Assigning Time Budgets to Component Functions in the Design of Time-Critical Automotive Systems

Ernest Wozniak
CEA LIST, France and McGill University, Canada
ernest.wozniak@cea.fr

Marco Di Natale
Scuola Superiore Sant'Anna, Italy
marco@sssup.it

Haibo Zeng
McGill University, Canada
haibo.zeng@mcgill.ca

Chokri Mraidha
CEA LIST, France
chokri.mraidha@cea.fr

Sara Tucci-Piergiovanni
CEA LIST, France
sara.tucci@cea.fr

Sebastien Gerard
CEA LIST, France
sebastien.gerard@cea.fr

ABSTRACT

The adoption of AUTOSAR and Model Driven Engineering (MDE) for the design of automotive software architectures allows an early analysis of system properties and the automatic synthesis of architecture and software implementation. To select and configure the architecture with respect to timing constraints, knowledge about the worst case execution times (WCET) of functions is required. An accurate evaluation of the WCET is only possible when reusing legacy functionality or very late in the development and procurement process. To drive the integration of SW components belonging to systems with timing constraints, automotive methodologies propose to assign WCET budgets to functions. This paper presents two solutions to assign budgets, while considering at the same time the problem of SW/HW synthesis. The first solution is a one-step algorithm. The second is an iterative improvement procedure with a staged approach that scales better to very large size systems. Both methods are evaluated on industrial systems to study their effectiveness and scalability.

Keywords

design space exploration, real-time, end-to-end deadlines, time budgeting, AUTOSAR, optimization, genetic algorithm

1. INTRODUCTION

Vehicles are today very complex high-technology products with an increasing number of software features. This evolution is not matched by quality and speed improvements in the software development processes. The AUTOSAR [1] automotive standard has been defined to improve the design and integration of automotive SW components and the automatic generation of architecture features. The AUTOSAR methodology [3] makes use of model-driven engineering principles (MDE), i.e. the use of models as basic building blocks for software and hardware description in a top-down process, in which the top-most level consists of the specification of a platform-independent software architecture (application components and their connections) and a separate hardware model (defining the network of nodes and buses). In a refinement process, the functional components are mapped to the hardware nodes, generating the node configuration and the definition of the OS tasks. The result of the mapping is the application deployment, i.e., a set of communicating tasks running on the nodes and communicating through messages sent on buses.

The AUTOSAR methodology is defined according to the typical automotive supply chain process. The integrator (car maker) has control over the functional architecture design at the system-level, and selects the execution platform, including the (distributed) HW architecture and the basic SW (RTOS, middleware, communication stack, drivers). The integrator defines the SW components that are needed for the implementation of the functionality and sends the AUTOSAR specifications of their interfaces and behaviors to the suppliers. Starting from version 4.0, timing constraints become part of the AUTOSAR standard [2]. This requires *the integrator to validate the correctness of the architecture solution with respect to time constraints (end-to-end deadlines) and define the (partitioned) timing constraints that apply to the component specifications for the suppliers.*

The validation of timing properties during the development is a continuous process. Since refinements are top-down, deadlines can only be validated under assumptions abstracting the lower-level details, such as the worst-case execution times (WCETs) of functions. In AUTOSAR, functions are called *runnables* and represent the atomic executable entities defining the internal behavior of components. An application deployment is valid if and only if the set of tasks that execute the runnables is schedulable. Unless the implementation of runnables is re-used from previous systems, precise knowledge of WCETs of runnables is not available before code implementation in the design process. To define a system architecture supporting functions with time constraints and provide the specification of its components to the suppliers, automotive developers and domain experts propose a **time budgeting** activity as part of the development methodology. The system integrator specifies and assigns **time budgets**. These time budgets are *constraints* on worst-case execution time that must be respected by the suppliers delivering the component implementation [4]. If all the delivered components fulfill their local constraints, the end-to-end deadlines are satisfied. In the meantime, based on the budget *assumptions*, the system integrator can synthesize and optimize an architecture configuration.

1.1 Related Work

The problem of defining time budgets for components and runnables is affine to the issue of end-to-end deadline partitioning. Several research works have investigated the option of partitioning the end-to-end deadline into time windows or intermediate deadlines, upon the assumption that the interaction model allows the composition of the local response times to compute end-to-end response times. A graph-based algorithm for deadline partitioning to maximize the minimum slack is presented in [10], and an approach for periodic processes in [14]. More recently, deadline partitioning schemes for transaction chains scheduled under EDF or fixed priority can be found in [24, 16, 17]. Other efforts have been specifically tailored to automotive architectures. The TIMMO-2-USE project [25] discusses the need for time budgeting in the context of the process stages dedicated to the refinement of the system architecture. The project deliverables discuss a set of guidelines for budgeting the worst-case response times (WCRTs), based on the designer experience and do not provide a specific algorithm. Scheickl et al. [23] considers a similar process in which the definition of the WCRT budgets is based on the experience of the designer. Similarly, WCRTs are budgeted in [13], with a discussion on how different activation patterns (event- or time-driven) influence the specification of time budgets. Nevertheless, as in the previous two works, the approach relies on the experience of the designer to specify the time budget values.

The methodology of partitioning deadlines on response times is more suitable to the concept of federated automotive architectures [9], when suppliers provide hardware units or ECUs with operating systems and tasks, or at the very least when the responsibility of the task design is delegated to the suppliers. In the new concept of **integrated architecture**, enabled by AUTOSAR, the definition of the tasks and the design of the hardware architecture pertains to the integrator. Therefore, budgeting should be performed at the level of the WCET of the runnables.

The concept of **time budgeting** in the integration of automotive systems is among the research topics of the ALL TIMES project [4]. The approach proposed in the project deliverables takes as an input an already deployed architecture, i.e. the software components are already assigned to tasks and mapped onto the hardware platform. Since WCETs are not known, the deployment choices (assumed as predetermined and not subject to optimization) could very well be suboptimal and affect the final result (the assigned budgets). In [12] the budgeting problem is formulated and solved by applying Parametric Linear Temporal Logic (PLTL). A method is presented to automatically decompose end-to-end deadlines into a set of time budgets. The authors automatically compute a set of linear constraints for which they finally find a valuation (using a solver) that guarantees all deadlines and maximizes the values of the time budgets. The proposed solution also integrates the consideration of non-functional properties related to the ECU utilization [11]. As in all previous works, the authors assume that the deployment, i.e. the integration of the software architecture with the hardware platform and the design of the task, is already known. In addition, the deployment choices could not have been done in a qualitative way as WCETs for certain runnables were missing.

1.2 Our Contributions

In this paper we incorporate the problem of budgeting worst-case execution times in the design optimization. All the other works budget the worst case response time which does not fit well to the idea of the integrated architecture as supported by the AUTOSAR. Different from previous work [4] on WCET budgeting, we assume that the function deployment is not known in advance. Accordingly our objective is to jointly find the deployment of functions and the optimal assignment of time budgets given a functional (AUTOSAR) model with end-to-end timing constraints and an execution (hardware+OS) platform. Since time budgeting and deployment are both NP-hard problems, our solutions consists of two heuristic approaches, based on evolutionary algorithms. The first provides a holistic one-step solution to the problem, whereas the second divides it into two subproblems solved one after the other. On the other hand, as shown in the experiments, due to the scalability issues, it is impractical to employ the time budgeting algorithm as specified in [4] with our algorithm exploring the deployment.

The paper is organized as follows: section 2 formalizes the model and problem considered by the approach; section 3 details the proposed technique for time budgeting and section 4 presents its improvement in a form of staged approach; section 5 gives experimental evaluation results; finally, section 6 sketches some future work and concludes the paper.

2. SYSTEM MODEL

This section starts by introducing basic definitions and assumptions on the system model. Then, the time budgeting problem is formalized together with constraints on timing and deployment.

2.1 System Model

The input system model matches the modeling abstractions in AUTOSAR. The AUTOSAR **functional model** consists of a set of components. Each component has a set of *runnables* defining its functional behavior and cooperating by exchanging data and synchronization signals over ports. Of the possible interaction models allowed by AUTOSAR, we focus on a model of *synchronous transactions*, in which runnables exchange activation signals in a linear sequence. In addition, even if they are extremely important in the supply process and as an abstraction mechanism, in the context of our analysis, components are only containers of runnables and are therefore omitted.

We assume the runnable interaction model is represented as a system-level graph. $G_e = \{V_e, E_e\}$, in which $V_e = \{r_1, r_2, ..., r_n\}$ is the set of vertices representing the runnables and $E_e = \{s_1, s_2, ..., s_m\}$ is the set of edges or links between them representing data signals. The length of a signal s_i is denoted as l_{s_i} (in number of bits). The set of transactions is $\xi = \{\Gamma_1, \Gamma_2, ..., \Gamma_l\}$. Each transaction is an ordered interleaving sequence of runnables and signals $\Gamma_i = [r_{i_1}, s_{i_1}, r_{i_2}, s_{i_2}, ..., s_{i_{k-1}}, r_{i_k}]$ (e.g. from the runnable reading sensor data to the runnable communicating with the actuator). $src(\Gamma_i) = r_{i_1}$ is the transaction source, and $snk(\Gamma_i) = r_{i_k}$ its sink. The function $\Gamma(r_i)$ returns the transaction to which r_i belongs. We assume that transactions are linear and the activation pattern is event-triggered, i.e. each transaction is triggered by a periodic event of period P_{Γ_i}. This event is then propagated by the runnables, starting from the $src(\Gamma_i)$ through their communication links (each connection carries an activation event together with the associated data signal). The response time of the transaction Γ_i is R_{Γ_i} and its deadline D_{Γ_i}. Figure 1 shows an example model (based on the cruise control case study from [5]).

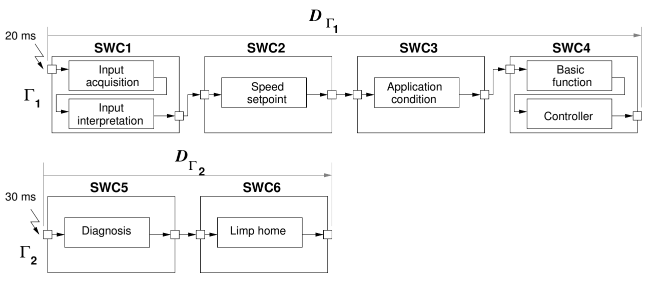


Figure 1: An example of linear transactions for a Cruise Control [5]

The **execution platform** is represented by an undirected graph $G_h = \{V_h, E_h\}$. Nodes are computing resources or (ECUs) $V_h = \{e_1, e_2, ..., e_s\}$ and edges represent the communication links (buses) between them $E_h = \{b_1, b_2, ..., b_p\}$. The function $E(b_i)$ returns the set of ECUs connected to b_i.

Runnables are partitioned in two sets. RW is the set of legacy runnables that are simply reused, and for which an estimate of the WCETs on the available ECUs is known. RB is the set of runnables under development, for which a budget assignment must be provided.

The WCET of a generic runnable r_i belonging to RW is represented as a vector $\vec{C}_{r_i} = (C_{r_i,1}, C_{r_i,2}, ..., C_{r_i,n})$, where $C_{r_i,k}$ is the WCET of r_i when executed on e_k. The (execution) time budget of a runnable r_i belonging to RB is denoted as tb_{r_i}.

In the **deployed architecture**, represented with Ψ, the code of each runnable entity executes in a context of an OS task. $T = \{\tau_1, \tau_2, ..., \tau_t\}$ is the set of tasks. Similarly, the signal needs to be transmitted in one of the messages $M = \{m_1, m_2, ..., m_u\}$, if the communication is between runnables mapped to different ECUs; otherwise, it is considered as local communication. The function $\tau(r_i)$ returns the task on which runnable r_i is partitioned and $m(s_i)$ the message transmitting the signal s_i. The function $e(r_i)$ returns the ECU on which r_i is allocated whereas $b(s_i)$ is the bus through which s_i is sent. Tasks and messages are assumed to be scheduled according to a fixed-priority mechanism or transmitted in the order of their priority, respectively. The automotive standards, e.g. AUTOSAR OS and Controller Area Network (CAN), are significant examples of such policies. The priority of a task τ_i (message m_i) is denoted as π_i (unless otherwise noted, it will also be assumed that they are indexed according to their priority, where a lower index means a higher priority). Considering the possible bit-stuffing of the CAN protocol, the worst-case transmission time (WCTT) of a message m_i is calculated as [8]

$$C_{m_i} = (g + 8l_{m_i} + 13 + \left\lfloor \frac{g + 8l_{m_i} - 1}{4} \right\rfloor)\tau_{bit} \qquad (1)$$

where g is the number of protocol bits subject to bit-stuffing ($g = 34$ for standard format with 11-bit identifiers, or $g = 54$ for extended format with 29-bit identifiers), l_{m_i} is the data length of the message (**bytes** unit), and τ_{bit} is the transmission time for a single bit. The data length of message m_i is the least integer number of bytes between 0 and 8 that is greater than the sum of the data signals packed in m_i,

$$l_{m_i} = \left\lceil \frac{\sum_{s_j : m(s_j) = m_i} l_{s_i}}{8} \right\rceil \qquad (2)$$

Figure 2 shows one possible partitioning of runnables into tasks and assignment of tasks onto ECUs for the functional model of Figure 1. In the figure, some signals exchanged between runnables are mapped onto messages (e.g. the one between "Speed setpoint" and "Application condition").

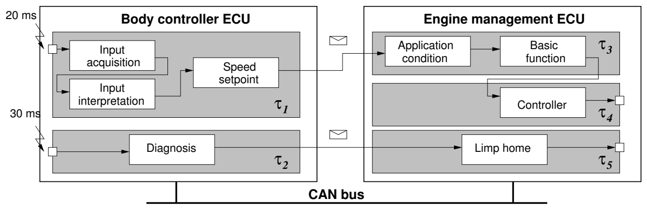


Figure 2: Example of a deployment configuration for a Cruise Control

2.2 Problem Formulation

The problem tackled in this work consists of four sub-problems. The first is the **(1)** ***placement*** of runnables on execution nodes and signals on buses (assigning values to $e(i)$ and $b(i)$). The second sub-problem is the **(2)** ***partitioning***, i.e. the definition of the operating system tasks (set T) and messages (set M) and the mapping function that assigns runnables and signals to them. The next sub-problem is the **(3)** ***scheduling*** or assignment of priorities to tasks and messages. Lastly, the fourth sub-problem is the **(4)** ***time budgeting***, which concerns the definition of the time budget values, i.e. finding tb_{r_i} for each $r_i \in RB$.

These four problems have significant cross-dependencies. Ideally, they should be solved as an integral problem, but this could be very challenging in terms of the computational effort that is required. Alternatively, the problem can be solved in stages, with the possibility of early choices restricting the set of available decisions for later stages. In our work, we try to lessen this problem by wrapping the staged solution in an iteration loop, in which the first stage is performed several times trying to improve on the results of the previous cycle.

In the remainder of the section, we define the set of constraints and the optimization objective.

2.3 Deployment Constraints

In order for the placement, partitioning and scheduling to be correct several constraints need to be respected to guarantee the functional correctness of the deployed architecture. We list them according to their purpose without a formal description (which is straightforward for all of them).

Placement Constraints:

- resources utilization - defining a maximum capacity utilization for all execution nodes and buses;
- each runnable/signal can be placed only on one ECU/bus;
- fixed placement - the placement of selected runnables is constrained to a subset of all ECUs (e.g. a runnable responsible for collecting sensor data must be placed only on ECUs linked to that sensor);
- software components constraint - all the runnables of the same atomic software component need to be placed on the same ECU;
- each global signal (i.e. communicated between two runnables mapped on different ECUs) must be transmitted over a bus connecting the ECUs where its sender and receiver runnables/tasks are allocated.

Partitioning Constraints:

- harmonic rate - two runnables/signals with non-harmonic periods cannot be partitioned in the same task/message;
- functional partitioning - each runnable must be partitioned exactly in one task. Likewise, each global signal must be partitioned in one message.

Scheduling Constraints:

- local order - tasks/messages allocated on the same ECU/bus must be assigned with unique priorities;
- runnables order - the priorities assigned to tasks must be consistent with the order of execution of the runnables partitioned on them. That is, if runnables r_i and r_j are on the same ECU and runnable r_i precedes runnable r_j then r_i should be partitioned in a task with priority higher than that of the task executing r_j.

2.4 Timing Constraints

All transactions must complete within their end-to-end deadlines

$$\forall \Gamma_l \qquad R_{\Gamma_l} \leq D_{\Gamma_l} \tag{3}$$

The verification of this constraint requires the evaluation of the end-to-end response time R_{Γ_l} of each transaction Γ_l, which equals the WCRT of its sink runnable r_i, i.e. $R_{\Gamma_l} = R_{r_i}$. We use the response time analysis with jitter propagation given in [26] and reformulate it to consider runnables (see Equations (4)-(5)), where J_{r_i} is the release jitter of the runnable, $W_{r_i}^{(q)}$ is the completion time of the q^{th} instance of runnable r_i. The function $hp(r_i)$ returns the indexes of all the runnables allocated on the same ECU with a priority higher than that of r_i. The completion time is computed for $q = 1, 2, ...$ until the busy period ends, that is, an instance completes at or before the activation of the next instance.

$$W_{r_i}^{(q)} = q(C_{r_i} + \sum_{r_j:\tau(r_i)=\tau(r_j)} C_{r_j}) + \sum_{r_j \in hp(r_i)} \left\lceil \frac{W_{r_i}^{(q)} + J_{r_i}}{P_{r_j}} \right\rceil C_{r_j} \tag{4}$$

$$R_{\Gamma_l} = R_{r_i} = \max_{q=1,2,\ldots} [W_{r_i}^q - (q-1)P_{r_i} + J_{r_i}] \tag{5}$$

The jitter of runnable r_i inherits the response time of the signal s_{i-1} sent by its direct predecessor in the transaction

$$J_{r_i} = R_{s_{i-1}} \tag{6}$$

If a signal s_k is local, its response time is the same as its sender runnable r_{k-1}

$$R_{s_k} = R_{r_{k-1}} \tag{7}$$

Otherwise, the WCRT of s_k is computed as the response time of the message $m_p = m(s_k)$ carrying this signal

$$R_{s_k} = R_{m_p} \tag{8}$$

To calculate the response time of message m_p, a set of formulae similar to (4)-(5) is used, except for 1) an additional term B_{m_p} that represents the blocking time caused by the impossibility of preempting lower priority messages (as in a CAN bus); 2) a reduction in the queuing delay W_{m_p} since m_p cannot be preempted. We use the approximated sufficient formulation provided in [8], in which B_{m_p} is the longest transmission time (largest WCTT) of any message that shares the same communication bus (or simply the longest 64-bit message in the CAN bus). $W_{m_p}^{(q)}$ is the start time of the q-th instance of message m_p.

$$W_{m_p}^{(q)} = B_{m_p} + (q-1)C_{m_p} + \sum_{m_j \in hp(m_p)} \left\lceil \frac{W_{m_p}^{(q)} + J_{m_p}}{P_{m_j}} \right\rceil C_{m_j} \tag{9}$$

$$R_{m_p} = \max_{q=1,2,\ldots} [W_{m_p}^q - (q-1)P_{m_p} + J_{m_p} + C_{m_p}] \tag{10}$$

The queuing jitter of m_p is the largest among all the jitters of signals packed in m_p. The jitter J_{s_k} of a signal s_k equals the response time of a runnable r_{k-1} that sends the signal s_k, i.e. $J_{s_k} = R_{r_{k-1}}$.

$$J_{m_p} = \max_{s_k:m(s_k)=m_p} J_{s_k} \tag{11}$$

2.5 Optimization Objective

We consider an optimization metric expressing the relaxation of time budgets within the end-to-end deadline constraints. The function $f_{tb}(TBA)$ in (12) requires as an input the set TBA of runnables with the specific valuation for their time budgets. It is defined as the minimum time budget value for all runnables in RB normalized with respect to the target range $(tb_{r_i}^m, tb_{r_i}^M)$. The optimization objective is to maximize $f_{tb}(TBA)$, or equivalently, to maximize the minimum normalized time budget among runnables in RB.

$$f_{tb}(TBA) = \min_{r_k \in RB} \frac{tb_{r_k} - tb_{r_k}^m}{tb_{r_k}^M - tb_{r_k}^m} \tag{12}$$

The designer has the option to provide a minimum value for tb_{r_i} as $tb_{r_i}^m$ (its intuitive meaning is a preliminary evaluation of the minimum required execution time for the functionality, based on the experience of the designer). If it is not specified, then $tb_{r_i}^m = 0$. $tb_{r_i}^M$ is the maximum time budget for the runnable r_i. If not set by the designer, it is assigned with the period of the transaction to which r_i belongs.

Providing for a time budget that allows for additional slack time mitigates the design risks associated with uncertainties about the execution time of the runnable implementation delivered by the supplier. The authors of [22] propose a method to derive a certainty of obtaining a feasible system configuration under the assumption of uncertain design parameters such as WCET of new runnables. They define an uncertainty function that enables the system integrator to estimate the risk of obtaining an infeasible design and consider it in the definition of the contract with a supplier.

Indeed, relaxation of budget values lies in the interest of the [4]. [4] employs the binary sensitivity analysis [15] which searches for an upper bound of the runnable execution time so that the system remains schedulable. The proposed algorithm is designed to consider the relaxation of only one runnable, which is why the metric of interest does not need to be normalized. In our case, relaxation should affect all the runnables in RB. [21] accounts for more than one runnable, by simply returning a schedulability region of all the possible combinations of the time budgets for runnables in RB. Our proposed metric in (12) instead targets at only the best combination, i.e. one that equally distributes the budget constraints among different suppliers delivering implementation of runnables from RB.

3. A ONE-STEP SOLUTION

To solve the problem of time budgeting defined in the previous section, we first consider a one-step approach and a solution based on a Genetic Algorithm (GA) and MILP (Mixed Integer Linear Programming). A genetic algorithm is used to find solutions for the deployment problem [18], which includes placement, partitioning, and scheduling (the first three sub-problems defined in section 2.2). The GA implementation for this particular problem requires the specification of the *encoding* and *evolution* operators along with the fitness function, and stop condition. These issues are addressed in sections 3.1–3.4.

For a given deployment solution, the time budgets of runnables (the fourth sub-problem in section 2.2) are assigned using Algorithm 1, which has as inputs the maximum value of runnable time budgets. The maximum value of time budgets is formulated and computed by a MILP framework. These algorithms are described in sections 3.5 and 3.6.

3.1 Encoding

Each solution to the problem is encoded as a chromosome ch_j, representing a specific deployment configuration. This work uses the value encoding, in which each gene g_i (subset of bits) in ch_j contains a specific value. A gene relates to either a runnable entity or a data signal. For a runnable gene, $g_i = ch_j(r_k)$ stores the value $V_{r_k}(g_i)$ representing the allocation and partitioning of r_k, that is, the index of the ECU on which it is allocated and the index of the task (also representing its priority) in which it is partitioned. $NR^{\max}$ denotes the maximum number of runnables that can be hosted by any ECU without violating the utilization constraints (WCETs or $tb^m_{r_i}$ and periods of runnables are used for computing $NR^{\max}$). If runnable r_i is allocated to ECU e_m and partitioned into the task τ_l, its encoding $V_{r_k}(g_i)$ is computed according to the following Equation (13).

$$V_{r_k}(g_i) = (m-1)NR^{\max} + (l-1) \tag{13}$$

For a data signal, $V_{s_k}(g_i)$ depends on whether it is a global or local data signal. The gene for a global signal holds information about the bus and the message in which it is partitioned. For a local signal s_k the value will not have any meaning. The gene value for a data signal is computed in a similar way (14). $NS^{\max}$ is the maximum number of signals that can be transmitted over any BUS without violating the utilization constraints (WCTTs and periods of signals are used to compute $NS^{\max}$).

$$V_{s_k}(g_i) = (m-1)NS^{\max} + (l-1) \tag{14}$$

3.2 Evolution

The evolution of a population consists in the generation of new chromosomes (new solutions) using the *crossover* and *mutation* mechanisms, followed by the selection of chromosomes with the highest fitness values. The crossover operator is very important for the quality of the GA solution, which combines information from two parent chromosomes to generate new ones. In this work, we use the OX3 crossover operator [7] with a tournament selector [19] (with size equal to 5). The tournament selector with size 5 will first create two sets with 5 randomly chosen chromosomes. The most fit chromosome from each set will be taken and these two chromosomes will be used as parents for the crossover. Then the OX3 operator will randomly select the "crossover points", i.e. indexes of genes that will constitute the boundaries of the crossover operation. The values between these points are copied from the first/second parent to the second/first child in the same absolute position. The remaining values are copied from the first/second parent to the first/second child. The mutation operator randomly chooses a gene in a chromosome to change its value to a new value randomly selected among those that do not violate the constraints of a correct deployment (see section 2.3).

3.3 Fitness Function

The fitness function defines how much the solution optimizes the time budgeting optimization criterion. Chromosomes are ranked according to the result of the metric function. The higher the value, the higher the probability that the chromosome will be chosen as a parent for a crossover. The fitness function requires as an input the deployment specification, i.e. Ψ (encoded in the chromosome) and the time budgets assignment TBA. The time budgets assignment is obtained by using Algorithm 1.

3.4 Stop Condition

The stop condition determines when the GA will terminate. In our case, the algorithm stops when n iterations of the GA internal loop deliver a maximum fit result with the same fitness value (the best solution does not improve in n rounds). The value of n used in our experiments is 30.

3.5 Time Budgeting Algorithm

Within the GA optimization cycle, Algorithm 1 is executed for each chromosome to compute the corresponding optimum set of time budgets based on which value for the metric function $f_{tb}(TBA)$ can be calculated.

The *time budgeting algorithm* has four inputs: Ψ, RB, ϵ and M_{RB}. ϵ is the maximum error on the computed budgets that controls the terminating condition (line 15). The lower is the value of ϵ, the more accurate are the time budgets, and the larger is the runtime of the algorithm. M_{RB} is a set of upper bounds on the runnable budgets computed for a specific deployment Ψ, where $M_{RB}(i)$ is the maximum value for r_i. The values in M_{RB} are computed before running Algorithm 1, based on the end-to-end deadlines, utilization bounds, and the constraints $tb^m_{r_i}$ and $tb^M_{r_i}$. The formulation that is used to compute the bounds M_{RB} is discussed after the description of the time budgeting algorithm (section 3.6).

Algorithm 1 tries to relax the time budgets for all the runnables in RB according to the metric (12) using a binary search algorithm (as in the sensitivity analysis test in

Algorithm 1 Time Budgeting Algorithm

```
Require: $\Psi$, $RB$, $\epsilon$, $M_{RB}$
for all $r_j \in RB$ do
    $Utb_{r_j} = M_{RB}(j)$
    $Ltb_{r_j} = tb^m_{r_j}$
end for
if $isSchedulable(\Psi, M_{RB})$ then
    for all $r_j \in RB$ do
        $TBA.set(r_j, M_{RB}(j))$
    end for
    return $TBA$
else
    for all $r_j \in RB$ do
        $tb_{r_j} = (Utb_{r_j} - Ltb_{r_j})/2$
    end for
end if
while $\exists r_i \in RB \quad Utb_{r_j} - Ltb_{r_j} \geq \epsilon$ do
    for all $r_j \in RB$ do
        $TBA.set(r_j, tb_{r_j})$
    end for
    if $isSchedulable(\Psi, TBA)$ then
        for $r_j \in RB$ do $Ltb_{r_j} = tb_{r_j}$
    else
        for $r_j \in RB$ do $Utb_{r_j} = tb_{r_j}$
    end if
    $tb_{r_j} = Utb_{r_j} - (Utb_{r_j} - Ltb_{r_j})/2$
end while
return $TBA$
```

[15]). The upper bound values are tried first, giving the maximum possible value of $f_{tb}(TBA)$ (lines 6-8). If the corresponding configuration is schedulable (function *isSchedulable()* validates it by running schedulability analysis test), it is returned as the optimum value (line 9). If not, then the algorithm assigns to each r_j in RB a budget value that is the medium value between the minimum $tb^m_{r_j}$ and the upper bound $M_{RB}(i)$ (lines 11-13).

From this point on, Algorithm 1 continues by iteratively reducing the range of the time budgets, defined as $[Ltb_{r_j}, Utb_{r_j}]$ for runnable r_j. The algorithm works as a binary search. In each iteration, if the current budget values, at the midpoint between the the upper and lower bounds result in a schedulable solution, the upper bound Utb_{r_j} remains the same, and the lower bound Ltb_{r_j} is updated to be midpoint (line 20), and the range is reduced to be half of the size. If the current settings result in a unschedulable solution, it means that the time budget value is too large, and the next iteration will search wiithin the lower half of the range (line 22).

In [15], budget values are computed for each runnable separately, in a set of recurrent calls, exploring all the possible options for the relaxation of each individual runnable budget, at the price of higher complexity. However, for the metric (12) this is not required. Given any optimal solution according to (12), there exists another solution with the same value of (12) that is computed by our bisection algorithm, performing an equal relaxation of all time budgets (i.e. proportionally to $tb^m_{r_i}$ and $tb^M_{r_i}$). Of course, the solution computed by Algorithm 1 can have smaller budget values for those runnables that are not affecting the value of (12).

3.6 Calculating M_{RB} using MILP

Finally, the upper bounds M_{RB} that are required to reduce the initial interval of possible budget values in Algorithm 1, are computed using an MILP (Mixed Integer Linear Programming) formulation. The values in M_{RB} are (optimistic) upper bounds and do not guarantee the system schedulability as the constraints used for their computation are a linear approximation representing only a necessary schedulability condition that does not consider interference. However, they are useful in constraining the search space for the bisection algorithm.

In the MILP formulation, the problem is represented with parameters, decision variables, and constraints over the parameters and decision variables. Moreover, an objective function is defined to characterize the optimal solution.

Variables: the only set of variables are $M_{RB}(r_i)$ where $r_i \in RB$.

Objective function: the objective is to maximize the metric function in Equation (12), with $M_{RB}(r_i)$ in place of tb_{r_i}.

Constraints: three types of constraints are considered:

- Utilization constraints - a utilization bound applies to each ECU (u_{e_i}). Unless specified, the default limit value is 1.

$$\forall_{e_i \in E}, \sum_{r_j \in RB \wedge e(r_j)=e_i} \frac{M_{RB}(r_j)}{P_{\Gamma(r_j)}} \leq u_{e_i} - \sum_{r_j \in RW \wedge e(r_j)=e_i} \frac{C_{r_j,e_i}}{P_{\Gamma(r_j)}} \tag{15}$$

- Computation time constraints - are a linear under-approximation of the deadline constraints, ensuring that the sum of the execution times and budgets on each chain is lower than the deadline. These constraints do not consider interference and therefore do not guarantee end-to-end deadlines.

$$\forall_{\Gamma_i \in \xi}, \sum_{r_j \in RB \wedge \Gamma(r_j)=\Gamma_i} M_{RB}(r_j) \leq D_{\Gamma_i} - \sum_{r_j \in RW \wedge \Gamma(r_j)=\Gamma_i} C_{r_j,e_i} \tag{16}$$

- Minimum and maximum value constraints

$$\forall_{r_i \in RB} \quad tb^m_{r_i} \leq M_{RB}(r_i) \leq tb^M_{r_i} \tag{17}$$

4. STAGED APPROACH

The one-step holistic approach is simple and effective but does not scale to very large-size problems. Hence, an alternate solution was developed by dividing the four sub-problems (see section 2.2) in two stages. The first stage solves the first three sub-problems on deployment (including placement, partitioning and scheduling). The second stage tries to optimize the time budgeting only. The two stages are computed sequentially inside a loop until there is no further improvement as shown in Figure 3. The computation time savings derive from the execution of Algorithm 1 once for each iteration instead of once for each chromosome.

The staged algorithm implements an iterative improvement strategy that is in essence a local search. Starting from an initial solution, the current best solution is tentatively improved in the iterations of an inner cycle that includes the two optimization stages. If at any iteration, the two stages fail to produce a better result, the algorithm terminates and returns the best solution found until that point. The algorithm starts with the initialization of the variables storing the population of chromosomes (λ), and the current best metric values (the first two blocks from Figure 3). The second stage (second block in the figure) initializes the values of time budgets with their minimum values, i.e. $\forall_{r_i \in RB}, tb_{r_i} = tb^m_{r_i}$ (section 5.4). The rationale for this choice is that we do not want the algorithm (a local search) to end prematurely and we try to ease schedulability (and provide for maximum allocation freedom) as much as possible in the first step. In the experiments section, we discuss the impact of different values for the initial time budgets.

Next, the deployment optimization is performed considering the current values of the time budgets (the first time the loop is entered these are the initial budgets). During the deployment optimization stage, budgets are fixed, the metric function $f_{tb}(TBA)$ has a constant value, and cannot be used to evaluate the quality and drive the selection of the deployment solutions. Hence, the fitness function considered in this step is based on the end-to-end response times. The function $f_{slack}(\Psi, TBA)$, defined in Equation (18), expresses the goal of maximizing the minimum slack time (the difference between the deadline and response time) of each transaction.

$$f_{slack}(\Psi, TBA) = \min_{\Gamma_i \in \xi}[D_{\Gamma_i} - R_{\Gamma_i}] \qquad (18)$$

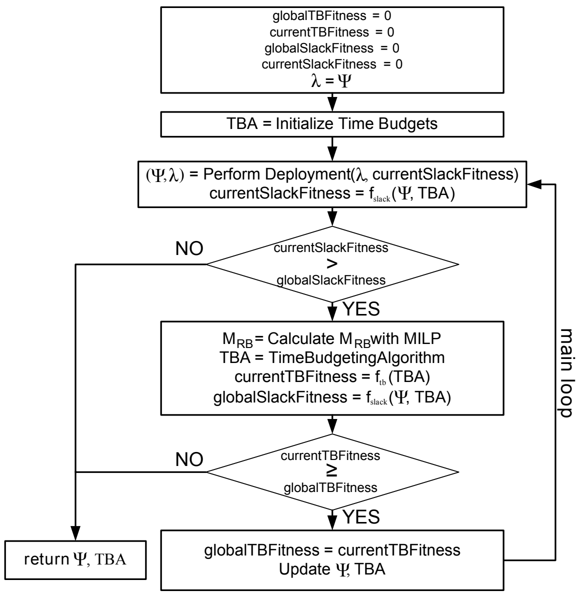
Maximizing the minimum slack means maximizing the minimum distance between the response time and deadline of any transaction, which is an indication of an opportunity for having larger budgets and hence a better deployment. The function $f_{slack}(\Psi, TBA)$ is computed based on the schedulability analysis formulas for computing the response times of runnables and messages (see section 2.4). In section 5, we discuss the results obtained when trying different metric functions at this stage.

At each iteration of the main loop, a new deployment solution is computed and then evaluated. If the value of $f_{slack}(\Psi, TBA)$ does not improve on the current best solution, i.e., the minimum slack is lower, the loop terminates and the best solution computed up to this point is returned. Otherwise, Algorithm 1 is executed to compute a new optimum set of time budgets (for the current deployment). Then, the fitness value $f_{tb}(TBA)$ is computed for the new set of time budgets and the new fitness value is compared with the current best. If the new deployment/budgets improve on the current best solution, they are considered for the next iteration, and the new budget drive the next deployment optimization step. Otherwise, the algorithm terminates and returns the best current deployment and time budget solution. The procedure is summarized in Figure 3.

Besides the different optimization metric used during the deployment, another significant difference with the one-step deployment algorithm described in section 3 is the stop condition. The loop terminates not only if no improvement is found after n internal GA iterations, but also when the fitness of the best chromosome from the new population is not better (lower or equal) than the current best fitness value ($currentSlackFitness \leq globalSlackFitness$). Finally, the set λ defines the initial population for the GA algorithm. At each iteration round, λ preserves the population selected in the previous run of the deployment algorithm. When the deployment is run for the first time and the set λ is empty, the deployment procedure initializes λ with a random initial set of chromosomes which is consistent with the constraints that apply to the system configuration.

5. EXPERIMENTAL EVALUATION

To evaluate the algorithms, a series of experiments have been performed on a collection of automotive case studies.



globalTBFitness = 0
currentTBFitness = 0
globalSlackFitness = 0
currentSlackFitness = 0
λ = Ψ
TBA = Initialize Time Budgets
(Ψ,λ) = Perform Deployment(λ, currentSlackFitness)
currentSlackFitness = f_slack(Ψ, TBA)
NO
currentSlackFitness > globalSlackFitness
YES
M_RB = Calculate M_RB with MILP
TBA = TimeBudgetingAlgorithm
currentTBFitness = f_tb(TBA)
globalSlackFitness = f_slack(Ψ, TBA)
main loop
NO
currentTBFitness ≥ globalTBFitness
YES
return Ψ, TBA
globalTBFitness = currentTBFitness
Update Ψ, TBA

Figure 3: Iterative improvement loop for the staged approach

First, a set of case studies is used to compare the one-step approach with the staged approach in terms of the quality of the results and the required execution time. Next, we discuss the robustness of the staged approach by evaluating the influence on the final results when replacing the metric in Equation (18) with a different function. Then, we present experiments to see if and by how much different initial assignments of time budget values affect the final result. Finally we compare with the work which is closest to ours, done in the AllTimes project [4]. All tests were run on a machine with 8GB of memory and a single processor running at 2.4GHz. Also, all the tests assume the maximum error factor $\epsilon = 0.5$ and the stop condition for the GA (regardless of initial population size) is that $n = 30$ consecutive iterations compute the same value.

5.1 An Automotive Case Study

We first apply our technique to an automotive case study constructed by merging two subsystems consisting of a CCS (Cruise Control System) [5] and ABS (Anti-lock Braking System) [20]. Figure 4 shows the input functional architecture together with the hardware topology. The functional model contains twelve runnables in four transactions with their deadlines and trigger periods. For five runnables, i.e., *Input acquisition, Input interpretation, Basic function, Diagnosis 1* and *Diagnosis 2*, the WCET information is not available and time budget must be assigned (they belong to the set RB). The other seven runnables are assumed as reused from legacy libraries and belong to the set RW. The hardware topology contains four ECUs, each connected to the single CAN bus. The WCETs of the runnables in RW are shown on Figure 4 and in Table 1. Table 1 also shows the minimum and maximum budget values for the runnables in RB.

The execution of the one-step optimization algorithm results in the deployment configuration shown in Figure 5.

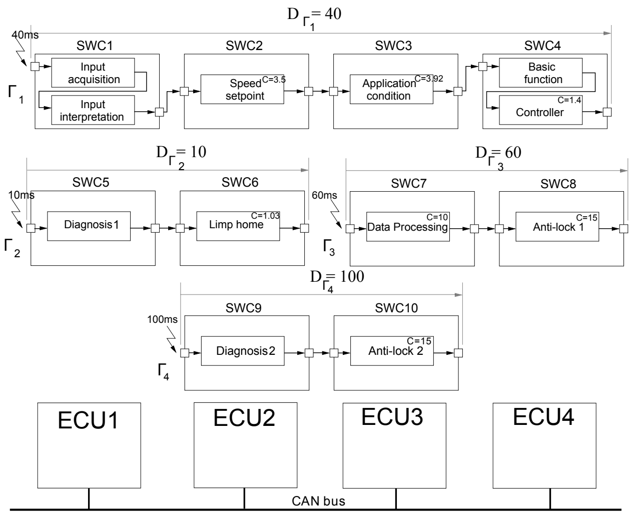


Figure 4: Input software and hardware architecture of combined CCS and ABS sub-systems

Table 1: Results for time budgets assignments and initial constraints

Runnable	$WCET$	tb^m	tb^M	tb	τ	ECU
Input acquisition	-	0	40	8.73	τ_1	1
Input interpretation	-	0	40	8.73	τ_1	1
Basic function	-	0	40	8.73	τ_1	1
Diagnosis 1	-	0	10	2.18	τ_1	4
Diagnosis 2	-	0	10	2.18	τ_2	1
Limp home	1.03	-	-	-	τ_1	4
Speed setpoint	3.5	-	-	-	τ_1	1
Application condition	3.92	-	-	-	τ_1	1
Controller	1.4	-	-	-	τ_1	2
Data processing	10	-	-	-	τ_1	3
Anti-lock 1	15	-	-	-	τ_2	2
Anti-lock 2	15	-	-	-	τ_2	4

For simplicity, the software components are omitted in the figure. The index of a task also represents its priority (the lower the value, the higher the priority). The task assignments and the computed time budget values are displayed together with the budgets constraints in Table 1.

5.2 One-step vs. Staged Approach

This subsection presents results of the comparison between the one-step holistic algorithm and the staged iterative approach. We examine and compare the quality of the solutions obtained with the two approaches, i.e. the final fitness values - the maximized $f_{tb}(TBA)$ and the runtimes that are required by the two algorithms.

For this purpose, the automotive case study has been extended with lower and higher complexity examples that have been generated starting from the 50-runnable case study (index 9 in our list) presented in [6] and extended to lower and higher sizes as described in [18]. Table 2 shows a summary of the fourteen system configurations by growing complexity. The table contains in the first column an index identifying the test case, in the second column the total number of runnables, in the third column the number of runnables for which budgets must be assigned, and, finally, the number of

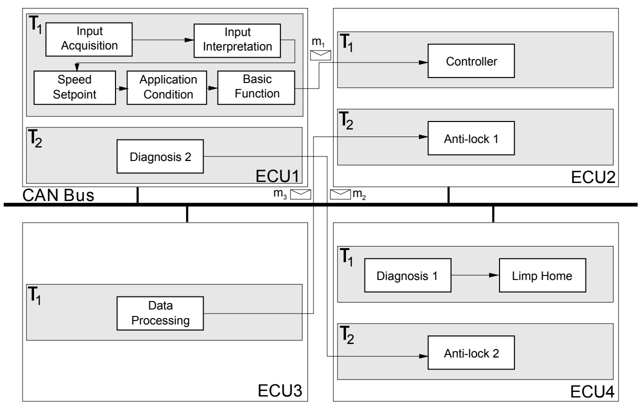


Figure 5: Deployment configuration for CCS and ABS

ECUs in the hardware architecture. In all these examples, we assume a single CAN bus connecting all ECUs. The original automotive case study is in the fourth row, with twelve runnables (as shown in the second column) and 5 of them in RB. Finally, as a further assumption, each software component has only one runnable entity. This means that for each runnable, its placement is independent of any other runnable's placement, as AUTOSAR requires that all runnables in the same component must be placed to the same ECU [1].

Table 2: Properties of the testing input architectures

Test nb	**Runnables**	$\lvert RB \rvert$	**ECUs**
1	5	2	2
2	6	2	2
3	10	4	4
4	12	5	4
5	16	6	6
6	20	8	8
7	32	12	9
8	40	14	9
9	50	17	9
10	60	20	9
11	70	25	10
12	80	27	12
13	90	30	16
14	100	35	18
15	200	70	36

5.2.1 *Quality evaluation*

Figure 6 shows the final fitness value of the best solution obtained by the one-step approach, compared with the one of the staged approach. The size of the initial population of the GA considered for these tests is 10000. The fitness value of the solutions computed by the two approaches for tests 1 to 6 is exactly the same, and also the same value was computed for test 10. For tests 7 to 9 and 11 to 13 the staged approach provided results slightly better than the one-step algorithm (in detail, 0.33%, 0.34%, 3.48% and 0.54%, 0.53%, 0.66% better, respectively). Finally, the one-step approach could not compute the final solution for case 14 after more than 24 hours of processing time. During this time, the GA internal loop performed 76 iterations. The best result obtained after

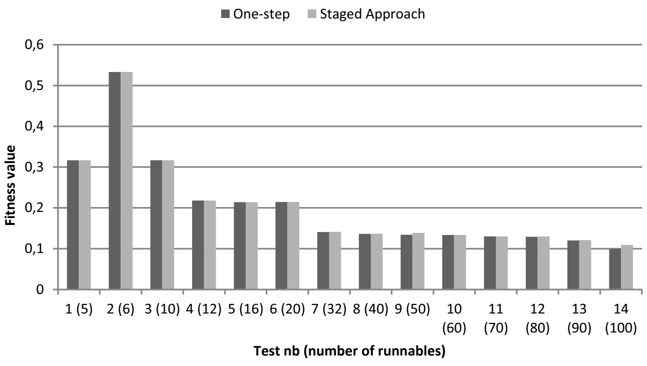


Figure 6: Results for One-step and Staged Approach (GA Initial Population = 10000)

this time was 8.75% worse than the final result computed by the staged approach (after 5.86 hours). We also tested the staged approach on a case (test 15) with 200 runnables, 70 runnables in RB and 36 ECUs. The best result (fitness value of 0.13) was reached after 28.7 hours. The processing time required by the one-step algorithm prevented a realistic comparison in this case.

5.2.2 *Runtime evaluation*

Not only the staged approach manages to get equal or better quality solutions than the one-step approach, but computes them in a much shorter time. The graph in Figure 7 shows a comparison of the execution times required by the two algorithms for each experimental case. The runtime of the one-step and staged approaches increases not only with the problem size but also with the size of the GA initial population. Augmenting the size of the GA population is desirable, as in many cases this leads to a better value for the final solution. In our experiments, the final fitness value was mostly independent from the size of the initial population if it has more than 1,000 initial chromosomes. The runtimes in the figures are shown for an initial population of 10,000 chromosomes. As shown by the graphs, the two algorithms have an execution time that still grows exponentially with the size of the problem. However, the staged approach can solve problem configurations of a size comparable with the typical problems of the industry (approximately 6 hours for 100 runnables).

5.3 Robustness of the Staged Approach

This subsection evaluates the sensitivity of the staged approach with respect to the metric used in Equation (18) to select placement solutions. As an alternative to maximizing the minimum laxity metric in (18), we used a metric function (19) that minimizes the sum of the latencies of (a subset of) the transactions (which is another indication of an opportunity for assigning larger budgets). Nonetheless please note that ultimately what matters is the result of metric in (12).

$$f_{e2e}(\Psi, TBA) = |\xi| - \sum_{\Gamma_l} \frac{R_{\Gamma_l}}{D_{\Gamma_l}} \tag{19}$$

As shown in Figure 8, the original metric (18) provides better optimization results on the tests from 1 to 14, in the average by 10.8%. The reason for this is intuitive. Metric (19) may lead to situations, in which for some transactions

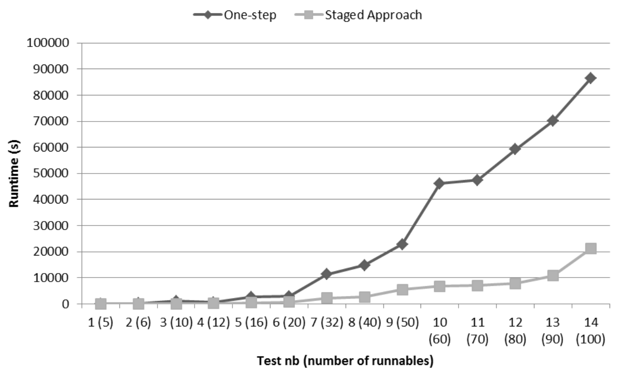


Figure 7: Runtimes of one-step and staged approach (GA initial population = 10000)

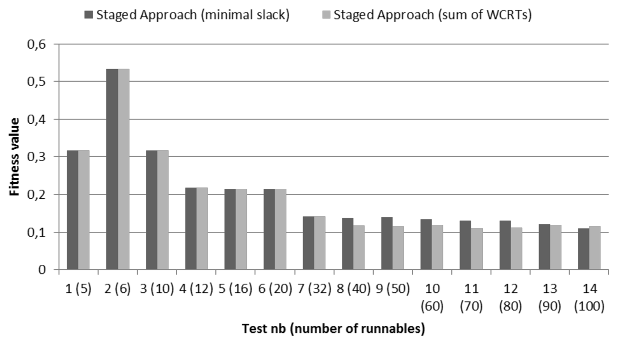


Figure 8: Comparison of two different metrics for staged approach

the response time is significantly reduced, whereas for others it is close to the deadline. This last set of transactions is a bottleneck for the relaxation of time budget values that follows next. This is not the case for the metric (18) which minimizes the response times with respect to the deadlines and maximizes the minimum value (does not operate on a sum of values).

Concerning the runtime, there is no significant difference between the two metrics. The slight differences in runtimes are mostly caused by the difference in the number of iterations of the main loop in the staged approach. However, the number of iterations (see Figure 9) is mostly similar and so are the runtimes. The only exception is test 11 where the use of metric (19) resulted in 138 iterations and a runtime of 7.6 hours, which is 29.75% higher than the case of a slack metric (18), but the final result is slightly better.

5.4 Influence of Initial Time Budgets

We tried additional test cases to confirm the choice of starting the iterative improvement algorithm with an initial setting of time budgets equal to the minimum allowed value for each runnable in RB. As previously stated, since the algorithm is a local search and terminates when no further improvements are possible, a selection of initial values that prevents schedulability would likely cause a premature termination. To verify, we tried initial budget assignments at $1\%, 2\%, 3\%, 5\%, 10\%$ and 20% of the range between the

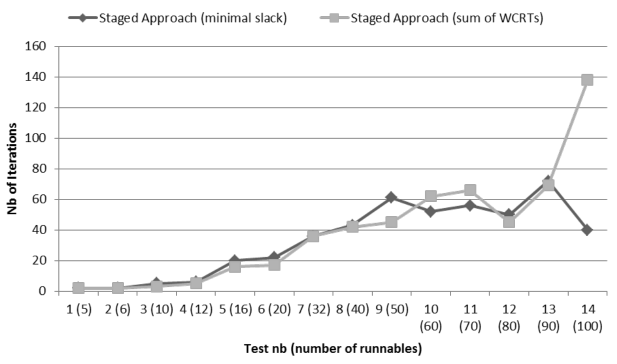


Figure 9: Comparison of nb of iterations of two different metrics for staged approach

minimum and maximum values $[tb^m_{r_j}, tb^M_{r_j}]$. In all our experiments, there was no sensible difference in the quality of the final result. However, for the highest values of the initial budgets (a 20% increase over the minimum value), the algorithm ended prematurely for all cases from 7 to 10. In these cases, the first deployment step from the iterative algorithm was not able to find any feasible solution.

5.5 Comparison with AllTimes approach [4]

In this section we compare our approach with the work on time budgeting coming from the AllTimes project [4].

5.5.1 Budgeting Algorithms

We studied the performance of our time budgeting algorithm (algorithm 1) with respect to the algorithm used in [4]. Authors of [4] claim that during each manual reconfiguration of deployment, they use the sensitivity analysis to find the relaxation of time budget values. The sensitivity analysis they refer to comes from [15]. This analysis is applicable if there is only one runnable in RB. The extended version of this algorithm that can budget multiple runnables is defined in [21]. Ultimately this is the algorithm that we implemented. We ported it in the one-step and staged approach for deployment instead of the algorithm 1 to see if it can be applied in the automated process of budgets and deployment specification.

Table 3 presents the runtimes (for the properties of the tests please refer to the Table 2), which clearly shows that usage of algorithm 1 leads to shorter runtimes in the context of both one-step and staged approaches. This difference is more significant for the one-step approach because the calls to the budgeting algorithm occur much more often. In fact, starting from test nb 3, usage of algorithm from [21] in the context of the one-step approach was too time consuming, i.e. after 24 hours the algorithm did not finish executing. The reason is that sensitivity analysis from [21] was designed to construct the map of all budget combinations for which the system remains schedulable (the schedulability region). Our algorithm is adapted to the metric of interest which allows to select one single configuration of time budgets, which equally distributes budget constraints among the suppliers based on the predefined values of $tb^m_{r_i}$ and $tb^M_{r_i}$. For all the tests for which algorithms terminated, we obtained the same fitness value, which supports the usage of the algorithm 1 in the automated process of time budgeting and deployment.

Table 3: Runtimes (seconds) of one-step and staged approach (GA initial population = 10000) when using different budgeting algorithms

Test nb	**One-step**	**Staged**	**One-step with [21]**	**Staged with [21]**
1	68.474	3.874	2088.665	25.416
2	177.383	4.667	2006.573	31.96
3	1014.547	27.442	> 86400	699.729
4	598.003	131.609	> 86400	1072.178
5	2565.429	429.153	> 86400	84132.114
6	2862.052	685.726	> 86400	> 86400

5.5.2 Improvements due to deployment

Interleaving of deployment with time budgeting has positive impact on the relaxation of budget values. The approach from [4] assumes that the deployment is known a priori. On the other hand, our techniques (either one-step or staged) interleave the deployment with budgets specification. This additional design freedom allows to further improve on time budget values. To show the gain, we first fix the deployment for tests 1 to 6 using our deployment technique (the runnables from RB were assigned WCETs equal to $tb^m_{r_i}$). Then we run the budgeting algorithm from [4] and obtained the following values for our metric in Equation (12): 0.31666666, 0.53333336, 0.178125, 0.088630214, 0.0712207, 0.07819336. This shows that by further manipulation of deployment interleaved with budgets assignment, we manage to get better results: 0%, 0%, 77.78%, 146.15%, 200.7%, 174.52% better than fixing the deployment for tests 1 to 6 respectively.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we present two algorithms that apply to an important problem in modern automotive design flows based on the AUTOSAR modeling paradigm for integrated architectures. The objective is to consider end-to-end deadlines in time-critical functionality distributed over several components and to define the assignment of time budgets to component functions developed by suppliers, while at the same time optimizing the ECU and task placement and priority assignment to the tasks.

Because of the intrinsic complexity of the problem, the proposed algorithms apply randomized optimization (GA) techniques. Further, to reduce complexity, we compare a one-step intuitive solution to the algorithm with an iterative approach. The iterative (two-stage) approach not only computes results faster, but also (and somewhat surprisingly) with equal or better quality. The runtime of the algorithm is already compatible with problems of industrial size, but we plan to further extend it by a careful evaluation of methods to adaptively choose the size of the initial population and possibly to reduce the execution time of the inner loop in the initial stages of the algorithm.

7. REFERENCES

[1] Autosar 4.1 specifications. `http://www.autosar.org/`.

[2] Autosar specification of timing extensions. `http://www.autosar.org/download/R4.0/AUTOSAR_TPS_TimingExtensions.pdf`.

[3] Autosar 4.1 methodology, version 3.1.0. `http://www.autosar.org/download/R4.1/AUTOSAR_TR_Methodology.pdf`, October 2013.

[4] ALL TIMES Partners. *ALL TIMES: D2.23.2 Final prototype of integrated system-level verification methodology*, August 2010.

[5] S. Anssi, S. Tucci-Piergiovanni, S. Kuntz, S. Gerard, and F. Terrier. Enabling scheduling analysis for autosar systems. *IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 152–159, 2011.

[6] E. Azketa, J. Uribe, J. Gutierrez, M. Marcos, and L. Almeida. Permutational genetic algorithm for the optimized assignment of priorities to tasks and messages in distributed real-time systems. In *Proceedings of the 10th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pages 958–965, 2011.

[7] L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, 1991.

[8] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien. Controller area network (can) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*, 35(3):239–272, 2007.

[9] M. Di Natale and A. L. Sangiovanni-Vincentelli. Moving from federated to integrated architectures in automotive: The role of standards, methods and tools. *Proceedings of the IEEE*, 98(4):603–620, 2010.

[10] M. Di Natale and J. A. Stankovic. Dynamic end-to-end guarantees in distributed real time systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 216–227, 1994.

[11] M. G. Dixit, P. Dasgupta, and S. Ramesh. Taming the component timing: A cbd methodology for real-time embedded systems. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 1649–1652, 2010.

[12] M. G. Dixit, S. Ramesh, and P. Dasgupta. Time-budgeting: a component based development methodology for real-time embedded systems. *Formal Aspects of Computing*, pages 1–31, March 2013.

[13] N. Feiertag, K. Richter, and C. Ficek. On the decomposition of end-to-end timing requirements in distributed partitioned automotive functions. In *SAE World Congress, Detroit, USA*, 2012.

[14] R. Gerber, S. Hong, and M. Saksena. Guaranteeing real-time requirements with resource-based calibration of periodic processe. In *IEEE Transactions on Software Engineering*, volume 21, pages 579–592, July 1995.

[15] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst. System level performance analysis - the symta/s approach. *Computers and Digital Techniques, IEE Proceedings -*, 152(2):148–166, Mar 2005.

[16] S. Hong, T. Chantem, and X. S. Hu. Meeting end-to-end deadlines through distributed local deadline assignment. In *Proceedings of the IEEE Real-Time Systems Symposium*, 2010.

[17] P. Jayachandran and T. Abdelzaher. Delay composition in preemptive and non-preemptive real-time pipelines. In *Real-Time Systems Journal*, volume 40, pages 290–320, 2008.

[18] A. Mehiaoui, E. Wozniak, S. T. Piergiovanni, C. Mraidha, M. D. Natale, H. Zeng, J.-P. Babau, L. Lemarchand, and S. Gérard. A two-step optimization technique for functions placement, partitioning, and priority assignment in distributed systems. In *Proceedings of the ACM SIGPLAN/SIGBED Conference on Languages, Compilers, Tools and Theory for Embedded Systems (LCTES)*, pages 121–132, 2013.

[19] B. L. Miller and D. E. Goldberg. Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, 9:193–212, 1995.

[20] C. Mraidha, S. Tucci-Piergiovanni, and S. Gerard. Optimum: a marte-based methodology for schedulability analysis at early design stages. *ACM SIGSOFT Software Engineering Notes*, 36(1):1–8, 2011.

[21] R. Racu, A. Hamann, and R. Ernst. A formal approach to multi-dimensional sensitivity analysis of embedded real-time systems. In *18th Euromicro Conference on Real-Time Systems*, 2006.

[22] J. Rox, K. Schmidt, A. Winter, T. Spengler, and R. Ernst. Estimating and mitigating design risk in a flexible distributed design process. *IEEE Embedded Systems Letters*, 2(2):35–38, 2010.

[23] O. Scheickl. *Timing Constraints in Distributed Development of Automotive Real-time Systems*. PhD thesis, Technische Universität München für Informatik, 2011.

[24] N. Serreli and E. Bini. Deadline assignment for component-based analysis of real-time transactions. In *2nd Workshop on Compositional Real-Time Systems, Washington, DC, USA*, 2009.

[25] TIMMO-2-USE Partners. *TIMMO-2-USE Timing Model - Tools, algorithms, languages, methodology, USE cases*, October 2012.

[26] K. Tindell and J. Clark. Holistic schedulability for distributed hard real-time systems. *Microprocessing & Microprogramming*, 40:117–134, 1994.