An Optimization Approach for the Synthesis of AUTOSAR Architectures

Ernest Wozniak, Asma Mehiaoui, Chokri Mraidha, Sara Tucci-Piergiovanni, Sébastien Gerard

An Optimization Approach for the Synthesis of AUTOSAR Architectures

Ernest Wozniak, Asma Mehiaoui, Chokri Mraidha, Sara Tucci-Piergiovanni and Sébastien Gerard
CEA, LIST, Laboratory of Model-Driven Engineering Applied to Embedded Systems
91191 Gif-sur-Yvette CEDEX, FRANCE
{name.lastname}@cea.fr

Abstract

Synthesis of automotive architectures is a complex problem that needs an automated support. AUTOSAR, standard for the specification of automotive architectures, defines a synthesis process of software components and their connections in a set of fixed-priority OS tasks distributed over a network of ECUs. During the synthesis process software components are allocated on ECU-s. Since each component encapsulates a set of so-called runnable entities, synthesis completes by partitioning runnable entities in OS tasks with assigned fixed priorities. This paper proposes an optimization approach for the synthesis of AUTOSAR architectures based on genetic algorithms and mixed integer linear programming techniques. Optimization criteria consider end-to-end timing responses and memory consumption.

1. Introduction

The AUTOSAR (AUTomotive Open System Architecture) [1] development process aligns to the MDE (Model Driven Engineering) principles as it is based on the use of models defined through meta-models and provides a development process based on a progressive refinement of models. At the heart of the development process lays the synthesis of the functional architecture (encapsulating the system logic) in a real-time architecture made of fixed-priority tasks distributed over a network of ECU-s. The synthesis process according to the AUTOSAR can be divided in four main activities. The first one is called the *allocation* as it consists in placing the atomic software components on the ECU-s and the exchanged signals on the buses. Each software component encapsulates the implementation of a specific functionality which is exposed to the outside world by means of ports. Internal behaviour of each atomic software component is represented by a graph of runnable entities, which in turn represent schedulable units of computation. Let us remark that each atomic software component contains at least one runnable entity. Runnables of the same component cannot be split among different ECU-s, hence we can say that allocation of atomic software components determines the allocation of runnables. From now on, when discussing allocation we refer to the allocation of runnable entities and data signals.

Second, third and fourth activity are *partitioning*, *scheduling* and *ordering*. Runnables and signals are being partitioned in OS tasks and messages which are then scheduled by the assignment of static priorities. Moreover the order of runnables inside a task is defined with respect to the functional constraints.

It is common and recommended practice [2] in this demarche, to specify the end-to-end timing constraints at the highest level, between input and output ports of the highest level component of the architecture, usually representing the system under study (see Figure 1). For each external stimulus, consumed by an input port, the constraint specifies a deadline for the response to be produced on the output port. Each end-to-end constraint is progressively refined by specifying the end-to-end chain of runnable entities (traversing one or more atomic components) that are activated to produce a system response triggered by the given stimulus.

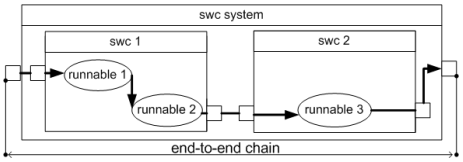

Figure 1 Visualization of end-to-end chain

By knowing runnables allocation their partitioning and order within the tasks as well as priorities of the tasks, the designer can now compute the response times of runnables end-to-end chains to see if end-to-end timing constraints are met. If some deadlines are violated, the designer has to find another configuration. This is a cumbersome process as the synthesis is an NP-hard problem. Hence appropriate support is required. In the current state of practice, only partial solutions exist as none of them handle all four dimensions (allocation, partitioning scheduling and ordering) at once. Most common solutions take as input a task model which means that the partitioning is already known or is done manually based on an engineer's expertise. This severely minimizes the design space to explore and therefore might exclude feasible solutions as shown later in this work.

This paper presents two techniques for the synthesis of AUTOSAR architectures in its entire form. The first

978-1-4799-0864-6/13/$31.00 ©2013 IEEE

technique is based on mixed integer linear programming (MILP). It returns the optimal solution but is limited to small size systems. In order to improve scalability – to address industrial size systems - a second technique, based on the genetic algorithms (GA) is proposed. For both techniques optimization criteria relate to the end-to-end responses and consumed memory. As runnables on the same ECUs communicate asynchronously through shared variables, a protection mechanism is required to guarantee data consistency and behaviour predictability. Protection mechanisms either take time or consume memory. Our techniques will operate a choice between a time-consuming and a memory-consuming protection mechanism, when optimizing response time and memory consumption.

This paper is organized as follows. The next section presents related work. Section 3 formalizes the system model, presents used schedulability analysis and memory consumption analysis, i.e. the way in which end-to-end responses and memory overhead are computed. Finally this section formulates the synthesis problem and optimization criteria. Section 4 presents the optimization technique, i.e. the Genetic Algorithms and the MILP formulation. Section 5 evaluates our techniques. First, results of the GA technique are compared against the MILP results. Then the GA technique is evaluated against current approaches disregarding the partitioning dimension. Section 6 concludes the paper and discusses the future work.

2. Related Work

The literature on the synthesis problem is rich. There are many approaches that consider only the allocation and task scheduling problems and relate to different optimization metrics (like the inter-ECU communication bandwidth) [3-6].

Concerning the system model and the optimization objectives our approach is closely related to the following works [7-12]. The [7] and [8] similarly as in our approach consider periodic activation and end-to-end responses as optimization criteria. The main difference is that the authors are considering OS tasks as an allocation unit and hence partitioning and ordering is fixed for them. Ferrari et al. [9] is the first work discussing possible strategies to protect shared data items and memory/timing tradeoffs. The work in [10] proposes a two-step technique for the allocation of AUTOSAR software components to the ECUs, taking into account protection mechanism as a parameter to specify. However it considers neither partitioning nor ordering. Authors of [11] and [12] also relate to the periodic runnables in their model. They consider additional mechanisms that can assure data consistency like the absence of preemption. The last can be done by defining so called preemption thresholds or preemption groups. In their work the allocation is fixed and hence their approach is for local optimization. Interestingly, the order of runnables as the parameter to manipulate is considered.

The main contribution of our work in relation to the current approaches ([13] gives an exhaustive survey on the existing methods for the optimization of real-time embedded systems) lies in a holistic deployment of the automotive architectures. None of the existing works is solving the four problems, i.e. allocation, partitioning, scheduling and ordering together, considering also assignment of a memory protection mechanism.

3. Formalism

3.1. System Model

The input system model consists of two graphs. The first one is an AUTOSAR execution model represented by a directed graph $G_e = \{V_e, E_e\}$ in which V_e is the set of vertices representing runnables and E_e is the set of edges related to the links between them. The second one is an undirected graph $G_h = \{V_h, E_h\}$ that expresses hardware architecture. Nodes represent hardware resources and the edges represent communication links between them. The hardware resources are ECUs and communication buses. The remaining notions used throughout this work were gathered in the table below for a better readability.

Concept	Explanation
E	Set of ECUs
e_i	ECU
β	Set of BUSes
b_i	BUS
$E(b_i)$	This function returns set of the ECUs communicating through the bus b_i
R	Set of runnable entities. In this work we restrict to periodic runnables, i.e. runnables that are activated in response to periodic timer events.
r_i	Runnable entity
P_{r_i}	Period of a runnable entity
d_{r_i}	Local deadline of the runnable entity r_i
$\vec{C}_{r_i} = (C_{r_i,e_1}, C_{r_i,e_2}, ..., C_{r_i,e_n})$	Worst case execution time of a runnable, characterized by a vector of WCETs, due to the heterogeneity of the hardware nodes. Later in the paper we omit the second index specifying the ECU, just for the simplicity of the notation.
swc_i	Atomic software component. Its behaviour is defined by the runnable entities.
$SC(r_i)$	This function returns atomic software component of a runnable r_i.
PS_{r_i}	Communication ports of runnable r_i
$P_{r_i}^{in}$	Set of input ports of the runnable entity r_i
$p_{r_i,j}^{in}$	j^{th} input port of the runnable r_i
$P_{r_i}^{out}$	Set of output ports of the runnable entity r_i
$p_{r_i,j}^{out}$	j^{th} output port of the runnable r_i

S	Set of data signals
s_i[1]	Data signal
$DS(s_i)$	Size of the data signal s_i
Ω	Set of shared resources
σ_i	Shared resource
$r^w(\sigma_i)$	Set of writer runnables, writing to the shared resource σ_i. We consider one-to-many communication hence $r^w(\sigma_i)$ contains only one writer.
$r^r(\sigma_i)$	Set of readers of the shared resource σ_i
$\zeta(\sigma_i)$	Data signal of the shared resource σ_i
$\zeta'(s_i)$	Shared resource corresponding to the signal s_i
Ω_{r_i}	Set of all the shared resources accessed by the input/output ports of the runnable r_i
$\omega_{p^{in}_{r_i,j}}$ $/\omega_{p^{out}_{r_i,k}}$	WCET of the runnable entity r_i on the critical section used for accessing shared resources. The access is through the input/output port $p^{in}_{r_i,j}/p^{out}_{r_i,k}$
$\xi = \{\Gamma_1, \Gamma_2, \ldots, \Gamma_n\}$	Set of the end-to-end chains that we call paths. Each path is defined as an ordered interleaving sequence of runnables and signals defined as $\Gamma_i = [r_{o_1}, s_{o_1}, r_{o_2}, s_{o_2}, \ldots, s_{o_{k-1}}, r_{o_k}]$. $src(\Gamma_i) = r_{o_1}$ is the path's source and $snk(\Gamma_i) = r_{o_k}$ is the sink. Multiple paths may exist between each pair of source-sink.
R_{Γ_i}	Response time of the path Γ_i
D_{Γ_i}	Deadline of the path Γ_i
τ_i	OS task – the code of a runnable entity executes in a context of an OS task
$e(\tau_i)$	ECU on which the task τ_i is allocated
$e(r_i)$	ECU on which the runnable r_i is allocated
$e(\sigma_i)$	ECU on which shared resource σ_i is specified
$\tau(r_i)$	Task in which r_i is partitioned
$idx(r_i)$	As multiple runnables can be partitioned in one task we specify the order by introducing the index of a runnable inside a task $j = idx(r_i)$. Hence $r_{i,j}$ means that runnable r_i is at the j^{th} position in a task.
π_{r_i}	Priority of a runnable entity. It equals to the priority of $\tau(r_i)$ which is defined as π_τ
C_{τ_i}	WCET of the task τ_i. It equals to the sum of WCETs of all the runnables partitioned within this task, i.e. $C_{\tau_i} = \sum_{\tau(r_i)=\tau_i} C_{r_i}$.
m_k	BUS message
$m(s_i)$	Message that transmits the signal s_i
$\vec{C}_{s_i} = (C_{s_i,b_1}, C_{s_i,b_2}, \ldots, C_{s_i,b_n})$	WCTT (Worst Case Transmission Time) of the signal s_i when transmitted on the BUS. Also here for the sake of simplicity, the second index indicating the bus is omitted when relating to the signal WCTT.
C_{m_i}	WCTT of the message m_i. It is a sum of WCTT of all the signals partitioned in this message, i.e. $C_{m_i} = \sum_{m(s_i)=m_i} C_{s_i}$.
P_{s_i}	Period of the signal s_i. It is equal to the period of a writer runnable.

Table 1. Main Concepts of the System Model

[1] Runnables communicate by sharing data signals accessed through their ports. Data signal can be communicated either through a shared resource or via a message passing. The identification of shared resources uniquely depends on the runnables allocation and partitioning. For each identified data signal communicated between runnables of different tasks but of the same ECU we define a shared resource. Data signals communicated between runnables of the same task don't require defining a shared resource. Also, for the inter-ECU communication no shared resource is required as in this case data signal is communicated by the BUS message.

3.2. Memory

The manner in which allocation and partitioning of runnables is done has an impact on the used memory. First, as it is specified in the AUTOSAR metamodel [14], runnable entity r_i has a stack memory usage that differs, depending on the ECU on which it is deployed. This is specified with the vector $\vec{M}_{r_i} = (M_{r_i,e_1}, M_{r_i,e_2}, \ldots, M_{r_i,e_n})$. The next factor affecting the used memory is the protection mechanism specified to protect the shared resources. Protection mechanism has to be specified for each signal communicated between runnables of different tasks that are deployed on the same ECU. This is due to the asynchronous communication between periodic runnables and hence, mechanism to provide the data consistency is necessary. This work considers two mechanisms:

- Wait-free access method such as Rate Transition (RT) block [15] – this mechanism behaves like a Zero-Order Hold block or a Unit Delay block plus a Hold block or Sample and Hold (for slow to fast transitions). Its implementation consists of a switched buffer. This mechanism incurs negligible time overhead but it consumes additional memory.
- Semaphore Lock (SL) – in this work we assume immediate priority ceiling semaphores. Priority of a runnable that is accessing a shared resource is raised to the ceiling priority of a resource. The SL, opposite to the RT, imposes no additional memory overhead, however it suffers timing delays in the form of a blocking time.

Function $\gamma(\sigma_i)$ will return the value representing the protection mechanism used to protect a shared resource σ_i. Value SL concerns semaphore lock, whereas RT means Rate Transition block. Overall memory overhead M_{e_i} for ECU e_i is computed according to (1).

$$M_{e_i} = \sum_{e(r_j)=e_i} M_{r_j} + \sum_{e(\sigma_k)=e_i} M_{\sigma_k} \quad (1)$$

The M_{σ_k} is a memory overhead caused by the RT and is computed according to [11] (see (2)). For this we define additional notation. We denote the set of readers with higher (lower) priority than the writer $r^w(\sigma_i)$ as $r^{HR}(\sigma_i)$ ($r^{LR}(\sigma_i)$). Our formula is a simplification of what is included in [11] as in this work we are not considering the preemption thresholds.

$$M_{\sigma_k} = DS\big(\zeta(\sigma_k)\big)n_k \quad (2)$$

$$n_{\sigma_k} = \begin{cases} \sum_{r_i \in r^w(\sigma_k), r_j \in r^{LR}(\sigma_k)} w_k + 2w_k \; if r^{HR}(\sigma_k) \neq \emptyset \\ \sum_{r_i \in r^w(\sigma_k), r_j \in r^{LR}(\sigma_k)} w_k + w_k \; if r^{HR}(\sigma_k) = \emptyset \end{cases} \quad (3)$$

$$w_{\sigma_k} = \begin{cases} 1 \; if \; \gamma(\sigma_i) = RT \\ 0 \; if \; \gamma(\sigma_i) = SL \end{cases} \quad (4)$$

3.3. Timing Analysis

Timing analysis concerns computation of the response times for runnables and global signals and also computation of the end-to-end responses. It is based on the work presented in [8] and adapted to consider runnable entities. Adaptation is because the entities considered in the analysis of [8] focus on the OS tasks and doesn't consider functional architecture as in our case.

3.3.1. Schedulability of Runnable Entities

Worst case response time of runnable r_i, for which $idx(r_i) = j$, is represented with $R_{r_{i,j}}$ and computed according to (5). The $C_{\tau_{i,j}}$ (see eq. 6) is the worst case computation time of the task until the j^{th} runnable partitioned in this task. Please note that we allow partitioning of the runnables with the harmonic periods in the same task. This means that when the task is executed not all of the runnables will be activated. Therefore the $C_{\tau_{i,j}}$ varies. However we assume the worst case scenario hence we account for all the runnables up till the j^{th} when computing the $C_{\tau_{i,j}}$. The B_{τ_i} is a blocking time of a task τ_i. Blocking time depends on the shared resources accessed by the task and the way in which the shared resources are protected from multiple accesses. If the shared resource is protected with a semaphore lock, it causes a blocking time. The semaphore lock in our case is realized through the Priority Ceiling Protocol (PCP) [16]. The same blocking time applies to all the runnables that are partitioned in the same task and therefore it is computed for a task. To compute the blocking time with the PCP few additional things have to be clarified. First, the shared resources of a task τ_i are specified with the set $\Omega_{\tau_i} = \bigcup_{\tau(r_j)=\tau} \Omega_{r_j}$. This means that the task inherits the access to the shared resources from the runnables partitioned in this task. The WCET of a task τ_i for accessing (reading/writing) a critical section of a shared resource σ_i is represented with $\tau_i(\omega_{\sigma_i}^{in}) = \max_{r_i}(\omega_{p_{r_{i,j}}^{in}}) / \tau_i(\omega_{\sigma_i}^{in}) = \max_{r_i}(\omega_{p_{r_{i,j}}^{out}})$. Function $hp(r_i)$ returns all the runnable entities allocated on the same ECU as r_i, with the priority higher than r_i.

$$R_{r_{i,j}} = B_{\tau(r_i)} + C_{\tau(r_i),j} + \sum_{r_k \in hp(r_i)} \left\lceil \frac{R_{r_{i,j}}}{P_{r_k}} \right\rceil C_{r_k} \quad (5)$$

$$C_{\tau_{i,j}} = \sum_{\tau(r_k)=\tau_i \wedge k \leq j} C_{r_k} \quad (6)$$

3.3.2. Schedulability of Signals

Worst case response time for a signal is computed in case when s_i represents inter-ECU communication. We are considering an event triggered bus, such as the CAN bus. Similarly to (5) the computation of R_{s_i} accounts for a blocking time B_{b_l}. This blocking time characterizes entire bus b_l. It is caused due to the non-preemptive scheduler of a bus, such as the CAN. This blocking time B_{b_l} applies to all the signals of a bus b_l, except those partitioned in a message with the lowest priority. Function $hp(m_i)$ returns all the messages of the same bus as m_i with a priority higher.

$$R_{s_i} = B_{b_l} + C_{m(s_i)} + \sum_{m_k \in hp(m(s_i))} \left\lceil \frac{R_{s_i}}{P_{s_i}} \right\rceil C_{m_k} \quad (7)$$

3.3.3. End-to-end Responses Computation

The worst case end-to-end latency R_{Γ_i} is computed for each path Γ_i by adding the worst case response times of all the runnables and global signals (i.e. signals representing intra-ECU communication), as well as the periods of all the global signals and their reader runnables on the path (see (8)). Set Φ represents all the global signals. The $read_{s_k,\Gamma_i}$ is the reader runnable of s_k on the specific path Γ_i.

$$R_{\Gamma_i} = \sum_{r_j \in \Gamma_i} R_{r_j} + \sum_{s_k \in \Gamma_i \wedge s_k \in \Phi} R_{s_k} + P_{s_k} + P_{read_{s_k,\Gamma_i}} \quad (8)$$

3.4. Problem Formulation

The main objective of this work is an automated support for the synthesis problem, i.e., the integration of functional architecture (represented by the communicating, periodic runnable entities and the exchanged data signals) with the execution platform, i.e. ECUs/BUSes and OS tasks/messages. This means that for each runnable/data signal, its hosting ECU/BUS has to be assigned (allocation), then runnables/data signals allocated to the same ECU/BUS have to be partitioned in tasks/messages, for which the priorities needs to be defined. Also the order of runnables inside the task needs to be established as it has a significant impact on the runnables local responses and in consequence end-to-end responses as well. Additionally, to protect the shared resources the protection mechanism needs to be specified (either RT or SL). This process should also respect multiple constraints which are specified with our MILP formulation (see subsection 4.2). Finally and most importantly, the synthesis process is driven by predefined optimization criteria. We are defining four optimization metrics and for each, its importance can be specified by assigning a weight. Therefore our final fitness function $F(\Psi_i)$ where Ψ_i represents a final configuration, i.e. deployed architecture, is a weighted sum of four functions: $F(\Psi_i) = w_1 \mathrm{f}_{e2e}(\Psi_i) + w_2 \mathrm{f}_{b_m}(\Psi_i) + w_3 \mathrm{f}_{th}(\Psi_i) + w_4 \mathrm{f}_{r_{ld}}(\Psi_i)$ explained below.

1) End-to-end Responses Optimization

Optimization of the end-to-end responses (9) aims at minimizing the response times of paths, relatively to their deadlines. Their optimization improves the system performance.

$$f_{e2e}(\Psi_i) = 1 - \sum_{\Gamma_j} \frac{R_{\Gamma_j}}{D_{\Gamma_j}} \quad (9)$$

2) Memory Optimization

Optimization of memory aims at minimizing the additional memory overhead that can be caused by using the Rate Transition blocks and inappropriate balancing when placing runnables on the ECUs. The last is due to the heterogeneous nature of the ECUs.

$$f_m(\Psi_i) = |E| - \sum_{e_j} \frac{M_{e_j}}{M_{e_j}^{max}} \quad (10)$$

The $M_{e_i}^{max}$ represents the worst case possible memory overhead caused for e_i; $M_{e_i}^{max} = \sum_{r_j} M_{r_j,e_i} + \sum_{\sigma_k} M_{\sigma_k}$. Its computation assumes that each runnable is partitioned in one task, writer has always higher priority than all its readers and all shared resources are protected with RT.

3) Bus Throughput

This objective concerns the increase of the buses throughput. For this (11) is used. Optimization of throughput is a common approach to provide higher extensibility of a bus.

$$f_{b_{th}}(\Psi_i) = 1 - \frac{\sum_{s_j \in \Phi} S(s_j)}{\sum_{s_k \in S} S(s_k)} \quad (11)$$

4) Runnables Local Deadlines

The runnables local deadlines optimization aims at minimizing the response times of runnables, relatively to their local deadlines (if any). Local deadlines might result from the refinement of the end-to-end timing constraints.

$$f_{r_{ld}}(\Psi_i) = 1 - \sum_{r_j} \frac{R_{r_j}}{D_{r_j}} \quad (12)$$

4. Optimization Technique

This section presents the Genetic Algorithm and the MILP formulation used to solve the synthesis problem. The MILP assures provision of an optimal solution if the solver terminates with no error. This property makes it a good comparator for such heuristic approach as the GA. The shortcoming of the MILP is the difficulty in handling larger use-cases. This was the main rationale behind using the GA. The MILP formalizes also the constraints that have to be respected during the synthesis. The last are preserved by our implementation of the GA.

4.1. Genetic Algorithm

Genetic Algorithm is an optimization technique patterned after natural selection in biological evolution. Each possible solution i.e. Ψ_i is encoded using a string of bits that we call a chromosome. One or few bits encode a solution for a specific parameter, in our case runnable entity or data signal. Group of bits corresponding to one parameter is called a gene. Later in this paragraph we specify our encoding, generation of initial population and how it is evolved in the consecutive iterations until the stop criteria is met. We also describe the correction mechanism that keeps our population consistent in regard to the constraints.

4.1.1. Encoding

Each chromosome ch_i represents a specific deployment configuration. Gene g_i relates either to runnable entity or a data signal. For the first, gene $g_i = ch_j(r_k)$ stores the value $V(g_i)$ representing runnable's allocation and partitioning. For a data signal, value stored depends whether it is a global data signal or a data signal that is communicated locally. Value for a global data signal will hold information about the BUS and the message in which it is partitioned. If this is a local data signal s_i, value depends on whether s_i is communicated through the shared resource or no. For the first case, value represents one of the two mechanisms, either SL (value = 1) or RT (value = 2). For the second, value equals 0.

The gene value $V_{r_j}(g_i)$ for the runnable r_j, this is one number but stores information about the ECU number on which runnable r_j is allocated, the task number in which it is partitioned, and the position (order) inside the task. The $V_{r_j}(g_i)$ for runnable r_j for which selected ECU is e_k, task τ_l and position p is computed in a specific way, according to (13). The max_{ECU} is the maximal number of runnables that can be allocated on one ECU and max_{Task} is the maximal number or runnables that can be partitioned in one task. These values are automatically initialized before running the GA. The max_{ECU} is computed as a maximal number of runnables that can be hosted by one ECU without violation of utilization (for this WCETs and periods of runnables are used). The max_{Task} is computed based on the maximal number of runnables with harmonic periods.

The gene value for a data signal, if transmitted on the bus, is computed in a similar way (see (14)). Figure 2 presents an example of a chromosome for a specific deployment configuration.

$$V_{r_j}(g_i) = (k-1) * max_{ECU} * max_{Task} + (l-1) * max_{Task} + (p-1) \quad (13)$$

$$V_{s_j}(g_i) = (k-1) * max_{BUS} * max_{Msg} + (l-1) * max_{Msg} + (p-1) \quad (14)$$

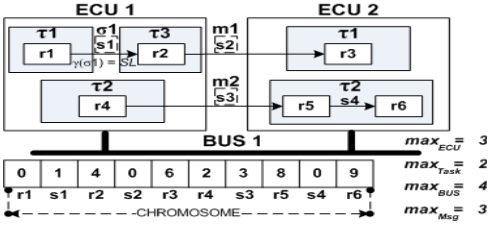

Figure 2 Example of a chromosome for specific configuration

4.1.2. Initial Population

The initial population is generated randomly but to generate correct chromosomes, possible range of values for each gene depends on the values already assigned to others. Correct means chromosome representing deployment configuration that respects the constraints formalized below with the MILP (see subsection 4.2).

4.1.3. Evolution

The evolution of a population is through the selection of chromosomes with good fitness and applying the crossover and mutation mechanism on them. The fitness is computed as presented in Section 4. For the selection we are using tournament selector [17] with a tournament size equal to 5. The crossover operator this is OX3 [18]. It creates two child chromosomes from the two parents. The OX3 choses two random positions in parent chromosomes. Then the values between them are copied from the first/second parent to the second/first child. The rest is copied from the first/second parent to the first/second child. The mutation operator choses a random gene in a chromosome and changes its value to the new random value. The last is selected from the values that don't violate the constraints. In addition it is possible to select the probability for applying the mutation operation. Mutation is done for the child chromosomes resulting from the crossover operation.

4.1.4. Correction Mechanism

When generating initial population or applying the mutation operator, the correctness of a new chromosome is preserved. This doesn't hold for the chromosomes resulting from the crossover operation. Therefore on each child chromosome the correction mechanism is called. This is to fix the genes which represent the values violating the constraints.

4.2. MILP Formulation

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.

As we have a large number of parameters, decision variables and constraints, they are directly discussed along with the specific aspects of the problem, in the following sections.

Runnables allocation: Runnables allocation is implicit through the allocation of components. Constraint (15) specifies that each software component is allocated on exactly one ECU. The $\varepsilon(swc_i)$ is the set of swc_i candidate ECUs. The a_{swc_i,e_j} is a binary variable that is 1 if the component swc_i is allocated on ECU e_j.

$$\sum_{e_j \in \varepsilon(swc_i)} a_{swc_i,e} = 1 \quad (15)$$

Constraint (16) defines the binary variables x_{swc_i,swc_j,e_k} based on a_{swc_i,e_k} and a_{swc_j,e_k}. x_{swc_i,swc_j,e_k} is set to 1 if swc_i and swc_j are allocated on the same ECU e_k, otherwise, x_{swc_i,swc_j,e_k} is 0.

$$0 \leq a_{swc_i,e_k} + a_{swc_j,e_k} - 2.\, x_{swc_i,swc_j,e_k} \leq 1 \quad (16)$$

Signals allocation: It is based on the allocation of runnables. Signal is either allocated on none or one bus. In constraint (17), the binary variable g_{s_i} indicates if the signal s_i is allocated on a bus ($g_{s_i} = 1$) or not ($g_{s_i} = 0$). The binary variable $a_{s_i b_k}$ indicates if the signal s_i is allocated on the bus b_k. Constraint (17) guarantees that if a signal represents inter-ECU communication, it is allocated on exactly one bus.

$$\sum_{b_k \in \beta} a_{s_i,b_k} = g_{s_i} \quad (17)$$

Constraint (18) assures that a signal s_i is allocated on a bus iff the components of its writer runnables and its reader runnables are allocated on different ECUs.

$$\forall\, r_i \in r^w\big(\zeta'(s_i)\big),\ r_j \in r^r\big(\zeta'(s_i)\big): 1 - \sum_{e_k \in E} x_{swc_{(ri)},swc_{(rj)},e_k} = g_{s_i} \quad (18)$$

Constraint (19) expresses the condition that the signal s_i is allocated on the bus b_k iff its readers and writers are on the ECUs communicating via b_k.

$$\forall_{r_i \in r^w\left(\zeta'(s_i)\right),\ r_j \in r^r\left(\zeta'(s_i)\right)}: 0 \leq \sum_{e_l \in E(b_k)} a_{swc_{ri},e_l} + \sum_{e_l \in E(b_k)} a_{swc_{rj},e_l} + g_{s_i} - 3\, a_{s_i,b_k} \leq 2 \quad (19)$$

Priority assignment: Constraints on priority assignment are specified in the same way for runnables and signals. Due to the lack of space, we give only the constraints for runnables. Constraint (20) defines the binary variable $\rho_{ri,rj}$ that expresses the priority order between runnables. $\rho_{ri,rj} = 1$ means that r_i has higher priority than r_j. If $\rho_{ri,rj} = 0$ and $\rho_{rj,ri} = 0$ then r_i and r_j have the same priority order.

$$\rho_{ri,rj} + \rho_{rj,ri} \leq 1 \quad (20)$$

Constraints (21), (22), (23), (24) and (25) ensure the observance of the symmetric, transitive and inversion properties of the priority order relation.

$$\rho_{ri,rj} + \rho_{rj,rk} - 1 \leq \rho_{ri,rk} \quad (21)$$

$$\rho_{ri,rj} - (\rho_{rj,rk} + \rho_{rk,rj}) \leq \rho_{ri,rk} \quad (22)$$

$$\rho_{rj,rk} - (\rho_{rj,ri} + \rho_{ri,rj}) \leq \rho_{ri,rk} \quad (23)$$

$$\rho_{ri,rj} + \rho_{rj,ri} + \rho_{rj,rk} + \rho_{rk,rj} \geq \rho_{ri,rk} \quad (24)$$

$$\rho_{ri,rj} + \rho_{rj,ri} + \rho_{rj,rk} + \rho_{rk,rj} \geq \rho_{rk,ri} \quad (25)$$

Runnables with non-harmonic periods are not allowed to have the same priority order. This is represented by the constraint (26). If ($P_{ri} \geq P_{rj}$) and ($P_{ri} mod\ P_{rj} \neq 0$):

$$1 = \rho_{rj,ri} + \rho_{ri,rj} \quad (26)$$

Runnables sequence order: A total order is defined for runnables to express the execution sequence order for runnables with the same priority. Constraint (27) defines the binary variable $So_{ri,rj}$ that represents the sequence order between r_i and r_j. The sequence order is total, i.e. either r_i is executed before r_j ($So_{ri,rj} = 1$) or r_j before r_i ($So_{rj,ri} = 1$).

$$So_{ri,rj} + So_{rj,ri} = 1 \quad (27)$$

Constraint (28) guarantees the antisymmetric and transitive properties of the sequence order relation.

$$So_{ri,rj} + So_{rj,rk} - 1 \leq So_{ri,rk} \quad (28)$$

Dependency constraints: Dependencies between runnables allow to set some sequence and priority orders i.e. when the execution of r_j depends on the execution of r_i, i.e. $(r_i \rightarrow r_j)$, it doesn't make sense to give higher priority or sequence to r_i (29).

$$if\ (r_i \rightarrow r_j): \rho_{rj,ri} = 0\ and\ So_{rj,ri} = 0 \quad (29)$$

To express next constraints we are defining the following set of binary variables: $SnHp_{ri,rj,e_k} = 1$ indicates that r_i and r_j are allocated on the same ECU e_k and r_j has higher priority than r_i (30).

$$0 \leq \rho_{rj,ri} + x_{swc_{(ri)},swc_{(rj)},e_k} - 2\,SnHp_{ri,rj,e_k} \leq 1 \quad (30)$$

$SnDp_{ri,rj,e_k} = 1$ determines that r_i and r_j are allocated on the same ECU e_k but they have different priority order (31).

$$SnDp_{ri,rj,e_k} = SnHp_{ri,rj,e_k} + SnHp_{rj,ri,e_k} \quad (31)$$

$SnSp_{ri,rj,e_k} = 1$ indicates that r_i and r_j have the same priority and they reside on the same ECU e_k (32).

$$SnSp_{ri,rj,e_k} + SnDp_{ri,rj,e_k} = x_{swc_{(ri)},swc_{(rj)},e_k} \quad (32)$$

Protection mechanism: Communication variables in inter-tasks communication within an ECU are either protected by RT blocks or semaphore locks. Constraints (33) and (34) determine the binary variable $Y_{\sigma i}$ that says if the shared resource σ_i should be protected ($Y_{\sigma i} = 1$) or not ($Y_{\sigma i} = 0$). When all writer and reader runnables of a shared resource are on the same ECU and task (i.e. they have the same priority), the protection of the shared resource is not needed (33). Within the same ECU, if there is at least one reader runnable with different priority as one of the writer runnables, the shared resource needs to be protected (34).

$$Y_{\sigma i} \leq \sum_{r_i \in r^w(\sigma_i)} \sum_{r_j \in r^r(\sigma_i)} \sum_{e_k \in E} SnDp_{ri,rj,e_k} \quad (33)$$

$$\forall r_i \in r^w(\sigma_i), r_j \in r^r(\sigma_i): Y_{\sigma_i} \geq \sum_{e_k \in E} SnDp_{ri,rj,e_k} \quad (34)$$

To specify the mechanism of protection, we define two binary variables $mem_{\sigma i}$ and $lock_{\sigma i}$. $mem_{\sigma i} = 1$ indicates that σ_i is protected using RT blocks and $lock_{\sigma i}$=1 that the protection mechanism for σ_i is the semaphore lock. Constraint (35) gives the relationship between variables $Y_{\sigma i}, mem_{\sigma i}$ and $lock_{\sigma i}$.

$$Y_{\sigma i} = mem_{\sigma i} + lock_{\sigma i} \quad (35)$$

Memory utilization: The additional memory cost in the MILP is defined using some variables. We define the binary variable $V_{\sigma i}$ based on the priorities of writer and readers runnables. The $V_{\sigma i} = 0$ means that for the shared resource σ_i there are no reader runnables with higher priority than the writer, the value of $V_{\sigma i}$ in this case is fixed by constraint (36). If there is at least one reader runnable with higher priority than the writer runnable then the value of $V_{\sigma i}$ is set to 1 in (37).

$$\forall r_i \in r^w(\sigma_i): V_{\sigma_i} \leq \sum_{r_j \in r^r(\sigma_i)} \sum_{e_k \in E} SnHp_{r_i,r_j,e_k} \quad (36)$$

$$\forall r_i \in r^w(\sigma_i), r_j \in r^r(\sigma_i): \sum_{e_k \in E} SnHp_{r_i,r_j,e_k} \leq V_{\sigma_i} \quad (37)$$

The memory needed for each shared resource is computed as in constraint (38), where, Z_{r_i,r_j,e_k} is a binary variable equal to $SnHp_{r_i,r_j,e_k} * mem_{\sigma_i}$. Z'_{σ_i} is another binary variable set to 1 if both $V_{\sigma i}$ and mem_{σ_i} are equal to 1, otherwise, it is equal to 0. Z_{r_i,r_j,e_k} and Z'_{σ_i} are defined in the same way as the variable $SnHp_{ri,rj,e_k}$ in constraint (30).

$$\forall r_i \in r^w(\sigma_i): memSize_{\sigma i} = \sum_{rj \in r^r(\sigma_i)} \sum_{e_k \in E} Z_{r_i,r_j,e_k} + mem_{\sigma_i} + Z'_{\sigma_i} \quad (38)$$

Semaphore lock: Memory consumption can be avoided by using semaphore locks. However, this will result in a blocking time for runnables (according to PCP) equal to the largest critical section of lower priority runnables sharing resources with higher or equal priority ceiling. Constraint (39) expresses the blocking time of r_i based on the binary variable $cond_{ri,rj,\sigma i}$, which represents the necessary condition to consider r_j during the computation of B_{ri}. The definition of this condition is given in constraint (40). It consists in the combination of three sub conditions: ***i)*** r_j is lower priority than r_i, ***ii)*** σ_i is a shared variable protected by a lock and ***iii)*** the priority ceiling of σ_i is higher or equal to the priority of r_i. The third condition is expressed by the binary variable $PC_{ri,\sigma i}$.

$$B_{ri} \geq cond_{ri,rj,\sigma i} * \omega_{p_{r_j,\sigma i}} \quad (39)$$

$$0 \leq PC_{ri,\sigma i} + lock_{\sigma i} + \sum_{e_k \in E} SnHp_{ri,rj,e_k} + -3.cond_{ri,rj,\sigma i} \leq 2 \quad (40)$$

As priority of runnables sharing directly a resource is always lower or equal to the priority ceiling of this resource, $\forall r_i \in \left(r^w(\sigma_i) \cup r^r(\sigma_i)\right): PC_{ri,\sigma i} = lock_{\sigma i}$.

For runnables r_i that do not share the resource σ_i, the value of $PC_{ri,\sigma i}$ depends on the priority of all runnables r_j sharing σ_i i.e. if there is at least one runnable r_j sharing σ_i with higher or equal priority than r_i, $PC_{r_i,\sigma_i} = 1$ (41) otherwise if all runnables r_j sharing σ_i have lower priority than r_i then $PC_{ri,\sigma i} = 0$ (42).

$$\forall_{r_i \notin (r^w(\sigma_i) \cup r^r(\sigma_i))} \; and \; \forall_{r_j \in (r^w(\sigma_i) \cup r^r(\sigma_i))} : PC_{r_i,\sigma_i} \geq \sum_{e_k \in E} SnSp_{ri,rj,e_k} + \sum_{e_k \in E} SnHp_{ri,rj,e_k} \quad (41)$$

$$\forall_{r_i \notin (r^w(\sigma_i) \cup r^r(\sigma_i))} PC_{r_i,\sigma_i} \leq \sum_{r_j \in (r^w(\sigma_i) \cup r^r(\sigma_i))} (1 - \sum_{e_k \in E} SnHp_{ri,rj,e_k}) \quad (42)$$

Worst-case response time computation: The computation of runnables WCRT needs the definition of some variables, $SnHo_{ri,rj,e_k}$, $V_{ri,rj}$and I_{ri,rj,e_k}. The binary variable $SnHo_{ri,rj,e_k}$ indicates whether the runnable r_j , that is allocated on the same ECU as r_i and has the same priority as r_i, has a higher execution order than r_i. The definition of the integer variable V_{r_i,r_j}, which represents the number of possible interferences of r_j on r_i $\left(V_{r_i,r_j} = \left\lceil \frac{R_{r_i}}{P_{r_j}} \right\rceil \right)$, is captured by (43).

$$0 \leq V_{r_i,r_j} - (R_{r_i}/P_{r_j}) \leq 1 \quad (43)$$

The integer variable I_{ri,rj,e_k} is the number of possible interferences of r_j on $r_i$when r_j is higher priority than r_i ($I_{ri,rj,e_k} . = V_{r_i,r_j} * \rho_{rj,ri}$). We use in the same way as previous the Big M method to linearize I_{ri,rj,e_k}. Finally, (44) gives the computation of WCRT.

$$\sum_{r_j} \sum_{e_k \in E} SnHo_{ri,rj,e_k} . C_{rj,e_k} + \sum_{r_j} \sum_{e_k \in E} I_{ri,rj,e_k} . C_{rj,e_k} + B_{r_i} + \sum_{e_k \in E} a_{swc(r_i),e_k} . C_{r_i,e_k} = R_{r_i} \quad (44)$$

The computation of signals WCRT is similar to runnables with the only difference that the blocking time B_{s_j} is computed as the largest WCTT of any group of equal priority signals sharing the same bus (45).

$$\forall_{s_j \neq s_i} : B_{s_i} \geq \sum_{b_k \in \beta} SnDp_{s_i,s_j,b_k} . C_{s_j,b_k} + \sum_{s_l} \sum_{b_k \in \beta} SnSp_{s_j,s_l,b_k} . C_{s_l,b_k} . SnDp_{s_i,s_l,b_k} \quad (45)$$

5. Experiments

There are two goals of the conducted experiments. The first one is to show the quality of results obtained with the GA (with respect to the optimal solutions given by the MILP) and the runtimes. Next, we compare our technique with the existing approaches that do not consider partitioning. The last has a high influence when optimizing the timing responses and the memory overhead.

5.1. GA vs MILP

In order to assess the quality of results obtained with the GA we compared it to the MILP. MILP assures optimality of the result in case if the solver finishes with no error. Unfortunately for the larger use-cases, solver that we used – CPLEX [19], although run on a powerful machine, is not able to provide an optimal result. It either finishes computation with an error message not providing any result or it stops with an out of memory message. In the second case, it returns a result, however it is not sure whether it is optimal. Therefore we have set use-cases for which we can infer optimal configurations (solutions). This has been done by first applying the weight 1 for the function responsible for optimizing end-to-end responses and fixing a simple use-case shown in the Figure 3[2]. For this case, with the only optimization of response time, the set of optimal configurations contains any possible partitioning, and for each shared resource (if any) the protection mechanism is the RT. The left configuration presented in Figure 4 is an example of an optimal solution for the simple use-case.

Figure 3 Non-replicated Use-Case

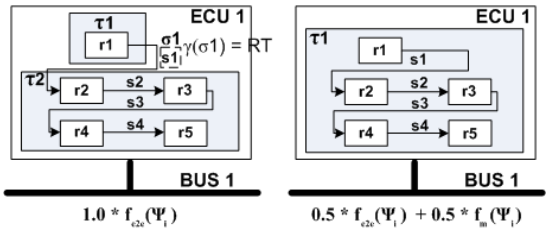

Figure 4 Optimal configurations

Once the simple use-case has been fixed, other use-cases were found by replicating the simple use-case. This means that, each path, runnable, ECU and BUS is replicated. Hence when replicating by 11 we obtained a use-case with 55 runnables, 11 ECUs and 11 BUS-es. Also, we connected each ECU to the original BUS and all the replicas. Let us remark that for the replicated use-cases, the set of optimal configurations is characterized by having each ECU containing only one path (no inter-ECU communication). Figure 5 shows the runtime for the MILP and the GA. Indexes on the horizontal axe express the factor for the replication. As can be seen, MILP on average gives the results in shorter time. However the Figure 6 shows that when architecture has been multiplied 6, 9, 10 and 11 times, the solver didn't return any solution. This was due to the returned error. For the factor 5, 7 and 8, the CPLEX finished execution with "out of memory exception". Nevertheless for the factor 5, returned result is optimal, which is not the case for 7 and 8. The GA for all the replication factors was able to return the optimal solution. We run similar tests but with weights 0.5 for the end-to-end responses and 0.5 for the memory optimization. Set of the optimal solutions for the non-replicated use case contains only one configuration in which all the runnables are partitioned in one task.

[2] For sake of simplicity, we omitted from the image the presentation of software components. In this case we assume one software component per runnable entity.

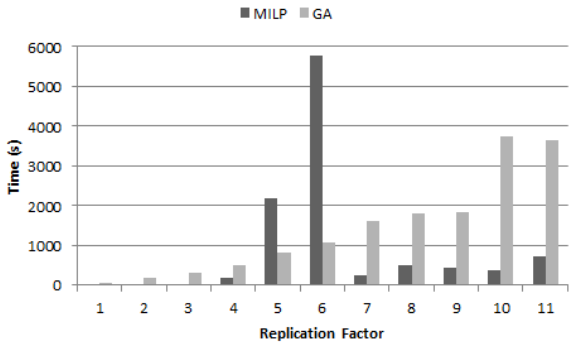

Figure 5: Runtime for MILP and GA ($1.0 * \mathbf{f}_{e2e}(\Psi_i)$)

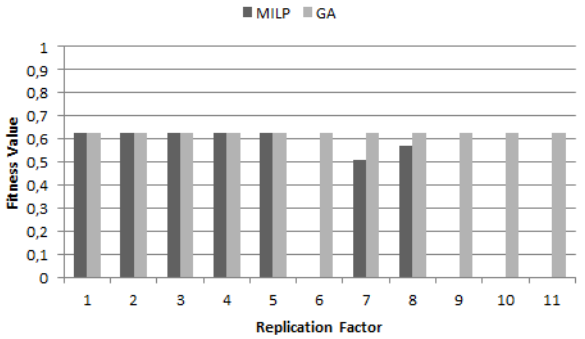

Figure 6: Results for MILP and GA ($1.0 * \mathbf{f}_{e2e}(\Psi_i)$)

For the replicated use-cases, each ECU must contain only one path. In this case, on average, MILP provides the results in a shorter time (see Figure 7). However, already for the replication factor 5 (see Figure 8) the returned result was not optimal and starting from 9, CPLEX didn't provide any result. The degradation of the results given by the GA, started from factor 8. In general, the reason for this is that when using equal weights for the latency and memory optimization functions, the set of optimal configurations is smaller than if optimizing only end-to-end latencies. This is due to the fact that optimal solutions only have the runnables of the same path partitioned in the same task. The configuration on the right side of the Figure 4 represents the only optimal configuration for the non-replicated use-case. In all of those use-cases the GA was run with an initial population of 10000. The algorithm stops if during the 20 consecutive evolutions, the fittest chromosome doesn't change. When we run the GA on the population of 100000 we were able to obtain the optimal solution for the second metric and the replication from 8 to 11. This however increased the runtime to around 12 hours on a 2.4 GHz single processor computer with 8GB of memory.

5.2. Evaluation against approaches with none or partial Partitioning

This set of tests shows the added value of considering the partitioning. Therefore we will compare the results obtained with the GA with those that doesn't consider

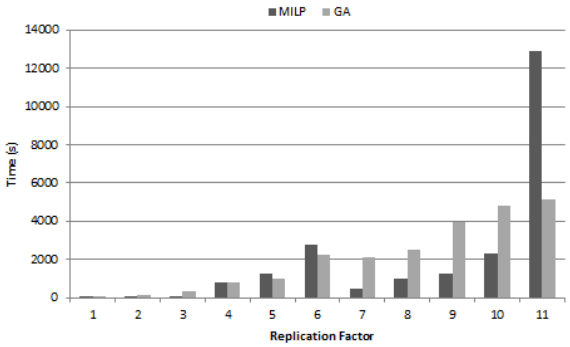

Figure 7: Runtime for MILP and GA ($0.5 * \mathbf{f}_{e2e}(\Psi_i) + 0.5 * \mathbf{f}_m(\Psi_i)$)

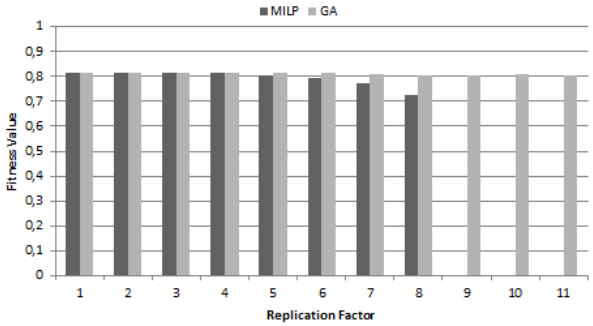

Figure 8: Results for MILP and GA ($0.5 * \mathbf{f}_{e2e}(\Psi_i) + 0.5 * \mathbf{f}_m(\Psi_i)$)

the partitioning (which is the case for [8]) or approaches that consider only partial partitioning, i.e. only runnables of the same period can be merged together (the case for [10]). The last two were implemented in MILP hence for them the obtained results are optimal in case when solver returned the result without error. The tests were run on a set of random input architectures.

Figure 9 and 10 show that consideration of the partitioning has an impact on the optimization metrics. For the fitness 1.0 for the $f_{e2e}(\Psi_i)$ (Figure 9) the GA obtained results 34,87% better than those with no partitioning and 16,48% from those with partial partitioning. Please note that for all of the approaches the same schedulability test is used (see subsection 3.3). Hence the metric improvement is a consequence of the constrained design space for the approach with partial and no partitioning. Possible, further improvement is to change the way in which $C_{\tau_{i,j}}$ from the equation 6 is computed for the approach with full partitioning. For the moment it has pessimistic assumption that all the runnables, even those with harmonic but not the same periods will always be activated.

Considering Figure 10, the results of the GA were 6.8% better than those obtained with the approach disregarding partitioning, and 5.7% better from those which limit partitioning to the same periods. Let us note that for 35 runnables, CPLEX didn't return any result.

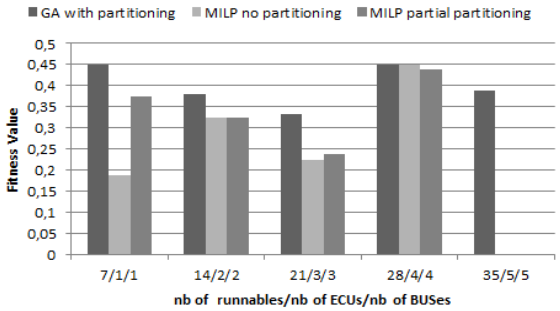

Figure 9: Comparison using fitness function $(1.0 * \mathbf{f}_{e2e}(\Psi_i))$

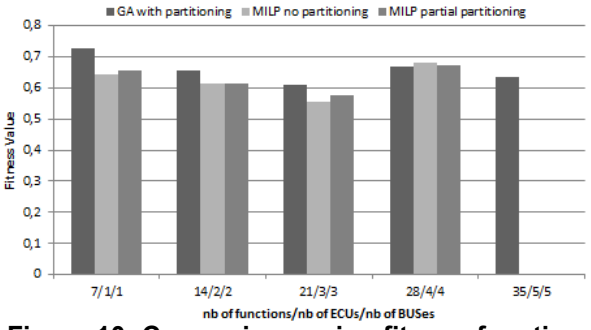

Figure 10: Comparison using fitness function $(0.5 * \mathbf{f}_{e2e}(\Psi_i) + 0.5 * \mathbf{f}_m(\Psi_i))$

6. Conclusions and Future Work

We presented a method for the optimized synthesis of the AUTOSAR compliant architectures. We proposed two techniques, MILP and GA. The main contribution of this work lies in a holistic approach that considers allocation, partitioning, scheduling and ordering together. As shown in the experimental part, employment of partitioning that is neglected in the current works, improves the optimization metrics. In addition we have evaluated our heuristic, i.e. the GA against MILP (i.e. exact approach) in terms of runtimes and the quality of results. As a future work we envisage to further scale our technique by provision of the parallelism in the implementation of the GA.

7. References

[1] *AUTOSAR Methodology*, AUTOSAR Std. http://-www.autosar.org/download/-AUTOSARMethodology.pdf

[2] *AUTOSAR Specification of Timing Extensions*, AUTOSAR Std. http://www.autosar.org/download/-R4.0/AUTOSAR_ TPS_TimingExtensions.pdf

[3] I. Bate and P. Emberson, "Incorporating scenarios and heuristics to improve flexibility in real-time embedded systems," in *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2006, pp. 221–230.

[4] W. Peng, H. Li, M. Yao, and Z. Sun, "Deployment optimization for autosar system configuration," in *Computer Engineering and Technology (ICCET)*, 2010.

[5] R. Long, H. Li, W. Peng, Y. Zhang, and M. Zhao, "An approach to optimize intra-ecu communication based on mapping of autosar runnable entities," in *ICESS*, 2009.

[6] R. Racu, M. Jersak, and R. Ernst, "Applying sensitivity analysis in real-time distributed systems," in *Proceedings of the 11th IEEE Real Time on Embedded Technology and Applications Symposium*, 2005, pp. 160–169.

[7] Q. Zhu, H. Zeng, W. Zheng, M. D. Natale, and A. Sangiovanni-Vincentelli, "Optimization of task allocation and priority assignment in hard real-time distributed systems," *ACM Transactions on Embedded Computing Systems*, vol. 11, no. 4, pp. 85:1–85:30, 2012.

[8] Q. Zhu, Y. Yang, E. Scholte, M. Di Natale, and A. Sangiovanni-Vincentelli, "Optimizing extensibility in hard real-time distributed systems," in *Proceedings of the 15th IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE, 2009, pp. 275–284.

[9] A. Ferrari, M. D. Natale, G. Gentile, G. Reggiani, and P. Gai, "Time and memory tradeoffs in the implementation of autosar components," in *DATE*, 2009, pp. 864–869.

[10] M. Zhang and Z. Gu, "Optimization issues in mapping autosar components to distributed multithreaded implementations," in *International Symposium on Rapid System Prototyping*, 2011, pp. 23–29.

[11] H. Zeng and M. D. Natale, "Efficient implementation of autosar components with minimal memory usage," in *SIES*, 2012, pp. 130–137.

[12] H. Zeng, M. D. Natale, and Q. Zhu, "Optimizing stack memory requirements for real-time embedded applications," in *ETFA*, 2012, pp. 1–8.

[13] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya, "Software architecture optimization methods: A systematic literature review," *IEEE Transactions on Software Engineering*, To appear.

[14] "Autosar system template," http://www.autosar.org/-download/R4.0/AUTOSAR_TPS_SystemTemplate.pdf

[15] *The Mathworks Simulink and StateFlow User's Manuals*. http://www.mathworks.com

[16] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization," *IEEE Trans. Computers*, vol. 39, no. 9, 1990.

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

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

[19] http://www-01.ibm.com/software/integration/-optimization/cplex-optimizer/.