Permutational genetic algorithm for the optimized mapping and scheduling of tasks and messages in distributed real-time systems

Ekain Azketa, Juan P. Uribe
Software Technologies
Ikerlan Research Center
Mondragón, Spain
{*eazketa, jpuribe*}*@ikerlan.es*

J. Javier Gutiérrez
Computers and Real-Time
University of Cantabria
Santander, Spain
gutierjj@unican.es

Marga Marcos
Systems Eng. and Automation
University of Basque Country
Bilbao, Spain
marga.marcos@ehu.es

Luís Almeida
Electrical and Computer Eng.
University of Porto
Porto, Portugal
lda@fe.up.pt

***Abstract*—The mapping of tasks and messages and the assignment of fixed priorities in distributed real-time systems are known to be NP-hard problems, and thus there are no optimal methods to accomplish them in polynomial time. This fact makes them suitable problems to be approached with generic search and optimization algorithms. In this paper we propose a genetic algorithm with a permutational solution encoding, which apart from solving the mapping and the fixed priority assignment problems using a holistic approach, can simultaneously minimize the average use of computing, memory and communication resources, the average worst-case response time of the transactions and the number of the used processors. The experimental results show that this genetic algorithm can find good solutions for industrial size distributed real-time architectures and in reasonable times from the perspective of a complex system design process.**

***Keywords*-distributed real-time; holistic; mapping; scheduling; priority assignment; genetic algorithm; linear programming**

I. Introduction

A common distributed real-time system is made up of time constrained applications composed of tasks and messages. The tasks execute in processors and the messages are transmitted through the real-time communication networks that connect the processors. The control of critical systems such as industrial, automotive, avionics, etc. is often accomplished using distributed real-time systems.

The design of distributed real-time systems is very challenging. On the one hand, applications and their functional and non-functional requirements (such as time constraints) have to be defined. On the other hand, the physical architecture has to be designed to accommodate the applications in such a way that all the constraints are met. The accommodation of the applications in the physical architecture involves some other difficulties: tasks and messages have to be mapped and scheduled in the processors and networks, respectively, i.e. where and when each task is going to execute and each message is going to be transmitted.

The scheduling can be viewed as the assignment of such scheduling parameters to tasks and messages that make the applications to execute within their deadlines. Depending on the type of scheduling -fixed or dynamic priority-based- the parameters may be priorities or deadlines, respectively.

Both the task scheduling and schedulability analysis in monoprocessor systems is an already solved problem. Although the schedulability analysis of a distributed real-time system can be made quite precisely with the holistic [1] or the offset-based [2], [3] methods, the scheduling is proved to be a NP-hard problem [4], and hence there are no algorithms to accomplish it optimally in polynomial time. The NP-hardness makes it suitable to be approached with generic search and optimization algorithms.

In this paper we propose a genetic algorithm that implements a permutational solution encoding for the simultaneous mapping of tasks and messages and the assignment of fixed priorities to them in distributed real-time systems using a holistic approach. This algorithm has been successfully applied to the assignment of priorities [5] and to the optimization of the network traffic in CAN bus [6]. Additionaly, the genetic algorithm can minimize the average use of computing, memory and communication resources, the average worst-case response time of the transactions and the number of the used processors. Similar approaches have been also applied in the literature to design system-on-chip architectures with cyclic schedulings, but as far as we know it has not been used to simultaneously map tasks and assign priorities in fixed priority-based distributed real-time systems. On the other hand, the vast majority of the work related to the mapping and priority assignment in distributed real-time systems do not use a holistic schedulability approach.

In order to validate our proposal, we compare the performance of the genetic algorithm with Mixed Integer Linear Programming (MILP), a mathematical method to model complex problems and find optimal solutions.

This paper is structured as follows. Section II discusses related work. Then, Section III describes the system model. Section IV explains the genetic algorithm with the permutational solution encoding while Section V shows the experiments and results that validate the proposed algorithm. Finally, Section VI outlines conclusions and future work.

II. Related Work

The mapping and scheduling of tasks and messages in distributed real-time systems have been addressed in the literature with different generic search and optimization algorithms. The proposal in [4] is a classical work that applies the simulated annealing algorithm to solve these problems, but it neither uses a transactional system model nor applies a holistic approach. The work in [7] proposes a branch and bound method, but it does not take into account memory issues, tasks have no mapping restrictions and deadlines are defined at task level. Dedicated heuristics have also been developed [8], [9], but obviating memory aspects. These problems have also been approached with genetic algorithms [10], [11], [12], [13], [14], but the system models differ substantially from ours because they consider non-preemptive cyclic scheduling and do not use a holistic approach.

Another set of works propose mathematical programming methods that in theory are able to obtain optimal solutions. The work in [15] uses a constraint satisfaction technique, but it neither optimizes nor applies holistic analysis. A satisfiability checking approach is proposed in [16] but it does not use holistic techniques. The linear programming is another optimization method that allows to model complex problems in terms of equations and inequalities and find optimal solutions by means of some well-known algorithms. The work done by Sangiovanni-Vicentelli's group in the use of Mixed Integer Linear Programming (MILP) methods to design distributed real-time systems is remarkable. Their proposals cover the synthesis of the activation paradigms of the tasks [17], the maximization of the extensibility of the system [18] and the minimization of the response times of the applications [19]. Although the latter work applies a periodic task activation model that is more restrictive than our holistic approach, its system model has similarities with our proposal. Moreover, it contains some interesting contributions, such as a rigorous mathematical modelling of a distributed real-time system mapping and fixed priority assignment problem and the application of MILP to solve it. This detailed model description combined with its theoretical optimality makes MILP a very suitable technique for comparison with our genetic algorithm.

III. System Model

The physical architecture is composed of several processors Ph ($h = 1, 2, ..., L_P$) connected to one or more communication networks Nh ($h = 1, 2, ..., L_N$). The processors can be heterogeneous in their core architecture, processing speed and maximum memory capacity $Sh^{\max}$. Each processor has maximum computing $UPh^{\max} \in [0, 1]$ and memory $USh^{\max} \in [0, 1]$ and each network has maximum communication $UNh^{\max} \in [0, 1]$ utilization limits imposed by the user. Each processor can provide the tasks different hardware or software resources such as sensors, actuators, programs, libraries, etc. The processors, or the operating systems they execute, implement preemptive fixed priority scheduling. The speed of the networks is known and they are scheduled by a packet-based preemptive fixed priority scheme, e.g. as in CAN bus.

The logical architecture consists of one or several transactions Aj ($j = 1, 2, ..., L_A$) composed of one task Ti ($i = 1, 2, ..., L_T$) or several tasks with precedence relations that exchange messages Mi ($i = 1, 2, ..., L_M$). Each transaction has defined an activation period or a minimum interarrival time between activations T_j, as well as a deadline D_j.

A task may have mapping restrictions in the processors, i.e. each task can be mapped in one, some or any processor of the architecture depending on the hardware and software resources offered by the processors and the resources required by the task. The processors where a task can be mapped are called the candidate processors of that task. The worst-case execution time C_i^h and the memory S_i^h required by a task depend on the processor computing and memory characteristics, and therefore they may be different in each of the candidate processors. A task has the period of the transaction to which it belongs. At first, tasks usually do not have specified priorities, but they may have defined deadlines that are relative to the event that triggers the activation of the transaction.

If the sender and receiver tasks of a message are mapped in the same processor, the message is exchanged through a shared memory mechanism which transmission time is considered negligible. Otherwise, the message is transmitted through one network. In the latter case, messages are modeled like tasks with the exception that they do not have memory requirements.

The schedulability of this kind of systems can be analyzed using a holistic approach, understanding holistic as the simultaneous consideration of both processors and networks in the schedulability test. There are some holistic analysis methods such as the Tindell's technique [1] and Palencia's offset based techniques [2], [3] which allow to calculate the worst-case response time R_j of each transaction in order to determine whether it meets the imposed deadline.

IV. Genetic Algorithm

A genetic algorithm [20] is a search and optimization metaheuristic based on evolving an initial population of candidate solutions -individuals- to the problem towards better solutions through generations of populations by means of biologically inspired techniques such as inheritance, natural selection, crossover and mutation. Initially, a generation of individuals is created, usually applying random or heuristic methods. The fitness of each individual is computed and pairs of them are selected according to their fitness: the larger the fitness, the bigger the probability to be selected. Those pairs of individuals crossover to generate two new individuals that may suffer a mutation and may be members

of the next generation, which will inherit part of the solutions of the previous generation. The new generation is used in the next iteration of the algorithm, which commonly finishes when some individual has reached a satisfactory fitness level or when a maximum number of generations have been created.

The genetic algorithm presents some advantages over other search and optimization techniques. The solution space can be non-linear and since it operates with multiple and possibly very different solutions at the same time, it is less vulnerable to converge towards local optimal solutions. Furthermore, the genetic algorithm can optimize several different objectives simultaneously and can be quite easily extended by adding new weighted factors to the fitness function. Moreover, its execution can be readily parallelized for the proper exploitation of the modern multicore and multiprocessor architectures.

A search and optimization problem consists of assigning values to some variables in a way that all the restrictions are met and some function is maximized or minimized. In the present mapping and scheduling problem the value assignment is done to the following variables:

- Map each task and message to one of its candidate processors and networks, respectively.
- Assign each task and message a priority different from the priorities of the tasks and messages mapped in the same processor and network, respectively.

The restrictions are the following:

- Not exceed the maximum capacity of memory ($USh^{\max}$), processors ($UPh^{\max}$) and networks ($UNh^{\max}$).
- The worst-case response time R_j of each transaction has to be less than or equal to the deadline D_j of the transaction.

The factors that can be minimized are these ones:

- Average computation, memory and communication resource utilizations of the architecture.
- Average worst-case response time of the transactions.
- Number of used processors.

In this work we propose to use a genetic algorithm with a permutational solution encoding. In the following lines the representation, initial population creation, crossover, mutation and clustering operations of the genetic algorithm are exposed.

A. Representation

A candidate solution is a set of concrete values of the variables. In a genetic algorithm, a variable is encoded with an element called gene and a candidate solution is encoded with a string of genes called chromosome. Usually, each gene representing a variable has a binary encoded changeable value and a fixed position in the chromosome. However, there exist some other type of encodings, such as the permutation representation [21], which is used to search the optimal ordering of a set of elements. Unlike in the common representation, the permutation chromosome is encoded with a string of genes whose values are fixed but positions are changeable. Genetic algorithms based on the permutational encoding have been used to solve some classical NP-hard problems, such as Travelling Salesman Problem and Job-shop Scheduling Problem (JSP) [22]. As flexible JSP is a distributed system mapping and scheduling, it has some similarities with the present problem, but the differences are also substantial: JSP searches for a task execution order instead of a priority assignment, tasks are non-preemptive, there is no communication, the schedulability analysis is much simpler, etc. In this paper we propose to use a hybrid value-permutational representation that is based on the classical permutation representation [21] for encoding the candidate solutions of the mapping and the fixed priority assignment problems. Although the hybrid encoding has some structural similarities with [23], the latter approaches a JSP and thus the characteristics of the problems differ substantially.

The mapping and the fixed priority assignment problems consists of some variables representing the mapping and priority of the tasks in the processors and the messages in the networks. Each variable of the problem has an associated gene with the structure shown in Figure 1. We define a gene that is composed of two fields. *Code* is a fixed field that stores the name of the variable (in this case Ti for tasks and Mi for messages), which has to be unique for all variables. *Value* is a changing field that represents the mapping of the associated variable. The value in this field is always one among the candidate processors (Ph) in the case of task genes and possible networks (Nh) in the case of message genes. Additionally, the relative position of the gene in the chromosome with respect to other genes with the same value denotes the relative priority of the associated task or message in its processor or network, respectively. The more to the left, the larger the priority.

The hybrid value-permutational encoding has some advantages. The gen value is a candidate value and thus it is always a valid mapping value. Further, the gen position is unique and thus the priority is always different from the priorities of the other variables with the same mapping value. Additionally, it can manage legacy tasks that are mapped in concrete processors or have already assigned priorities. In brief, it allows an efficient representation of both task and message mapping and priority assignment that supports a simultaneous search in both dimensions, simplifying the whole problem solution and optimization process.

B. Initial population

The initial population is the set of individuals -candidate solutions- that are evolved by the genetic algorithm. It is essential that the fitness of the candidate solutions can be

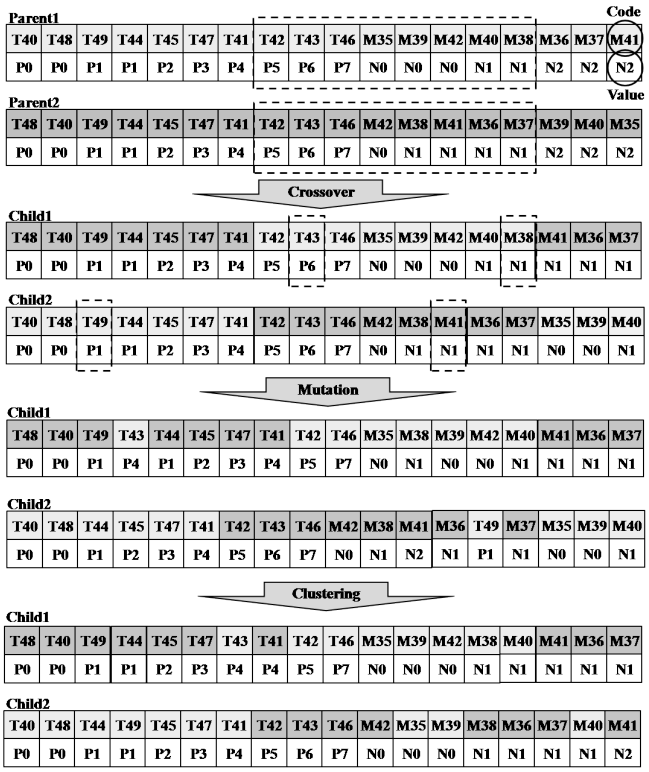

Figure 1. Crossover, mutation and clustering operations

calculated, and the larger the fitness the better. It is also very important that the initial population contains a high degree of genetic variety, i.e. the candiate solutions must have significant differences from each other.

Our system creates the initial population in three phases. In the first step all the individuals are generated randomly. Since this randomness may create candidate solutions with processing utilizations more than 100% and hence not schedulable (invalid), in the second phase tasks of the overloaded processors are selected randomly and moved to their candidate processors with least load until the processing utilization excess is eliminated. In the last step HOPA [24] heuristic is used. HOPA is a reference method for the assignment of fixed priorities in distributed real-time systems and works as follows.

Firstly, HOPA carries out an initial arbitrary local deadline assignment on the condition that the summation of them does not exceed the global deadline of the transaction. After that, it assigns priorities to tasks and messages according to the DMS scheme. Then, it applies a holistic schedulability analysis over the whole system. The excess of the response times of the transactions with respect to their deadlines are used to compute some metrics, and the local deadlines - and hence the priorities- of tasks and messages are adjusted according to those excess. The system with the new priority assignment is holistically analyzed again, and so on until a schedulable solution, a satisfactory level of optimization or a maximum limit of iterations is reached.

HOPA allows to configure two constant factors, k_a and k_r, that control the relative influence of the resources (processors and networks) and activities (tasks and messages), respectively, in the calculation of the new local deadlines. The smaller these constants are, the higher the influence is. These factors have a direct impact on the quality of the obtained schedulings, and different values for k_a and k_r give different priority assignments even over the same architecture. We use this behaviour to create initial candidate solutions with different partial schedulings by executing HOPA over each one of them a random integer number of iterations in the $[10, 50]$ interval and with a random rational number in the $(0, 10]$ interval selected independently for k_a and k_r for each candidate. This way, the initial population ends up being composed of analyzable candidate solutions with pseudorandom mappings and schedulings. Since the scheduling analyzability of the candidates facilitates their fitness evaluation and their pseudorandomness guarantees the genetic variety, the algorithm is able to evolve the candidates to better solutions.

C. Crossover

The genetic crossover operator combines information from two parent chromosomes to create two children chromosomes. Due to the characteristics of the encoding used in this algorithm, the crossover operator has to be able to recombine information about the relative position of the genes of both parents.

This genetic algorithm uses the OX3 [21] crossover operator, because it respects strictly the relative position, the absolute position and the value of the recombined genes of one parent chromosome, and also respects the relative position and the value of the recombined genes of the other parent chromosome. OX3 chooses two cut-points randomly and the block of the genes of the first (second) parent between those two points is inherited directly by the first (second) child in the same absolute position. The genes not included in the inherited block are taken from the other parent chromosome in strict order.

Figure 1 shows the crossover operation and the resulting children chromosomes, where the cut-points are the genes with absolute positions 8 and 15.

D. Mutation

The mutation is an operator that maintains the genetic diversity of the population through random changes in the information of the genes. The main objective of the mutation is to avoid the situation in which the individuals of a population resemble each other too much after some generations. Usually, the mutation is applied with a very low probability to each gene of the children chromosomes.

Our genetic algorithm defines a different mutation operator for each one of the two types of information that a gene can contain. The *position mutation* operator changes the position of the gene in the chromosome, and hence the information about the priority of the variable may be altered. The *value mutation* operator changes the value of a gene to another candidate value, and therefore the information about the mapping of the variable is altered. Each candidate value of every gen is configured with a related *value mutation probability factor*. This numerical factor is multiplied with the general mutation probability in order to increase it, which may be adequate in some circumstances, e.g. gen values that are less preferable than others because can deteriorate the fitness of the chromosome may have greater factors, thus increasing their mutation probability and facilitating their change to other values.

Figure 1 shows the mutation operation over the children chromosomes. Specifically, gen *T43* of *Child1* suffers value and position mutations, genes *M38* of *Child1* and *T49* of *Child2* suffer position mutations, and gen *M41* of *Child2* suffers a value mutation.

E. Clustering

After the crossover and mutation operations the clustering of genes is carried out with the aim of grouping the related genes side by side and creating a variable contiguous block per processor and network. This operation increases the probability of the crossover operator to transmit to the children chromosomes blocks of variables whose relationship makes them more likely to belong with the current values to a solution. The clustering reorders the genes but always maintaining the relative order of task and message genes with respect to other genes with the same value, because otherwise the priority assignment done by the algorithm would be altered. The chromosome is clustered according to the value of the genes in ascending order of processors (*P0*, *P1*, etc.) and networks (*N0*, *N1*, etc.) and always strictly respecting the relative order between them.

Figure 1 shows the clustering operation over the two children chromosomes. In the example, the genes *T43* and *M38* of the *Child1* and the genes *T49*, *M35*, *M39* and *M41* of the *Child2* are reordered.

F. Fitness Function

In a genetic algorithm the fitness function checks how well a candidate solution solves the problem, which is given by its restriction fulfillment degree and optimization level. A candidate is a valid solution if and only if it fulfills all the restrictions, and the greater the optimization level, the better the valid solution. The fitness function is a weighted sum of five partial fitness functions, one per restriction factor and one for the optimizable factor.

Table I
PARAMETERS OF RESTRICTION FACTORS FITNESS FUNCTIONS

f	g_h	Maximum limit	L
f_p	$UPh^{\max} - \sum_{\forall Ti\ in\ Ph} \frac{C_i^h}{T_i}$	$UPh^{\max} \in [0,1]$	L_P
f_s	$USh^{\max} - \sum_{\forall Ti\ in\ Ph} \frac{S_i^h}{Sh^{\max}}$	$USh^{\max} \in [0,1]$	L_P
f_n	$UNh^{\max} - \sum_{\forall Mi\ in\ Nh} \frac{C_i^h}{T_i}$	$UNh^{\max} \in [0,1]$	L_N
f_t	$1 - (R_j/D_j)$	$D_j > 0$	L_A

1) Restriction Factors Fitness: As referred earlier, there are four restriction factors, three of them related with resource utilizations and the other with the response time, which can be considered as another resource. We define a generic restriction fitness function (1) inspired by the scheduling index factor defined in [24] and that is valid for all of them. The restriction fitness function f computes the average of the resource slacks g_h, which is the subtraction between the normalized maximum utilization limit and the normalized current utilization. A negative slack represents the violation of the restriction of not exceeding the maximum utilization limit, and therefore denotes an invalid solution. If at least one resource slack is negative, the resource fitness f is the average value of only the negative parts of the resource slacks.

$$f = \begin{cases} \frac{1}{L} \cdot \sum_{h=1}^{L} g_h, & \text{if } \forall h : g_h \geq 0 \\ \frac{1}{L} \cdot \sum_{h=1}^{L} min[0, g_h], & \text{if } \exists h : g_h < 0 \end{cases} \quad (1)$$

Replacing f, g_h and L of Table I in Equation (1), the partial fitness functions of computing (f_p), memory (f_s), communications (f_n) and response time (f_t) are obtained. Note that f_t is the average of the scheduling index factor [24]. The worst-case response time R_j of each transaction can be computed by any holistic schedulability analysis technique, such as the Tindell's [1] or the offset-based [2], [3] methods. The current implementation of the genetic algorithm uses MAST [25], a free tool suite that provides several holistic schedulability analysis techniques, but the algorithm can be quite easily modified to invoque any other tool that uses a holistic approach.

2) Optimizable Factor Fitness: The genetic algorithm can minimize the average utilization of computing, memory and communication resources and the response time of the transactions. Nevertheless, beyond those restriction factors, we chose to add another optimization parameter to contribute positively to the overall fitness function, namely the number of processors, aiming at reducing this number. The computing utilization of a processor is Equation (2).

$$UPh = \sum_{\forall Ti \ in \ Ph} \frac{C_i^h}{T_i} \tag{2}$$

We consider a processor being dispensable if all the tasks for whom it is a candidate processor have more than one candidate processor. The set of dispensable processors is represented by P_{dis}, and $\overline{UP}_{\text{dis}}$ (3) is their average computing utilization. ($card(A)$ gives the number of elements in the set A.)

$$\overline{UP}_{\text{dis}} = \sum_{\forall Ph \ \in \ P_{\text{dis}}} \frac{UPh}{card(P_{\text{dis}})} \tag{3}$$

The dispensable processor penalty Y_h, $Ph \in P_{\text{dis}}$ (4) is the weighted sum of three factors. The first one increases the penalty if the corresponding dispensable processor is used; the second increases the penalty with the utilization level of the processor; and the last factor reduces the penalty with the utilization deviation of the processor, because a bigger deviation means that some processors are closer from zero utilization and thus closer from being dispensed. The sum of the weights has to be 1 and good values are $w_u^1 = 0.6$, $w_u^2 = 0.3$ and $w_u^3 = 0.1$.

$$Y_h = w_u^1 \cdot \lceil UPh \rceil + w_u^2 \cdot UPh + w_u^3 \cdot \left(1 - \left| UPh - \overline{UP}_{\text{dis}} \right| \right) \tag{4}$$

The used processor fitness function f_u is computed with Equation (5).

$$f_u = 1 - \sum_{\forall Ph \ \in \ P_{\text{dis}}} \frac{Y_h}{card\,(P_{\text{dis}})} \tag{5}$$

3) Total Fitness: The total fitness F is computed by the weighted sum of f_p, f_s, f_n, f_t and f_u. Negative partial fitness denotes a restriction violation and therefore it is an invalid solution. If at least one partial fitness is negative, the total fitness is the sum of only the negative ones. Being $K = \{p, s, n, t, u\}$, Equation (6) is the total fitness function.

$$F = \begin{cases} \sum\limits_{\forall k \ \in \ K} w_k \cdot f_k, & \text{if } \bigwedge\limits_{\forall k \ \in \ K} (f_k \geq 0) \\ \sum\limits_{\forall k \ \in \ K} w_k \cdot min[0, f_k], & \text{if } \bigvee\limits_{\forall k \ \in \ K} (f_k < 0) \end{cases} \tag{6}$$

The weights w_p, w_s, w_n, w_t and w_u denote the importance given by the genetic algorithm to the corresponding factor in the solution searching process. The weights can be configured but taking into account that their sum has to be 1, that a value of 0 means that the corresponding factor will not be considered, and that the bigger the value, the more the importance of the factor during the optimization process.

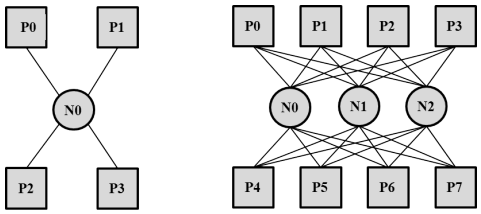

Figure 2. Physical arquitectures: (a) Small system. (b) Large system

V. Experiments

The objectives of the following experiments are to test the genetic algorithm in terms of time to search and optimize valid solutions, their quality, and compare its performance with the MILP technique. The experiments are carried out over a small and a large distributed system based on [24] and composed of different number of processors, networks, transactions, tasks and messages. The main difference of our systems with respect to [24] is that the mapping is not fixed, i.e. almost all the tasks, and in the large system all the messages, have more than one candidate processor and network where they can be mapped, respectively.

The physical architecture of the large distributed real-time system is composed of 8 processors connected to 3 networks, as can be seen in Figure 2-(b). In this scenario the networks are equal but the processors have differences in their core architecture, speed and memory resources (see Table IV). On the other hand, the logical architecture has 7 transactions composed by tasks and messages. The structure of those transactions is shown in Figure 3 and their periods and deadlines in Table II. Each task has different worst-case execution time (C_i^h) and memory requirement (M_i^h) in each of its candidate processors, e.g. due to different processors and/or different task implementations, as described in Table III and IV, respectively. Table V-(a) shows the worst-case transmission times of the messages in the networks of the physical architecture. The maximum resource utilizations are configured as $UPh^{\max} = USh^{\max} = UNh^{\max} = 1$.

The small distributed real-time system is a subset of the large scenario. Its physical architecture is formed by 4 processors and 1 network of the large scenario (see Figure 2-(a)). Despite tasks and messages having the same computing, memory and transmission time characteristics, the small scenario has 4 transactions created by cutting some transactions of the large system (framed ones in Figure 3), and which periods and deadlines are shown in Table II.

The mapping and the fixed priority assignment process of the distributed real-time systems is started with each method and samples of best solutions are stored during the process. When the executions finish, the system load is altered by increasing the C_i^h of every task by 8 units of time in the small system and 4 units of time in the large one, and the

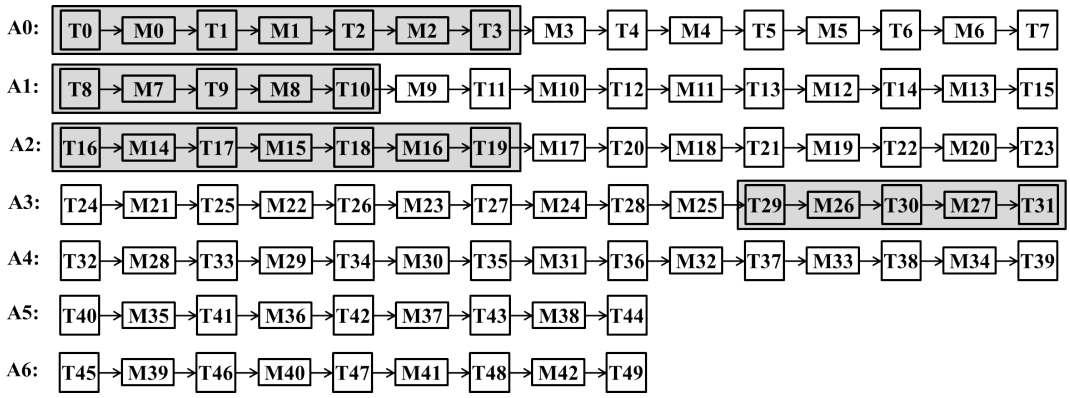

Figure 3. Structure of the transactions (framed ones are small system)

Table II
PERIODS AND DEADLINES

Transaction	Small system		Large system	
	Period	Deadline	Period	Deadline
A0	300	3000	400	4000
A1	400	4000	500	5000
A2	500	5000	700	7000
A3	300	3000	450	4500
A4	-	-	550	5500
A5	-	-	350	3500
A6	-	-	500	5000

experiments are repeated; and so on until 5 incrementations of C_i^h have been made. Note that in these experiments memory requirements do not increase and memory resources are large enough to accomodate the tasks. Nevertheless, the genetic algorithm can also deal properly with memory constraints as has been tested with some experiments.

The genetic algorithm is configured with a population of 100 individuals. In the initial population creation, the k_a and k_r pair of HOPA parameters are configured with all the combinations of $\{1, 2, 3\}$ set of values, and executed with each candidate solution in order to store the obtained best scheduling of those 9 executions. The selection method is a tournament of 2 individuals. The crossover and mutation probabilities are 0.8 and 0.005, respectively, because they have been identified empirically as good values. The mutation probability factor is configured as $10 * w_u$ for dispensable processors and 1 for the others. The genetic algorithm runs across 1000 generations and in each of them 10 new individuals are created. The new individuals are included in the population and the best 100 of those 110 individuals form the next generation. The worst-case response times are computed with the Tindell's holistic schedulability analysis [1] implementation of the MAST tool, essentially because it is faster -although more pessimistic- than the offset-based [2], [3] techniques.

The MILP formulation of these experiments is based on [19], but has some differences: we do not map signals but map messages and assign priorities to them, and we apply a holistic approach [1] -including jitter and blocking terms in tasks and messages-, which is less restrictive than the periodic activation model. The function to maximize is the fitness of Equation (6). SCIP is used as MILP solver since it is currently one of the fastest non-commercial tools [26].

The weights are configured as $w_p = w_s = w_n = 0.01$, $w_t = 0.02$, $w_u = 0.95$. This way, the minimization of the used processors becomes the main objective, whereas the worst-case response time minimization has a secondary importance and the computing, memory and communication resource utilizations are considered only restrictions.

The experiments are carried out on a dual-core Intel i5-650 processor running at 3.2GHz with 4GB of memory. The parallelism of the multi-core architecture is properly exploited by multithreading the schedulability analysis.

A. Results

We draw some graphs to show the results obtained by both techniques with different average utilizations of processing and communication resources in the small and the large scenarios. The graphs trace the fitness (F), the number of used processors and the average utilization of the obtained best solutions with respect to the execution time. Solutions with fitness values equal to or larger than 0 are valid solutions, i.e. they meet all the restrictions.

Figure 4 shows the best solutions obtained by the genetic algorithm and MILP in the small problem during 2000 seconds. The figure allows to compare both techniques directly for each utilization level. The first obvious conclusion is that with bigger systems loads both the genetic algorithm and MILP obtain worse solutions (less fitness and more used processors) and need more time to find them. However, even though those amounts of time are reasonable from the perspective of a complex system design process, the genetic algorithm converges to good solutions faster than MILP, and in some cases it finds better solutions (see the second row of graphs). Furthermore, the genetic algorithm is able to

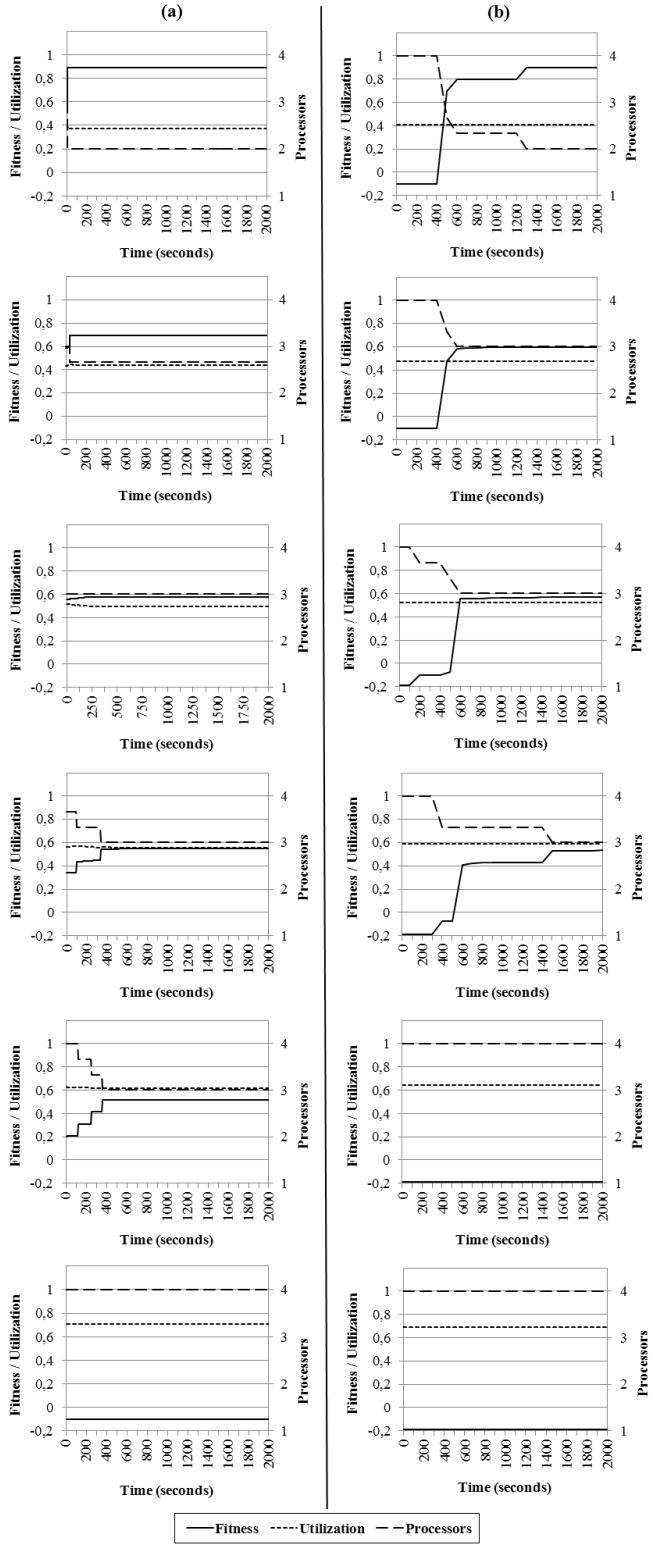

Figure 4. Results in the small system: (a) Genetic algorithm. (b) MILP

Table III
WORST-CASE EXECUTION TIMES

	P0	P1	P2	P3	P4	P5	P6	P7
T0	48	26	35	50	36	40	45	27
T1	36	52	59	34	26	43	50	48
T2			49					
T3				50				
T4					46			
T5						50		
T6							51	
T7	26	35	29	42	50	37	20	52
T8	50	40	31	29	33	46	37	29
T9			52					
T10	40	39	50	33	36	39	43	41
T11					50			
T12				39				
T13							38	
T14						52		
T15	33	46	37	29	35	29	42	50
T16		32		55		29		
T17			51		26			
T18	51	42	26	35	50	36	33	29
T19		50		28			36	
T20	35	29	42	42	26	35	50	54
T21						29	50	
T22						48	32	
T23			31		45			
T24	53	26	35	50	36	26	35	50
T25						48	36	
T26			26		50			
T27		51		30				50
T28						32	39	
T29	50	48	36	26	35	26	35	50
T30	31	39		44				
T31			52		27			
T32	44	26	35	26	35	30	28	49
T33					30	40	50	
T34					49	39	29	
T35					53	25	40	
T36		32	40	48				
T37		30	50	45				
T38		54	28	36				
T39	50	28	39	47	44	35	26	35
T40	52	28	35	26	35	50	45	33
T41					48	38	28	
T42					30	50	40	
T43					26	38	48	
T44		51	30	42				
T45		34	49	30				
T46	28	39	47	50	26	35	50	51
T47		36	29	53				
T48	51	44	33	28	50	26	39	33
T49	37	50	52	32	29	27	36	40

find valid solutions with larger utilization levels than MILP, as can be seen in the fifth row of graphs, where the latter is unable to obtain a valid result. These experiments show that our genetic algorithm with the permutational solution encoding is a suitable method to solve the mapping and the priority assignment problems in small distributed real-time systems, and according to the results, better than MILP.

Figure 5 shows the results obtained by the genetic algorithm in the large scenario. In this problem instance the genetic algorithm starts failing with average resource

Table IV
MEMORY REQUIREMENTS

	P0	P1	P2	P3	P4	P5	P6	P7
T0	192	104	140	200	144	160	180	108
T1	144	208	236	136	104	172	200	192
T2			196					
T3				200				
T4					184			
T5						200		
T6							204	
T7	104	140	116	168	200	148	80	208
T8	200	160	124	116	132	184	148	116
T9			208					
T10	160	156	200	132	144	156	172	164
T11					200			
T12				156				
T13							152	
T14						208		
T15	132	184	148	116	140	116	168	200
T16		128		220		116		
T17			204		104			
T18	204	168	104	140	200	144	132	116
T19		200		112			144	
T20	140	116	168	168	104	140	200	216
T21						116	200	
T22						192	128	
T23			124		180			
T24	212	104	140	200	144	104	140	200
T25						192	144	
T26			104		200			
T27		204		120				200
T28						128	156	
T29	200	192	144	104	140	104	140	200
T30	124	156		176				
T31			208		108			
T32	176	104	140	104	140	120	112	196
T33					120	160	200	
T34					196	156	116	
T35					212	100	160	
T36		128	160	192				
T37		120	200	180				
T38		216	112	144				
T39	200	112	156	188	176	140	104	140
T40	208	112	140	104	140	200	180	132
T41					192	152	112	
T42					120	200	160	
T43					104	152	192	
T44		204	120	168				
T45		136	196	120				
T46	112	156	188	200	104	140	200	204
T47		144	116	212				
T48	204	176	132	112	200	104	156	132
T49	148	200	208	128	116	108	144	160
$S_h^{\max}$	18300	19000	21000	20000	20400	16000	15500	18200

Table V
WORST-CASE TRANSMISSION TIMES

(a)	N0	N1	N2
M0	11	11	11
M1	11	11	11
M2	12	12	12
M3	13	13	13
M4	14	14	14
M5	13	13	13
M6	12	12	12
M7	11	11	11
M8	10	10	10
M9	9	9	9
M10	8	8	8
M11	11	11	11
M12	10	10	10
M13	11	11	11
M14	9	9	9
M15	10	10	10
M16	11	11	11
M17	12	12	12
M18	13	13	13
M19	8	8	8
M20	10	10	10
M21	12	12	12
M22	11	11	11
M23	10	10	10
M24	10	10	10
M25	8	8	8
M26	10	10	10
M27	10	10	10
M28	11	11	11
M29	10	10	10
M30	9	9	9
M31	8	8	8
M32	9	9	9
M33	10	10	10
M34	11	11	11
M35	10	10	10
M36	12	12	12
M37	13	13	13
M38	8	8	8
M39	10	10	10
M40	12	12	12
M41	11	11	11
M42	10	10	10

(b)	N0	N1	N2
M0	11		
M1	11		
M2	12		
M3	13		
M4	14		
M5	13		
M6	12		
M7	11		
M8	10		
M9	9		
M10	8		
M11	11		
M12	10		
M13	11		
M14		9	
M15		10	
M16		11	
M17		12	
M18		13	
M19		8	
M20		10	
M21		12	
M22		11	
M23		10	
M24		10	
M25		8	
M26		10	
M27		10	
M28			11
M29			10
M30			9
M31			8
M32			9
M33			10
M34			11
M35			10
M36			12
M37			13
M38			8
M39			10
M40			12
M41			11
M42			10

utilizations of 0.67. However, it has a great ability to find valid solutions and optimize them in terms of maximizing the fitness and minimizing the number of used processors in reasonable times. On the other hand, the MILP technique, whose mathematical model has 61792 variables and 346128 constraints after the presolving phase, is unable to find any valid mapping and fixed priority assignment even in a time scale of days, and that is why its results are not graphed. So our experiments corroborate that MILP models for the mapping and scheduling of distributed real-time systems are tipically too complex for the sizes of industrial applications [19] and therefore cannot find solutions in acceptable time frames.

The high complexity of some MILP models can be faced by decomposing the whole problem into two less complex subproblems [19]. Based on this approach, we propose a divide-and-conquer strategy. First, the mapping of the messages is fixed (see Table V-(b)) and their priority is assigned by Rate Monotonic Scheduling. The objective becomes the minimization of the used processors, but always meeting all the restrictions. The solution of this first subproblem is a mapping and a priority assignment for the tasks. In the second phase, this task mapping is used to find a mapping for the messages and priority assignments for both messages and tasks with the objective of maximizing the scheduling index. This decomposition process facilitates the solution of

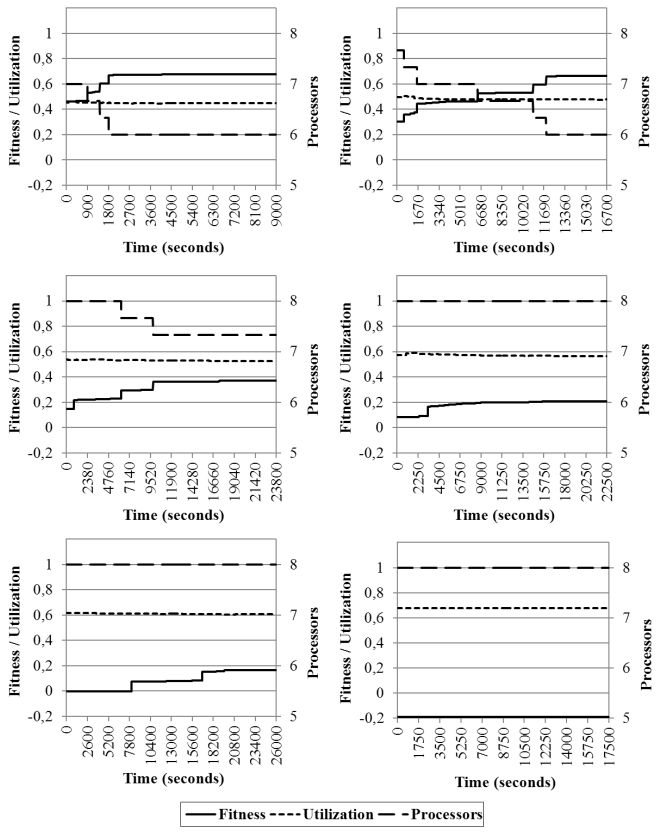

Figure 5. Best solutions of the genetic algorithm in the large system

the problem, and therefore the genetic algorithm is likely to find good results, which is confirmed by some experiments that are not included in this paper. However, the MILP technique, whose model has 31926 variables and 204944 constraints after the presolving phase, is still unable to obtain valid solutions even in a time scale of days. Although the two-phase MILP model converged satisfactorily in [19], it is very important to note that the cited work applies a periodic activation model with a simplified method for the worst-case response time computation in opposition to our holistic analysis, which is less restrictive but more complex, and thus harder to converge.

These results let us conclude that our genetic algorithm is more suitable than MILP to solve the mapping of tasks and messages and the fixed priority assignment to them in both small and large distributed real-time arquitectures based on the system model described in Section III.

VI. Conclusions and Future Work

Mapping tasks and messages and assign fixed priorities to them in distributed real-time systems is still an open issue. The NP-hardness of these problems precludes the existence of optimal algorithms to solve them in bounded time and makes them very suitable to be approached with generic search and optimization methods. In this paper we propose a genetic algorithm with a permutational solution encoding for the mapping of tasks and messages and the assignment of fixed priorities to tasks and messages in distributed real-time systems. Our genetic algorithm solves these problems using a holistic approach and simultaneously can minimize the average utilization of computing, memory and communication resources, the average worst-case response time of the transactions and the number of used processors in industrial size distributed real-time systems. We compare this genetic algorithm with the Mixed Integer Linear Programming (MILP) technique and conclude that the genetic algorithm can find better solutions and faster, obtaining good results even in problem instances for which MILP is unable to find any valid solution in reasonable time frames.

On the other hand, the permutational solution encoding of our genetic algorithm could be used not only for the mapping and fixed priority assignment but also for the network topology optimization of distributed real-time systems by mainly making the processors mapeable in subnets and adding some weigthed factors to the fitness function. In future work we plan to explore the network topology optimization, as well as the mapping and the fixed priority assignment in systems with tighter memory constraints and legacy tasks.

References

[1] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocessing and microprogramming*, vol. 40, no. 2-3, pp. 117–134, 1994.

[2] J. Carlos Palencia and M. González Harbour, "Schedulability analysis for tasks with static and dynamic offsets," in *Proceedings of the 19th IEEE Real-Time Systems Symposium*. Washington, DC, USA: IEEE Computer Society, 1998, pp. 26–37.

[3] ——, "Exploiting precedence relations in the schedulability analysis of distributed real-time systems," in *Proceedings of the 20th IEEE Real-Time Systems Symposium*. Washington, DC, USA: IEEE Computer Society, 1999, pp. 328–339.

[4] K. Tindell, A. Burns, and A. Wellings, "Allocating hard real-time tasks: an NP-hard problem made easy," *Real-Time Systems*, vol. 4, no. 2, pp. 145–165, 1992.

[5] E. Azketa, J. Uribe, M. Marcos, L. Almeida, and J. Gutiérrez, "Permutational genetic algorithm for the optimized assignment of priorities to tasks and messages in distributed real-time systems," in *2011 International Joint Conference of IEEE TrustCom-11/IEEE ICESS-11/FCST-11*. IEEE, 2011, pp. 958–965.

[6] E. Azketa, J. Uribe, M. Marcos, L. Almeida, and J. Javier Gutiérrez, "Permutational genetic algorithm for fixed priority scheduling of distributed real-time systems aided by network segmentation," in *Proceedings of the 1st Workshop on Synthesis and Optimization Methods for Real-time Embedded Systems (SOMRES)*, 2011.

[7] M. Richard, P. Richard, and F. Cottet, "Allocating and scheduling tasks in multiple fieldbus real-time systems," in *Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation*, vol. 1, 2003.

[8] P. Pop, P. Eles, Z. Peng, and T. Pop, "Analysis and optimization of distributed real-time embedded systems," *ACM Transactions on Design Automation of Electronic Systems*, vol. 11, pp. 593–625, June 2004. [Online]. Available: http://doi.acm.org/10.1145/996566.1142984

[9] T. Pop, P. Pop, P. Eles, and Z. Peng, "Optimization of hierarchically scheduled heterogeneous embedded systems," in *Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2005, pp. 67–71.

[10] Y. Monnier, J. Beauvais, and A. Deplanche, "A genetic algorithm for scheduling tasks in a real-time distributed system," in *Proceedings of the 24th Euromicro Conference*, vol. 2, 1998, pp. 708–714.

[11] R. Dick and N. Jha, "MOGAC: a multiobjective genetic algorithm for hardware-software cosynthesis of distributed embedded systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 10, pp. 920–935, 1998.

[12] J. Oh and C. Wu, "Genetic-algorithm-based real-time task scheduling with multiple goals," *Journal of Systems and Software*, vol. 71, no. 3, pp. 245–258, 2004.

[13] L. Shang, R. Dick, and N. Jha, "SLOPES: Hardware–Software Cosynthesis of Low-Power Real-Time Distributed Embedded Systems With Dynamically Reconfigurable FPGAs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 3, pp. 508–526, 2007.

[14] M. Yoo, "Real-time task scheduling by multiobjective genetic algorithm," *Journal of Systems and Software*, vol. 82, no. 4, pp. 619–628, 2009.

[15] P.-E. Hladik, H. Cambazard, A.-M. Déplanche, and N. Jussien, "Solving a real-time allocation problem with constraint programming," *Journal of Systems and Software*, vol. 81, no. 1, pp. 132–149, 2008.

[16] A. Metzner and C. Herde, "Rtsat– an optimal and efficient approach to the task allocation problem in distributed architectures," in *Proceedings of the 27th IEEE International Real-Time Systems Symposium*, 2006, pp. 147–158.

[17] W. Zheng, M. Di Natale, C. Pinello, P. Giusto, and A. S. Vincentelli, "Synthesis of task and message activation models in real-time distributed automotive systems," in *Proceedings of the conference on Design, automation and test in Europe*, 2007, pp. 93–98.

[18] Q. Zhu, Y. Yang, E. Scholte, M. D. 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*, 2009, pp. 275–284.

[19] W. Zheng, Q. Zhu, M. D. Natale, and A. S. Vincentelli, "Definition of task allocation and priority assignment in hard real-time distributed systems," in *Proceedings of the 28th IEEE International Real-Time Systems Symposium*, 2007, pp. 161–170.

[20] J. Holland, "Genetic algorithms," *Scientific American*, vol. 267, no. 1, pp. 66–72, 1992.

[21] L. Davis, *Handbook of genetic algorithms*. Arden Shakespeare, 1991.

[22] M. Garey, D. Johnson, and R. Sethi, "The complexity of flowshop and jobshop scheduling," *Mathematics of Operations Research*, pp. 117–129, 1976.

[23] I. Kacem, "Genetic algorithm for the flexible job-shop scheduling problem," in *IEEE International Conference on Systems, Man and Cybernetics*, vol. 4, 2003, pp. 3464–3469.

[24] J. Javier Gutiérrez and M. González Harbour, "Optimized priority assignment for tasks and messages in distributed hard real-time systems," in *Proceedings of the 3rd Workshop on Parallel and Distributed Real-Time Systems*. IEEE Computer Society, 1995, pp. 124–132.

[25] M. González Harbour, J. Javier Gutiérrez, J. Carlos Palencia, and J. M. Drake, "Mast: Modeling and analysis suite for real time applications," in *Proceedings of 13th Euromicro Conference on Real-Time Systems*. IEEE Computer Society, 2001, pp. 125–134.

[26] T. Achterberg, "Scip: Solving constraint integer programs," *Mathematical Programming Computation*, vol. 1, no. 1, 2009.