Guided Rule-Based Multi-objective Optimization for Real-Time Distributed Systems

Konstantinos Triantafyllidis
Eindhoven University of Technology,
The Netherlands 5600 MB
Email: k.triantafyllidis@tue.nl

Egor Bondarev
Eindhoven University of Technology,
The Netherlands 5600 MB
Email: e.bondarev@tue.nl

Peter H. N. de With
Eindhoven University of Technology,
The Netherlands 5600 MB
Email: p.h.n.de.with@tue.nl

***Abstract*—Automated optimization of real-time architectures with respect to cost, performance, robustness and safety has received considerable attention in the last decade. In this paper, we present an automated Design Space Exploration (DSE) method based on both a multi-objective genetic algorithm and a heuristic particle-swarm-optimization technique. The optimization process is guided to desired solutions by weight coefficients that are assigned to the system objectives. The proposed method automatically generates architecture alternatives by changing hardware topology and mapping the tasks on different nodes, CPUs and by modifying their execution priority. Based on multiple quality objectives, the optimization method concludes to the Pareto-optimal solution set of the architecture alternatives. Moreover, in this paper we present an addition to the pre-existing optimization heuristics, targeting the reduction of the exploration time and maintaining a high-quality Pareto-optimal solution set. Finally, we compare the NSGA-II algorithm against the OMOPSO and their "*Rule-Based* Initial Population versions", for the convergence speed and the quality of their solutions, by comparing the hypervolume and the epsilon quality indicators. The proposed DSE approach has been applied to an autonomously navigating robot system consisting of several processing nodes (real-time distributed system) and resulting into better optimized and balanced solutions when compared to the proposed system architecture by an architect specialist.**

I. Introduction

The development of component-based real-time systems has become a common practice, enabling rapid system prototyping by the system composition from existing HW and SW components. The system should comply with specific real-time requirements such as throughput and latency. In order to find the optimal system architecture, an automated Design Space Exploration (DSE) method needs to be deployed. Due to the enormous design space (multiple factors of freedom), this DSE approach is required to rapidly converge to a set of optimal solutions. Moreover, the DSE should be supplied with accurate performance prediction metrics, due to the critical factors characterizing (hard) real-time systems. An incorrect performance prediction may lead to the implementation of an inefficient system architecture leading to system re-design or re-implementation.

In our previous work, we have presented an accurate and rapid performance analysis technique [1] [2] [3], based on cycle-accurate performance models. This performance analysis method is consisting of three individual phases and supports profiling and modeling of the SW components, system composition and performance analysis. The performance analysis method supports both schedulability and simulation-based analysis techniques for both processing and communication resources. In this paper, we focus on the integration of heuristic optimization algorithms, as the fourth phase of the proposed ProMARTES performance analysis method, for (a) automated generation of architecture alternatives, (b) evaluation of these alternatives on specific system quality objectives, (c) obtain of a Pareto-optimal solution set of system architectures. Finally, the architect selects the best solution for implementation from the Pareto-optimal solution set.

Multiple quality objectives are relevant for determining a real-time system, which increases the design space and the number of architecture alternatives that need to be generated and analyzed. Therefore, an advanced heuristic approach is needed to enable fast convergence to the Pareto-optimal solution set, while maintaining the quality of the solution set. The convergence speed can be improved by following some design rules that apply to real-time distributed systems, avoiding the exhaustive random generation of architecture alternatives. In order to increase the quality of the obtained solutions, the optimization can be guided by individual weighting of the quality objectives. For example, the robustness of a system may be more important than the cost, so that the Pareto-optimal solution set includes primarily robust solutions.

The optimal architecture of a Component Based-Real Time Distributed System (CB-RTDS) is based on the classification of the system solutions, by comparing the quality objective values. A CB-RTDS normally integrates multiple scenarios with end-to-end delays. These delays can be correlated to hard, firm, or soft deadlines. Moreover, the robustness of the system depends on the resource utilization and on the distance from the predicted end-to-end delays to the hard deadlines. Depending on the system requirements, corresponding system functions for quality objectives are defined. These quality-objective functions take into account the performance predictions (schedulability/simulation analysis) and combine them such that they reflect the performance of the system on the "end-to-end delays", the "robustness" and the "cost" metrics. An emerging research question is then the definition of the best quality-objective functions for the DSE in CB-RTDS domain.

In this paper we use two state-of-the-art heuristics, which we compare for their convergence speed to optimal solution

sets. In addition, we examine the quality of the solution set, by collating the obtained hypervolume-quality-indicator metrics.

The sequel of this paper is structured as follows. Section II presents related work. Section III describes the Design Space Exploration framework. Section IV poses the Architecture Optimization problem for CB-RTDS. Section V illustrates a case study used for the validation of the architecture optimization framework, while Section VI presents the obtained results from that case study. Finally, Section VII concludes the paper and gives the findings of the work.

II. Related Work

Most of the conventional modeling and analysis [4] [5] [6] techniques support performance analysis of RTDS, but do not enable automated DSE. The system architect has to define architecture alternatives manually. This makes identification of balanced and optimal architecture alternatives impossible, due to the abundance of different SW/HW mapping, HW topology, priority, and protocol variations. However, some performance-analysis approaches enable automated DSE by integrating different types of heuristics including annealing techniques, genetic algorithms, etc.

Pimentel *et al.* [7] [8] have presented the Sesame framework, which integrates an automated DSE method based on the SPEA genetic algorithm. The authors propose a method that prunes the design space by checking valid architecture alternatives prior to performance analysis of those alternatives. The analysis is performed by an Instruction Set Simulator bounds the design space with a decelerated procuedure. The initial random population, crossover and mutation operators do not guarantee convergence to the Pareto-optimal set, moreover the optimization is not guided to specific objectives.

ArcheOpterix [9] is designed for DSE of embedded systems that are modeled in AADL modeling language. Two quality objectives are supported focusing on reliability attributes only. The optimization heuristics are based on evolutionary algorithms, while the analysis method is formal.

Martens *et al.* have proposed PerOpteryx [10], a DSE framework for the well-known Palladio Component Models (PCM). The performance evaluation of the system is based either on simulation (SimuCom), or on a formal-based analysis (LQN solver) which speeds-up the analysis. The LQN solvers provide only mean values compared to the simulation-based analysis. The case study is based on a distributed system and three different objectives are formulated for the system architecture optimization (cost, performance and reliability). The system architecture optimization is based on genetic algorithm heuristics (opt4j optimization framework [11]), so that a global Pareto-optimal solution set cannot be guaranteed.

Zaccaria V. *et al.* have designed the Multicube explorer [12], which is an open source framework for DSE of chip multiprocessors. The Multicube performs DSE by supporting multi-objective particle-swarm optimization, genetic algorithms and simulated annealing optimizers, thus enabling selection of the appropriate heuristics depending on the problem to be solved. Performance-analysis simulation techniques are used, providing performance metrics in XML format to the Multicube explorer. The obtained metrics are examined for their Pareto optimality. Ykman-Couvreur C. *et al.* [13] use the Multicube explorer together with platform simulators for DSE of an MPEG-4 video encoder. Based on the DSE results, the authors conclude to a Pareto-optimal solution set of configuration alternatives in terms of power consumption and performance trade-offs.

The CARAT toolkit [14] supports synthesis of SW/HW component models, composition of system models by defining execution scenarios and simulation of these scenarios, while providing worst, average and best cases of CPU load, latency and throughput. CARAT supports EDR, RMA, and DMA scheduling algorithms, but it does not support accurate network modeling. Moreover, it does not provide cycle-accurate performance models, leading to less precise performance analysis. Based on this work, Li R. *et al.* have initiated the AQOSA framework which integrates the optimization module opt4j [11], supporting evolutionary algorithms, simulated annealing techniques and Particle-swarm optimization.

III. Design Space Exploration framework

The optimization presented in this paper finalizes the performance analysis pipeline for Component-Based Real-Time Distributed Systems (CB-RTDS) [14] [15] [3] [16]. The proposed DSE framework consists of four phases Fig. 1. The first three phases are integrated in the CB-RTDS performance-analysis framework, ProMARTES [1], while the last phase is the contribution of this paper. ProMARTES is the core of the DSE exploration framework, enabling rapid and accurate performance prediction of real-time distributed systems. The following paragraphs describe the four pipeline phases: (1) Profiling and Modeling, (2) Architecture Composition, (3) Performance Analysis, (4) Automated Architecture Generation and Optimization (see Fig. 1).

Profiling and Modeling. A component developer profiles the SW components at cycle-accurate instruction level, generating a performance model for each individual component by using our ProMo tool [3]. The ProMo tool provides all the profiling and modeling primitives for this phase. A component performance model addresses various HW resource-usage aspects (CPU, BUS, RAM, Network, etc.) and can be specified for multiple platforms. The generated performance models are loaded into the repository for Phase (3).

Architecture Composition. The system architect selects required components and, based on functional requirements, composes an architecture alternative with the ProMARTES graphical aids. The system composition can be repeated multiple times, generating a number of individual architecture alternatives. Each alternative specifies component instantiations and connections, HW nodes/network topology and mapping of SW components onto the HW platform. The system delivers services that combine a few operations, while these services may have real-time requirements and triggering events or periods. The service is defined by instantiating the required

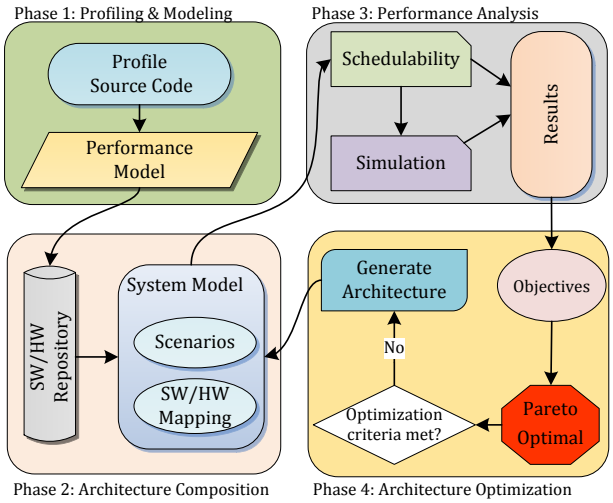

Fig. 1. Overview of the DSE approach in four phases.

operations from the SW components, defining the triggering periods/events and the possible real-time deadlines, consists of a system scenario. The functionality of a system is the definition of all execution scenarios such that the system delivers the required services. A system model is a combination of the execution scenarios and the architecture alternative. The system models are based on the MARTE modeling profile.

Performance Analysis. The generated system models are applied to two types of ProMARTES performance analysis: (a) schedulability and (b) simulation. Both analysis types support various HW platforms, multiple scheduling policies and different network protocols. Results yield accurate prediction of performance properties such as latency, hardware usage, robustness and throughput. The schedulability analysis enables early system performance prediction, while providing guaranteed worst-case execution time. The simulation-based analysis provides average-case execution times and detailed behavior time-line data. However, the simulation-based analysis requires substantial execution time in order to obtain converging results. Therefore, ProMARTES analysis firstly applies schedulability analysis to the generated system models, filtering out alternatives that do not satisfy the requirements, and secondly, simulates the survived alternatives providing accurate performance metrics and giving detailed data (WCET, ACET, bottlenecks) for each architecture alternative.

Architecture Optimization. The *Composition* and the *Evaluation* phases are connected through an optimization loop, enabling iterative architecture improvement. In this phase, new architecture alternatives are automatically generated using two different optimization techniques: the multi-objective Genetic Algorithm (GA) NSGA-II [17] and the multi-objective Particle Swarm Optimization (OMOPSO) [18]. Each newly generated alternative is applied to the performance analysis. The obtained performance metrics are translated to the quality objectives. Based on the objective values of the alternative, the solution is examined whether it dominates any of the existing Pareto-optimal solutions. When the optimization criteria of the optimization loop are satisfied, the architect obtains a final Pareto-optimal solution set (of alternatives). The automated architecture optimization phase is further detailed in the following section.

IV. Architecture Optimization

RTDS performance optimization typically results in optimally mapping of SW components onto the available HW nodes/platforms. The architecture optimization then explores the joint variation of key factors (best task-to-node mapping, its priority, which CPU platform). The triggered variation of the key factors leads to generation of different architecture alternatives. The quality objectives (cost, end-to-end delays, robustness, etc.), obtained from the performance analysis characterize each alternative. Moreover, the objectives may have different importance levels (can be weighted with higher or lower coefficient), thereby guiding the optimization to specific design space area. We have integrated the open-source multi-objective optimization framework jMetal [19] into the ProMARTES analysis framework, thereby enabling the applicability of many state-of-the art multi-objective optimization heuristics. The optimization module and its internal structure are depicted in Fig. 2.

In the following subsections, we describe the input model for the phase of alternatives generation and the optimization model with the correspondent objectives. Finally, we present the heuristc optimization algorithms that are deployed in the ProMARTES framework.

A. Input model for optimization loop

Prior to the first optimization loop, the original architecture alternatives are analyzed and form an optimization model structure. The optimization model structure define entities that can be and should not be changed during optimization. Permanent entities include tasks, components and scenarios that should be always executed to meet functional requirements. Variable entities define the above key factors of freedom which can be changed during optimization loops. The factors of freedom that often define a real-time distributed system are the HW platform (CPU, number of nodes), topology and the mapping of the SW components onto the HW nodes/CPUs. The HW platform is modeled by defining processing nodes, their topology, communication protocols, etc. For each task/operation of the SW components involved in the system, the execution node, the specific core of the CPU, the execution priority and the scheduling policy are defined. By varying the previous key factors, a number of individual architecture alternatives can be generated.

Identification of an optimal solution set for a real-time distributed system largely depends on the optimal task mapping. The Fig. 3 depicts the mapping of a $task_k$ on a specific node n_k, CPU cp_k, core c_k with specific execution priority p_k. $Task_k$ can be any task of a system that an architect has defined in the original architecture. Variable n_k and CPU cp_k, define the node and the CPU type which the task is

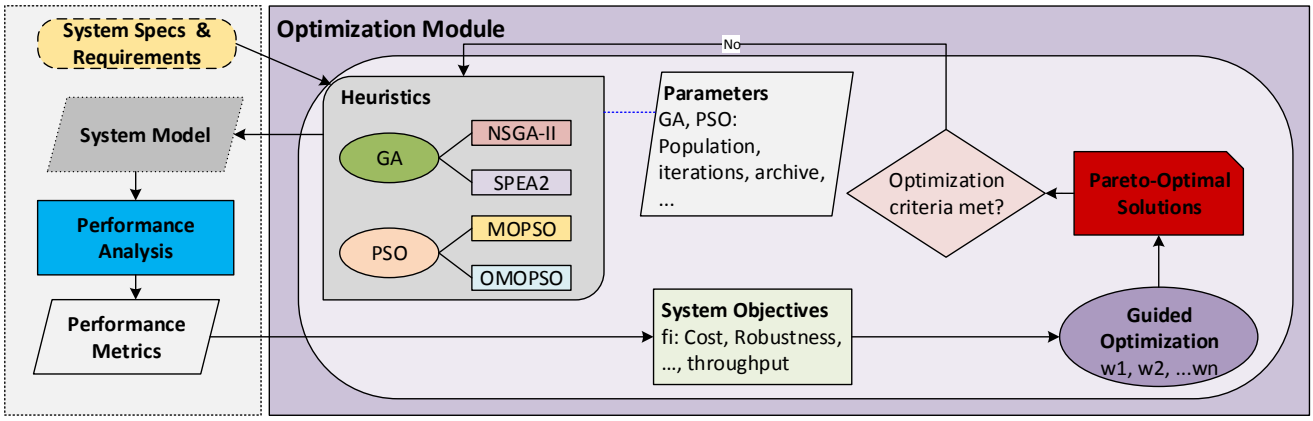

Fig. 2. Illustration of the architecture optimization process using system objectives and guided optimization.

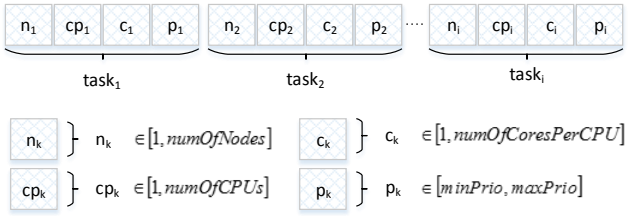

Fig. 3. Task mapping involving various system parameters.

mapped on respectively. The lowest value for the cp_k is 1 (we should have at least 1 CPU in the repository) and the highest is equal to the number of CPUs, $numCPUs$, included into the HW component repository. Value c_k defines the CPU core on which the task is going to be executed. This value is dependent on the selected CPU cp_k, since each CPU may have several number of cores. Finally, the execution priority p_k of the task $task_k$, has a value in the range $minPrio$ and $maxPrio$, where $minPrio$ is the minimum and $maxPrio$ is the maximum priorities that the system can handle. This is specified in Eq. (1) below

$$\begin{aligned} task_k = \{n_k, cp_k, c_k, p_k\}, n_k \in [1, numNodes], \\ cp_k \in [1, numCPUs], c_k \in [1, numCores], \\ p_k \in [minPrio, maxPrio], k \in [1, numTasks]. \end{aligned} \quad (1)$$

For each task, two boundary vectors are created, the $lLimit$ and the $uLimit$, as in Eq. (2) below

$$\begin{aligned} lLimit = [1, 1, 1, minPrio], \\ uLimit = [numNodes, numCPUs, numCores, maxPrio]. \end{aligned} \quad (2)$$

These two vectors are used to generate random task mapping in the value range of these two vectors. In order to create architecture alternatives in an automated way, heuristics are used to generate the mapping of each task $task_i$ (Eq. (1)). The heuristics generate random values with respect to the boundary vectors, thereby defining different architecture alternatives. The following sections describe the heuristics pipeline deployed in our analysis.

B. NSGA-II algorithm

The Nondominated Sorting Genetic Algorithm II (NSGA-II) [17] is a Multi-Objective Evolutionary Algorithm (MOEA) based on a non-dominated sorting approach. The NSGA-II starts with a randomly generated population. Using evolution operators (crossover and mutation) it explores the design space. Each individual generation is evaluated by fitness values and if it assessed as Pareto optimal then it is included in the solution set. Due to the random initial population and to the random generations, this optimization algorithm requires substantial time-span to converge to a set of optimal solutions.

C. OMOPSO

The Mutli-Objective Particle Swarm Optimization (MOPSO) [20], extends the heuristic Particle Swarm Optimization (PSO) [21]. The OMOPSO heuristic presented in [20] is based on Pareto dominance and uses a crowding factor to filter the list of available leaders. Moreover, it incorporates the ϵ-dominance factor concept correcting the final solution-set generated by the heuristic. The OMOPSO, similarly to the NSGA-II, uses a random generated initial population, while the solution set approaches gradually the Pareto-optimal curve.

D. Rule-based Initial Population for RTDS

The DSE of a real-time distributed system is a time-consuming procedure, due to the broad range of possible architecture alternatives (combination of factors of freedom) and the substantial analysis time that each individual alternative requires. Aiming at faster convergence to the optimal solution set, while not decreasing its quality, we have proposed several extensions to the current heuristics. These extensions are based on the properties of a common real-time distributed system.

In our previous work [1], for performance analysis, we have identified a set of guidelines for the mapping of the SW

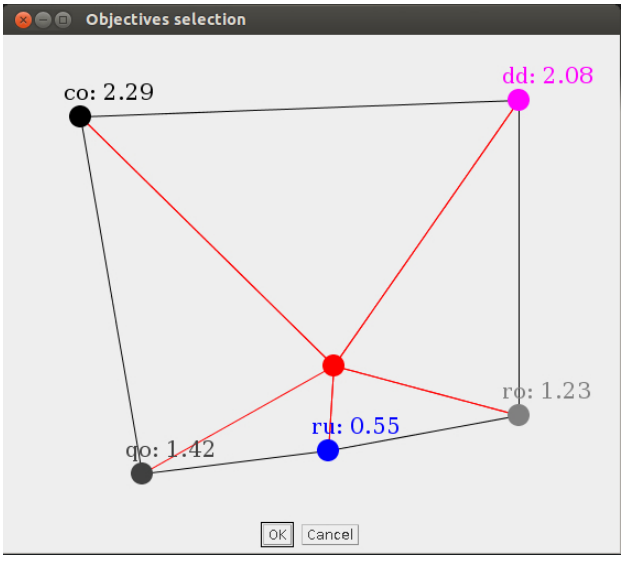

Fig. 4. Weighted objective definition.

components onto HW nodes. According to our findings, heavy tasks should be mapped on separate CPU cores. In practice, this reduces latencies and increases robustness of each task. Moreover, tasks that are data- and execution-dependent and especially the low-latency tasks, should be mapped on the same node/CPU to reduce the communication delays, which in most cases are disproportionally high, compared to the execution time delay on the CPU cores.

As we have mentioned above, the existing heuristics generate an initial population randomly and then, by tuning the factors of freedom, they generate new alternatives which are examined for their Pareto optimality. Based on the above mapping guidelines, we have proposed *rule-based* optimization (RBO). Instead of a random initial population, RBO generates a population that complies to the following rules: (a) the data-dependent real-time tasks are mapped on the same node, thereby reducing communication delays, (b) the computationally expensive tasks are mapped on different cores, if possible, (c) a task is assigned a specific priority, depending on the type (hard, firm, soft) of the end-to-end deadline of the scenario in which the task is involved. Nevertheless, the initial population is still generated randomly, but within the specific constraints which are set by the above-mentioned rules.

Formally defined, we create a d set which contains the data-dependent tasks, specified by

$$d = \{\{task_i, task_m\}, ..., \{task_n, task_l\}\}. \quad (3)$$

Following, we associate the tasks with the type of the deadlines by checking the scenarios they are involved in. By this, we define 3 individual sets sd, fd and hd, that contain tasks with soft-, firm- and hard-deadlines, respectively, by

$$\begin{aligned} sd &= \{task_i, task_m, ..., task_n\}, \\ fd &= \{task_j, task_k, ..., task_l\}, \\ hd &= \{task_o, task_p, ..., task_q\}. \end{aligned} \quad (4)$$

The performance models [3] of the involved individual components contain the execution cycles c_i that each operation op_i in task requires to be executed. In order to distinguish which tasks are more resource demanding, we have implemented the factor f_i. This factor is the rate of the cycles c_l divided by the amount of cycles c_m, c_l is the number of the cycles of the most resource-demanding operation l and c_m is the number of the cycles of the least resource-demanding operation m, out of all the operations that are involved in the system model. As a result, each operation op_i becomes associated with the metric f_i, representing the operation processing load, given by

$$\begin{aligned} f_i = c_l/c_m, c_l = max\{cycles_1, cycles_2, ..., cycles_n\}, \\ c_m = min\{cycles_1, cycles_2, ..., cycles_n\}, cycles_j \in op_j. \end{aligned} \quad (5)$$

E. Guided optimization

The quality attributes and objective functions of a system depend on the system application domain. Hard real-time system architects focus on real-time execution validation, while soft-real time system architects are more interested in graceful degradation capabilities. For some systems, platform cost is more important than its robustness, and vice versa. To support this, we have adopted a weighted approach for the quality objectives, where a system architect can define weight coefficients for each objective, penalizing solutions that do not meet the system constraints, thereby guiding the optimization process to solutions that comply with the system's application domain and requirements.

In order to facilitate the definition of weight coefficients, we have developed a GUI (see Fig. 4), where the architect can drag the vertices of the polygon and thus define the weight coefficient of each objective.

V. Case Study: 3 Navigating Robots

We have studied the optimal architecture for a sensing system, incorporating three autonomously navigating robots. The target quality objectives are latency in the control loops, robustness and hardware costs. This case study is a more advanced setup than the one presented in [1], where only one robot navigates through the space and composes a 2D environment map. Three robots generate three individual 2D maps which are combined to create a global reference map.

Component models, system composition, scenarios and performance analysis results have already been presented for the one-robot system in [16] and for three-robots case study in [2]. In this section we will focus on the optimization phase, where we use the factors of freedom in order to rapidly obtain a high-quality Pareto-optimal set of architecture alternatives.

A. Profiling and Modeling

The SW components have been profiled on specific HW platforms (`i5-2520, i5-750` etc.). During the profiling phase, the ProMo tool automatically generates the performance model of each individual SW component. These performance models contain cycle accurate performance metrics [3].

B. Architecture Composition

In this phase, we have manually defined 5 architecture alternatives defining HW topology, task mapping, priorities and network protocols. The execution behavior of the three autonomously navigating robot systems is described by $30+1$ execution scenarios. The 30 execution scenarios represent the navigation and the composition of the three 2D-maps, while 1 scenario provides merging of the three individual maps into one global map. Each scenario delivers a specific functionality and it is characterized by a trigger period and a deadline. The trigger periods and the deadlines are depicted in Table I; a more detailed description is available in [2].

Scenario Name	P (ms)	D (ms)	ML (bytes)
GM:Odom Scenario	37	100	165
GM:Laser Scenario	79	100	1209
MB:Odom Scenario	37	100	165
MB:Laser Scenario	79	100	1209
GM:Frame Scenario	50	100	1999
GM:Conversion Scenario	50	100	159
MB:NewGoal Scenario	500	500	171
MB:Frame Scenario	50	100	1999
GM:Map Scenario	1000	1000	16777216
MB:Nav Scenario	50	100	144
GM:Map Stitch	60000	60000	16777216

TABLE I
PERIODS (P), DEADLINES (D) AND MESSAGES LENGTH (ML) FOR THE RT SCENARIOS

C. Factors of Freedom

For this case study we have selected the following factors-of-freedom: (a) number of processing nodes, (b) type of the CPUs, (c) target core for each task, (d) execution priority of each task, (e) network protocol and transmission-rate. These factors-of-freedom are tuned by using the two different heuristics `NSGA-II` and `OMOSPO`.

D. Performance Analysis

Performance analysis for each architecture alternative is being performed by the schedulability analysis tool MAST [22]. The schedulability analysis yields guaranteed worst-case end-to-end delays in a short analysis time. Finally, when the optimization heuristic is terminated and a Pareto-optimal solution set is generated, a simulation-based analysis is applied to the found optimal architecture alternatives, thereby providing the values of the additional performance metrics: ACET, bottlenecks and timeline. The simulation-based analysis is performed by the JSimMAST analysis tool [23].

E. Quality Objectives

For the evaluation of the robot system architecture alternatives the following quality objectives are defined: (a) meeting all the real-time deadlines, (b) timeliness quality on the end-to-end delays, (c) robustness, (d) resource utilization and (e) platform cost.

The robot system features a number of scenarios with hard real-time end-to-end deadlines. Ideally, for each individual scenario an individual objective should be defined. However, this would substantially increase the complexity of the optimization process. Therefore, we have defined the hard-real time requirement objective dd as the sum of the scenarios that fail meeting the hard-real time deadlines (see Eq. (6)), giving

$$dd = \sum_{i=1}^{n} f(i), f(i) = \begin{cases} 1, & delay_i >= d_i \\ 0, & delay_i < d_i, \end{cases} \quad f(i) \in S^h, S^h = \{S_1^h, S_2^h, ..., S_i^h\}. \tag{6}$$

Here $f(i)$ is equal to 1, when the hard deadline of a scenario S_i^h is not met and 0 when the deadline is met, S_i^h is the union of the scenarios with hard deadlines. The aim is to design a system where the dd quality objective equals 0.

Even when all the real-time requirements are met (dd = 0), it is important to have another objective that denotes timeliness quality of an alternative. To this end, we introduce objective qo that computes how close the actual delays to their required deadlines are. With this objective we analyze the end-to-end delays of all the system scenarios, trying to find an alternative that for all the end-to-end delays has the maximum time distance from the scenarios deadlines. Quality objective qo is the sum of the ratios between the end-to-end delays and their corresponding deadlines, multiplied with specific weights (dependent on deadline type: soft, firm, hard), specified by

$$qo = \sum_{i=1}^{n} (delay_i/d_i)^{td} * W(td), \quad W(td) = \begin{cases} 1, & td \in S_i^s \\ 2, & td \in S_i^f \\ 3, & td \in S_i^h \end{cases}. \tag{7}$$

Here $delay_i$ is the end-to-end delay of the scenario i, d_i is the deadline of this scenario i, td represents the deadline type, S_i^s, S_i^f, S_i^h are the scenarios with soft, firm and hard deadlines respectively. The larger the sum, the better the timeliness quality of the architecture alternative. When the objective qo has a negative value the alternative is considered as an outlier.

Robustness of a system under overload conditions is also an important design objective. When a system meets all the real-time requirements, it can be considered as a promising candidate. However, in case a system fails to meet one or more hard real-time deadline(s) under overload conditions, the system might be considered as under-provisioned, depending on the requirements. Taking this into account, we introduce the ro robustness objective value. In contrast to the qo quality objective where we analyze all the end-to-end deadlines, in the ro quality objective we are interested only for the scenario delay that is closest to its hard deadline, specified by

$$ro = \max_{i=1}^{n} (delay_i/d_i)^h, \tag{8}$$

where h denotes the hard end-to-end scenarios. The smaller the value of ro, the better the overall robustness performance of a system under overload conditions.

Another crucial system objective is utilization of processing resources, where lower processing utilization values show the stability of a system. The resource utilization objective ru

Objective	max / min	weight
dd (hard deadline)	min	2.0
qo (timeliness objective)	max	1.0
ro (robustness objective)	min	1.0
ru (resource utilization)	min	1.0
co (cost objective)	min	2.0

TABLE II
MAXIMIZATION OR MINIMIZATION OF THE OBJECTIVE IN THE CB-RTDS PROBLEM.

is equal to the highest utilization factor of all the system processing resources, which is defined by

$$ru = \max_{i=1}^{n} coreUtil_i, \tag{9}$$

where $coreUtil_i$ is the core utilization factor of $core_i$.

The platform cost objective co plays a definitive role for the selection of the optimal system architecture in most of the cases. Low cost and, at the same time, high performance, are the two main characteristics of an optimal architecture. The platform cost is computed by

$$co = \sum_{i=1}^{n} cost_i, i \in \{HW\}, \tag{10}$$

where i is a processing node of the system and a member of the existing set of HW components.

The optimization heuristics identify the optimal solution set according to the above-mentioned objectives, by maximizing or minimizing the values of these objectives (see Table II).

F. Guided Optimization

In this case study we prioritize the objectives by giving different weight coefficients, thereby guiding the DSE to specific solution areas. More specifically, we are mostly interested in a system that: (a) meets all the hard real-time requirements, (b) features low platform cost. Simultaneously, we pay less attention to the robustness and the timeliness balance of the solutions. The weights used for the guided optimization are presented in Table II.

VI. EXPERIMENTS AND RESULTS

We have applied both the NSGA-II and the OMOPSO optimization algorithms [1] , as well as our *rule-based* extension to the DSE of the autonomously navigating robot case study and concludes on the optimal architecture solution set.

From the set provided by the OMOPSO-RB heuristic, we choose the best architecture alternative, M, and compare this with another 3-node alternative, Architecture C, which was composed by the system architect in [2]. The alternative M that we choose is composed of 3 processing nodes (see Fig. 5) with quality objectives: dd=0, qo=-12, ro=0.511, ru=59.5, co=757. Compared to the Architecture C, despite the fact that they are both based on 3 processing nodes, the alternative M has better performance in the cost objective (10 % lower cost) while it does not miss any deadline. Therefore, we consider alternative M as more suitable than alternative C.

Regarding the results of the optimization heuristics, the original NSGA-II optimization includes only 2-node implementations in the solution set. In the solution set of the *rule-based* NSGA-II heuristic, 2-node and 3-node architecture alternatives are present. The 3-node alternatives are optimal, regarding the qo, ru and ro quality objective functions. On the other hand, the 2-node architectures feature low cost for the co objective function, while meeting all the hard real-time deadlines. The OMOPSO heuristic includes only five architectures in the resulting Pareto-optimal solution set. The proposed solutions are based on 3 nodes providing better performance in the qo, ru and ro quality objectives, while increasing the cost only by 10%, due to the deployment of inexpensive CPUs. The OMOPSO Pareto-optimal solutions are a subset of the OMOPSO-RB. The latter heuristic results in 13 different Pareto-optimal architecture alternatives. These alternatives are based either on 2 or 3 nodes. The systems that are based on 2 nodes have better evaluation regarding the cost quality objective, while the systems based on 3 nodes, feature better overall performance, except for the increased cost quality-objective co.

Fig. 6 depicts the optimization of the design space exploration of the four heuristics. In the three-dimensional space of the diagrams, we visualize the cost *vs.* the robustness *vs.* timeliness quality objective of the solution. To represent the fourth dimension, we use color showing the number of hard deadlines that each alternative does not meet. Since our robot case study is a hard real-time system, the optimal solutions for our requirements are colored in dark blue (no missed deadlines). It is visible by this diagram that the *rule-based* extensions yield a better performance, proposing close to Pareto-optimal solution sets. Moreover, the optimal solutions provided by the OMOPSO-RB dominate the optimal solutions generated by the NSGA-II-RB, since from the figure it is visible that for the same dd=0, the solutions provided by the OMOPSO-RB have a lower cost, are more robust and their timeliness is more consistent.

Findings. By comparing the two different Pareto-optimal solution sets (OMOPSO-RB *vs.* NSGA-II-RB), we conclude that the OMOPSO heuristic tends to concentrate on solutions that are close/local in the design space area (flock of birds), while the NSGA-II searches in broader space, providing more but not fundamentally better solutions. On the other hand, for the exploration of 1,000 alternatives, the OMOPSO-RB is running for 34 minutes compared to the 27 minutes that the NSGA-II-RB requires, while the hypervolume indicators of the solutions are 0.645 and 0.626, respectively. It is evident from Fig. 6 that the *rule-based* extension for the NSGA-II increases its performance significantly, since the NSGA-II heuristic includes a very limited number of solutions that meet the hard deadlines (dark blue color).

[1]The population parameter for the NSGA-II algorithm was set to 50, the evaluations to 5,000, the crossover was set to 0.9, the crossover distribution index to 20.0, the mutation probability is set to 1.0/N, (the number of tasks N=33) and the mutation distribution index is set to 20.0. Regarding the OMOPSO heuristic, we have set the swarm and the archive size equal to 50, the maximum iterations to 100 and the mutation probability 1.0/N, N=33 in the case study of the autonomously navigating robot. Both heuristics evaluate 1,020 architecture alternatives.

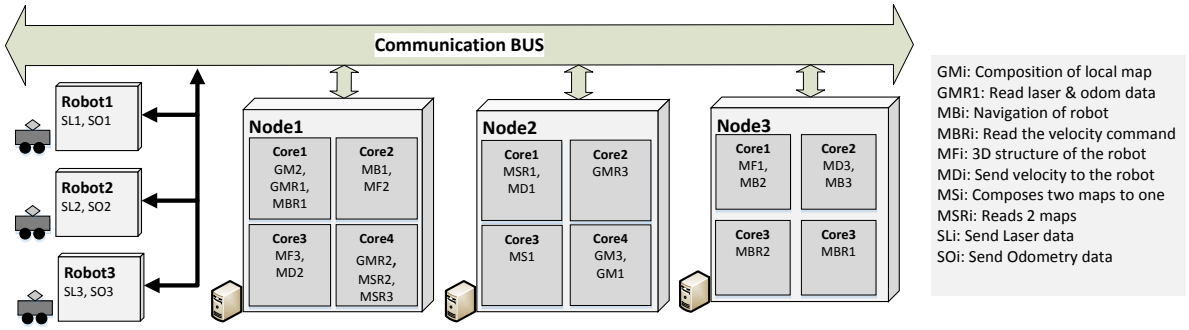

Fig. 5. Configuration of the optimal architecture alternative M, which is obtained after applying the OMOPSO-RB heuristic.

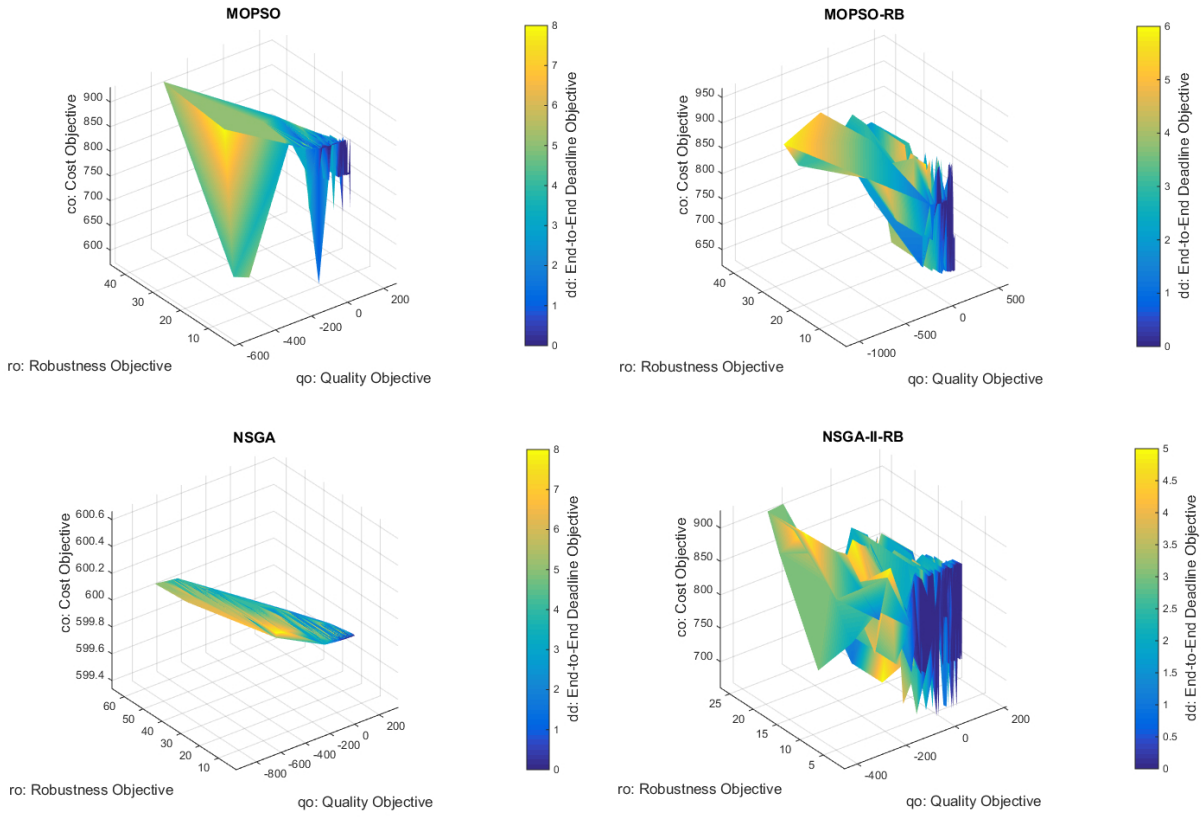

Fig. 6. The performance of the architecture alternatives explored by the four different optimization heuristics.

VII. Conclusions

In this paper we have presented a Design Space Exploration method for Component-Based Real-Time Distributed Systems. We have defined five different quality objective functions that represent defined requirements of an RTDS (end-to-end deadlines, robustness, timeliness quality, resource utilization and cost). We have experimented with two different heuristics, the NSGA-II (GA) and the OMOPSO. In addition, we have proposed a rule-based initial population for both heuristics. The extended OMOPSO-RB heuristic tends to find better solutions when a limited number of generation iterations is applied. Moreover, the *rule-based* initial population in both algorithms boosts the convergence speed, while maintaining the quality of the solutions. Especially, the NSGA-II is not able to find enough optimal solutions, while the NSGA-II-RB results into 20 solutions fulfilling the real-time requirements. Last but not least, the ProMARTES framework is an open-source project and available at [24]. In the near future, we are planning to apply the rule-based alternative generation during the whole optimization process, thereby increasing

the convergence speed. This is a challenging task, since the optimization process should still maintain generation of alternatives avoiding halts in local optimal space regions.

REFERENCES

[1] K. Triantafyllidis, E. Bondarev, and P. H. N. De With, "Promartes: Performance analysis method and toolkit for real-time systems," in *Languages, Design Methods, and Tools for Electronic System Design*, ser. Lecture Notes in Electrical Engineering, M.-M. Lourat and T. Maehne, Eds., 2015, vol. 311.

[2] K. Triantafyllidis, W. Aslam, E. Bondarev, J. Lukkien, and P. H. N. De With, "Promartes: Accurate network and computation delay prediction for component-based distributed systems," *submitted to The Journal of Systems and Software (JSS)*.

[3] K. Triantafyllidis, E. Bondarev, and P. H. N. De With, "Low-level profiling and marte-compatible modeling of software components for real-time systems," in *Software Engineering and Advanced Applications*, 2012, pp. 216–223.

[4] S. Määttä, L. Soares, L. Ost, and J. Nurmi, "Model based approach for heterogeneous application modelling for real time embedded systems."

[5] X. Wu and M. Woodside, "Performance modeling from software components," in *Proceedings of the 4th International Workshop on Software and Performance*, ser. WOSP '04. New York, NY, USA: ACM, 2004, pp. 290–301. [Online]. Available: http://doi.acm.org/10.1145/974044.974089

[6] S. Becker, H. Koziolek, and R. Reussner, "Model-based performance prediction with the palladio component model," in *Proceedings of the 6th International Workshop on Software and Performance*, ser. WOSP '07. ACM, 2007, pp. 54–65. [Online]. Available: http://doi.acm.org/10.1145/1216993.1217006

[7] A. D. Pimentel, M. Thompson, S. Polstra, and C. Erbas, "Calibration of abstract performance models for system-level design space exploration," *Signal Processing Systems*, vol. 50, no. 2, pp. 99–114, 2008.

[8] A. Pimentel, C. Erbas, and S. Polstra, "A systematic approach to exploring embedded system architectures at multiple abstraction levels," *Computers, IEEE Transactions on*, vol. 55, no. 2, pp. 99–112, Feb 2006.

[9] A. Aleti, S. Bjornander, L. Grunske, and I. Meedeniya, "Archeopterix: An extendable tool for architecture optimization of aadl models," in *Model-Based Methodologies for Pervasive and Embedded Software*, May 2009, pp. 61–71.

[10] A. Martens, H. Koziolek, S. Becker, and R. Reussner, "Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms," in *Proceedings of the First Joint WOSP/SIPEW International Conference on Performance Engineering*, ser. WOSP/SIPEW '10. New York, NY, USA: ACM, 2010, pp. 105–116. [Online]. Available: http://doi.acm.org/10.1145/1712605.1712624

[11] M. Lukasiewycz, M. Glaß, F. Reimann, and J. Teich, "Opt4j: A modular framework for meta-heuristic optimization," in *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '11. New York, NY, USA: ACM, 2011, pp. 1723–1730. [Online]. Available: http://doi.acm.org/10.1145/2001576.2001808

[12] V. Zaccaria, G. Palermo, F. Castro, C. Silvano, and G. Mariani, "Multicube explorer: An open source framework for design space exploration of chip multi-processors," in *Architecture of Computing Systems (ARCS), 2010 23rd International Conference on*, Feb 2010, pp. 1–7.

[13] C. Ykman-Couvreur, P. Avasare, G. Mariani, G. Palermo, C. Silvano, and V. Zaccaria, "Linking run-time resource management of embedded multi-core platforms with automated design-time exploration," *Computers Digital Techniques, IET*, vol. 5, no. 2, pp. 123–135, March 2011.

[14] E. Bondarev, M. Chaudron, and P. H. N. De With, "Carat: a toolkit for design and performance analysis of component-based embedded systems," in *Design, Automation Test in Europe Conf. Exhib., 2007.*, 2007, pp. 1–6.

[15] ——, "A process for resolving performance trade-offs in component-based architectures," in *Component-Based Software Engineering*.

[16] K. Triantafyllidis, E. Bondarev, and P. H. N. De With, "Performance analysis method for rt systems: Promartes for autonomous robot," in *Specification Design Languages (FDL), 2013 Forum on*, Sept 2013, pp. 1–8.

[17] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *Evolutionary Computation, IEEE Transactions on*, vol. 6, no. 2, pp. 182–197, Apr 2002.

[18] C. A. Coello Coello and M. Lechuga, "Mopso: a proposal for multiple objective particle swarm optimization," in *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*, vol. 2, 2002, pp. 1051–1056.

[19] J. J. Durillo and A. J. Nebro, "jmetal: A java framework for multi-objective optimization," *Advances in Engineering Software*, vol. 42, no. 10, pp. 760 – 771, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0965997811001219

[20] M. Sierra and C. Coello Coello, in *Evolutionary Multi-Criterion Optimization*, ser. Lecture Notes in Computer Science, C. Coello Coello, A. Hernndez Aguirre, and E. Zitzler, Eds., 2005, vol. 3410.

[21] R. Poli, J. Kennedy, and T. Blackwell, "Particle swarm optimization," *Swarm Intelligence*, vol. 1, no. 1, pp. 33–57, 2007. [Online]. Available: http://dx.doi.org/10.1007/s11721-007-0002-0

[22] M. Gonzalez Harbour, J. Gutierrez Garcia, J. Palencia Gutierrez, and J. Drake Moyano, "Mast: Modeling and analysis suite for real time applications," in *Real-Time Systems, 13th Euromicro Conference on, 2001.*, 2001, pp. 125–134.

[23] C. Cuesta, "Jsimmast: The performance analysis simulator for real-time," 2010, "http://mast.unican.es/jsimmast/index.html".

[24] K. Triantafyllidis, "Promartes: Profiling, modeling,analysis of real-time embedded systems," 2013, http://vca.ele.tue.nl/demos/ProMARTES.