation from Nantes Central School (Ecole Centrale de Nantes), France, in 2016. He is currently an Assistant Professor at ESTACA (Ecole Supérieure des Techniques Aéronautiques et de Construction Automobile), France, since 2016. His current research interests include system control, embedded software design, and embedded system performance analysis and optimization.

Khaled Chaaban received an engineering degree in computer engineering/telecommunication from the Lebanese University, Lebanon (2001), a master's degree in "Information and Systems Technology" in 2002 followed by a PhD degree in 2006 from UTC University, Compiegne (France). He worked at ESTACA engineering school (2007-2015) as an associate professor before joining UMM AL-QURA University since 2015. His main research interests include scheduling for real-time applications and design space exploration. He has published more than 30 journal articles and conference papers in the area of embedded and real-time systems.

Maryline Chetto received the degree of PhD in control engineering and the degree of HDR (Habilitée à Diriger des Recherches) in Computer Science from the University of Nantes, France, in 1984 and 1993, respectively. From 1984 to 1985, she held the position of Assistant professor of Computer Science at the University of Rennes, while her research was with the (Institut de Recherche en Informatique et Systèmes Aléatoires), Rennes. In 1986, she returned to Nantes and is currently a professor with the Institute of Technology of the University of Nantes. She is conducting her research at IRCCyN. Her main research interests include scheduling and energy management for real-time applications. She has published more than 100 journal articles and conference papers in the area of real-time operating systems.