

# Heuristic Techniques for Allocating and Scheduling Communicating Periodic Tasks in Distributed Real-Time Systems

Sébastien Faucou, Anne-Marie Déplanche and Jean-Pierre Beauvais

IRCCyN \*

1, rue de la Noë

44321 Nantes Cedex 03

{faucou | deplanche | beauvais }@irccyn.ec-nantes.fr

## Abstract

*This paper deals with the problem of pre-runtime allocating and scheduling communicating periodic tasks in a distributed real-time system. The task system is modeled with independent periodic macro-tasks. The physical architecture consists of a network of identical mono-processor sites, fully connected by a bus. Two medium access control protocols are considered : CSMA/CA and TDMA. Our objective is to find an allocation of tasks to sites and a subsequent schedule for them such that the periodicity and precedence constraints are satisfied. Besides dealing with these constraints (which is often the sole concern of many studies), the effective communication delays due to the message scheduling and the bus access control protocol are taken into account when the task schedule is being built. Two algorithms for solving this problem are presented : a clustering algorithm and a genetic algorithm.*

## 1. Introduction

Real-time computer systems are increasingly involved in various applications such as aeronautics, plant control, automotive embedded electronic, or patient monitoring. In such critical domains, the real-time computer system is required to produce correct results not only in their values but also in the times at which they are produced. To satisfy such timeliness requirements, the execution times of program units, or tasks, are constrained. This is a central idea of our work.

Furthermore, real-time computer systems often require distributed architectures to implement them. Some reasons are : a distributed architecture enables to take into account the natural geographical dispersion of the controlled environment ; in some cases, the only way to meet timing constraints is to run programs on distinct processing nodes, in true parallelism ; dependability, and fault-

tolerance in particular, is built on hardware or/and software replication.

Nowadays a significant step, and some related activities, is being emerging in the development process of real-time computer control systems ; it is known as the "validated operational (or operative) architecture design" [3, 12]. It consists, as soon as possible from the traditional design step, in mapping the entities of the software architecture onto those of the physical architecture, while verifying that the various constraints are met (even that some specified criteria are optimized).

Our research work contributes towards this problematic as far as it is concerned with techniques for pre-runtime computing allocations of communicating periodic tasks to processing nodes in a bus-like distributed system while taking into consideration the real-time constraints ; that is, being able to tell whether or not it is possible to schedule tasks under the given assignment such that all of their timing constraints can be met. It is the reason why allocation and scheduling of tasks must be dealt with together. Moreover since the end-to-end response times of a distributed application may be affected significantly by intertask communication, one must account for the effect of message scheduling and delays when task allocation decisions are taken.

Task assignment and scheduling problems are studied extensively for various software architecture and physical architecture models [5, 6, 7, 8]. However a majority of works do not take into account the behavior of the network : either they assume it is an unlimited resource with regard to task processing times (no communication delay at all), or they consider it induces only a constant overhead, or they build an hand-made network schedule [1, 18, 24, 25, 32]. On the other hand, our concern has been to take into consideration the communication delays due to an effective medium access control protocol [14, 28, 30]. For this, the well-known CSMA/CA and TDMA medium access control protocols have been considered ; they are now largely used in embedded systems, in particular automotive ones as CAN [26] and TTP [31] respectively.

---

\*IRCCyN : UMR n° 6597 CNRS / Ecole Centrale de Nantes, Université de Nantes / Ecole des Mines de Nantes.

The above problem of allocating and scheduling tasks in a distributed system is computationally complex (NP-hard), so it appears to be advantageous to use heuristics to find acceptable solutions in a reasonable time. To cope with complexity, constructive heuristics can be used like list algorithms [18, 32], or clustering algorithms [1, 25, 27]. Another group of methods dealing with it is non-guided search heuristics like tabu search, hill climbing, simulated annealing [6, 9, 11, 30], or genetic algorithms [2, 10, 15, 22]. In this paper, a clustering algorithm as well as a genetic algorithm are presented. They are both adapted from algorithms we developed previously for a very connected problem and which proved to be efficient.

The paper is organized as follows : the next section is a description of the problem (software and physical architecture, with a focus on the two medium access protocol CSMA/CA and TDMA). The third section is dedicated to the bus scheduling problem. Two allocation algorithms are presented in section four : a clustering algorithm and a genetic algorithm. The fifth section details the experimental results (including a description of the benchmark). Finally, the sixth and last section is the conclusion.

## 2. Description of the problem

### 2.1. The software architecture

In a distributed real-time system, many independent software units can co-exist. Each of them is dealing with a specific control function of the controlled process. Usually, these units are organized as a set of communicating tasks. We consider software architectures built on such a model. Thus, a software architecture is modeled as a set  $G = \{G_i, i = 1, \dots, g\}$  of  $g$  periodic, independent, hard real-time macro-tasks. The deadline for the execution of a macro-task is equal to its period. A macro-task  $G_i$  is defined as a tuple  $G_i = (T_i, E_i, \tau_i)$  and can be represented by a directed acyclic graph (cf. figure 1) :

- $T_i = \{t_{ij}, j = 1, \dots, n_i\}$  denotes the set of tasks of  $G_i$  (the nodes of the graph). Each task of this set is valued by its worst case execution time  $q_{ij}$  (to be estimated by timing analysis programs) ;
- $E_i = \{e_{ijl}, j = 1, \dots, n_i, l = 1, \dots, n_i\}$  is the set of oriented edges of the graph which symbolize the inter-task communications. The edges are valued by  $c_{ijl}$ , the size of the data to be exchanged in bytes ;
- and  $\tau_i$  is the period of the macro-task  $G_i$ .

The behavior of tasks is assumed to follow the "data-flow" model : when activated (after the completion of all its predecessors), a task starts by reading its input data ; then it computes results from them ; finally it sends those output to its successors. In such a model, no synchronization is allowed during the computation step of any task.

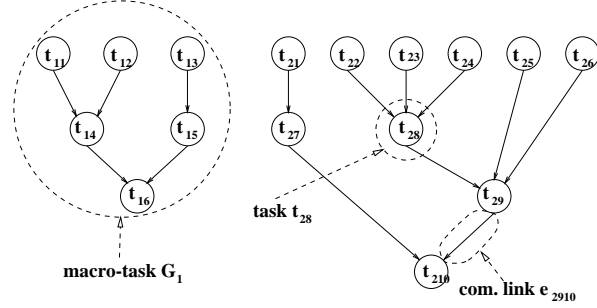


Figure 1. A software architecture

### 2.2. The physical architecture

The considered physical architecture is a local area network of mono-processor sites. All sites are assumed to be identical. They are fully interconnected by a shared broadcast bus. Each site contains the following elements :

- **a scheduler** : we consider a real-time scheduling problem with precedence constraints. We do not know any on-line algorithm able to solve efficiently this type of problem, that is why each site owns an off-line scheduler which simply follows a scheduling table (built before run-time). Because of the periodic behavior of the macro-tasks, the described scheduling can be restricted to the interval  $[0, L]$ , where  $L$  is the least common multiple of the period of the macro-tasks.
- **a communication system** : it passes the data from producers to consumers. If two communicating tasks are allocated to the same site, the communication uses shared memory (and the communication delay is negligible). If they are allocated to different sites, the shared broadcast bus is used to achieve the communication. To handle this, there is a network adapter on each site.

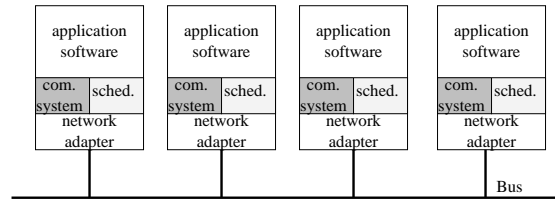


Figure 2. A physical architecture

For the moment, we have two medium access control (MAC) protocols : the CSMA/CA protocol (Carrier Sense Multiple Access / Collision Avoidance, used by the CAN network [26]) and the TDMA protocol (Time Division Multiple Access, used by TTP [31]).

#### 2.2.1 Overview of the CSMA/CA protocol

CSMA/CA is a medium access control protocol based on the CSMA strategy. It is well-suited for real-time systems

and embedded applications in particular, because it offers a deterministic collision resolution mechanism<sup>1</sup>. With this property, it is possible to compute the end-to-end communication delay of a frame, or at least an upper bound for it [29].

The first field of a CSMA/CA frame is called the arbitration field. It is the binary code of the frame's priority (in the CAN 2.0A standard ([26]), this field is 11 bits long. This allows  $2^{11}$  priorities. In CAN 2.0B, it is 29 bits long ( $2^{29}$  priorities)). This priority is used by the collision arbitration process : the frame with the lowest priority wins the exclusive access to the shared bus. It is also used as an identifier for the filtering of the incoming frames.

For a real-time system, this mechanism has to be efficient, ie. the overhead must be minimal. In a CAN network, the overhead is null (no frame are destroyed during the collision arbitration). The communication channel can be in two states : dominant (0) or recessive (1).

- it is in the dominant state if one site (at least) of the network is sending a dominant bit.
- it is in the recessive state if all the sending sites are sending a recessive bit.

A dominant bit overwrites all recessive bits : the communication channel behaves like the XOR function.

While sending the arbitration field of a frame, the network adapters are monitoring the bus : as long as the state of the bus is equal to the bit sent, it can go on ; if there is a difference, it must stop and continue with a receiving operation.

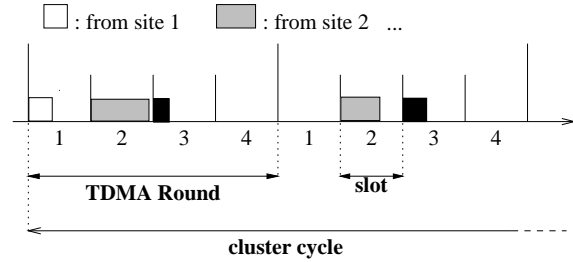
In our model, data productions are periodic. For CSMA/CA, this means that two frames used to transport two instances of the same data unit will have the same identifier (ie. the same priority). We are limiting ourselves to the case where a frame contains a single data and where a data is contained in only one frame (this imposes to limit the size of data to the size of the data field of a frame<sup>2</sup>).

### 2.2.2 Overview of the TDMA protocol

TDMA stands for Time Division Multiple Access : it is a medium access control protocol based on temporal multiplexing of data. It is the MAC protocol used by TTP [17, 31]. Like CSMA/CA, it is deterministic and well-suited for real-time systems.

As shown in figure 3, the TDMA protocol divides time into slots. During a slot, a frame can be sent. One slot is owned by one site. The owner site is the only one allowed to send data for the duration of the slot.

We consider a network with a TDMA protocol and the following features (from the TTP network) :



**Figure 3. Medium Access Control with TDMA**

- every site of the network is assigned to exactly one slot in the "TDMA Round", and it is always the same (ie. always the first of the round or always the third or ...). This is illustrated in figure 3.
- each communication system contains a dispatching table that contains the information : "which site is allowed to send which message at a particular point in time". This table is built before run-time. The table covers a period called the "cluster cycle". When a cluster cycle is over, a new one begins with the same table. For our model, the length of the cluster cycle is equal to the least common multiple of the production periods of the data units (ie. the least common multiple of the periods of the macro-tasks). The usual name of the table (in TTP) is MEDL for "Message Description List".

## 3. Bus scheduling

In a real-time system, tasks have timing requirements (they must execute before a deadline). When two tasks are communicating, the end-to-end communication delay must be bounded in order to allow the consumer to meet its timing requirements. The value of such a bound depends on both the deadline of the producer and the deadline of the consumer.

On the one hand, messages have timing requirements ; on the other hand, the communication channel is a shared resource. So, in addition to solve the task scheduling problem, we have to solve a bus scheduling problem.

In case of concurrent accesses to the bus, it is the MAC protocol which chooses the frame to be sent. To solve our bus scheduling problem, we have to control the behavior of the MAC protocol, ie. to analyze and exploit the protocol functioning. Thus, the concurrent frames will be sent in an order which fulfills the constraints.

To control the behavior of a CSMA/CA-based network, we must conveniently allocate priorities to frames. The CSMA/CA protocol has the same behavior as a non-preemptive fixed-priority scheduler. If the software architecture was appropriate, well-known task scheduling algorithm like "Rate Monotonic" (which has been proved

<sup>1</sup>CSMA/CA is not the only CSMA-based protocol with a deterministic collision resolution mechanism. One can mention CSMA-DCR or DOD/CSMA-CD [19]

<sup>2</sup>8 bytes

optimal) could be used for the priority allocation [20]. Unfortunately, we do not know any optimal algorithm for our software architecture model, so we have tried different heuristic strategies :

- the first one gives the strongest priority to the frame which transports the data with the highest **bus utilization** (BU). The BU is defined as the ratio of the frame propagation delay to the data production period.
- the second strategy gives the strongest priority to the frame which transports the data with the smallest **laxity**. The laxity of a data unit is equal to the "latest sending time" (LST) of the data minus the "earliest sending time" (EST) of the data. These times are computed from the graph of the macro-task. We need also to know a relative allocation of the tasks on the sites (we need to know which data are to be sent over the bus). The EST is climbed down from the root of the graph. It is the earliest production time of the data instance. The LST is climbed up from the leaves of the graph. If the data is sent after this date, it is sure that one of its successors will miss its deadline. For the computation of these values, we approximate the end-to-end communication delay to the physical propagation delay (we do not take into account the medium access delay).  
The EST of the  $l^{\text{th}}$  instance of message  $e_{ijk}$  is computed with formula 1. The EST of the  $l^{\text{th}}$  instance of task  $t_{ij}$  (which is the Earliest Starting Time of  $t_{ij}^l$ ) is computed with formula 2. To compute this, one need to compute the Earliest Availability Time of its incoming message (formula 3). The function  $\phi$  computes the physical propagation time of a message. The LST of the  $l^{\text{th}}$  instance of message  $e_{ijk}$  is computed with formula 5 and the LST of the  $l^{\text{th}}$  instance of task  $t_{ij}$  (which is the Latest Starting Time of  $t_{ij}^l$ ) is computed with formula 4.
- the third strategy gives the strongest priority to the frame which transport the data produced with the smallest **period**. In our case, this strategy must be combined with an other one : all the data exchanged in a macro-task have the same production period (equal to the period of the macro-task). To decide between them, we use the second strategy (laxity).

For the TDMA protocol, we can completely determine the behavior of the protocol by building the MEDL. To construct it, we need to know the production time of the data, so we must construct it while computing the scheduling of tasks on the sites (we can not do it after because we need to know the behavior of the bus to construct the sites schedules).

Most of the MEDL information are coming directly from the task scheduling. In fact, we have only one problem to solve : in case of concurrent accesses to the bus

from data produced on the same site, which ones must be sent during the next slot of that site ? To answer this question, we need to evaluate the importance of data :

- by one of the three strategies designed for the CSMA/CA protocol ;
- by a new strategy, based on the **emergency** of the data. The emergency is equal to the LST minus the real production time. This strategy can not be used with CSMA/CA because it can give a different priority to two frames transporting two instances of the same data.

$$\text{EST} (e_{ijk}^l) = \text{EST} (t_{ij}^l) + q_{ij} \quad (1)$$

$$\text{EST} (t_{ij}^l) = \max \left( (l-1)\tau_i, \max_{e_{iyj}^l} (\text{EAD} (e_{iyj}^l)) \right) \quad (2)$$

$$\text{EAD} (e_{iyj}^l) = \text{EST} (e_{iyj}^l) + \begin{cases} \phi(c_{iyj}) & \text{if } \text{alloc}(t_{iy}) \neq \text{alloc}(t_{ij}) \\ 0 & \text{else} \end{cases} \quad (3)$$

$$\text{LST} (t_{ij}^l) = \min_{e_{ijx}^l} (\text{LST} (e_{ijx}^l) - q_{ij}) \quad (4)$$

$$\text{LST} (e_{ijk}^l) = \text{LST} (t_{ik}^l) - \begin{cases} \phi(c_{ijk}) & \text{if } \text{alloc}(t_{ij}) \dots \\ 0 & \text{else} \end{cases} \quad (5)$$

## 4. Algorithms

In this section, we present the two algorithms that were developed to solve our problem. Our problem is a NP-hard search problem, so we decided to use algorithms based on heuristic techniques : a clustering algorithm and a genetic algorithm. Previous versions of these algorithms have been studied in [6, 5, 4, 21]. These versions have been modified later so as to include a realistic network management, as described in [13].

### 4.1. The clustering algorithm

This clustering algorithm has been initially designed and studied in [6, 4] and is inspired from the work described in [25]. it operates two steps :

- a clustering step ;
- an allocation/scheduling step.

### 4.1.1 The clustering step

The purpose of the clustering step is to create clusters of communicating tasks. Later, during the allocation step, all the tasks of a cluster will be allocated to the same site (as a result, the communication link between these two tasks will be withdrawn). In this way, the clustering step generates constraints on the allocation. Thus, the second step of the algorithm can perform an exhaustive exploration of the resulting small search space (which may contain no valid solution).

Here are the rules followed by the clustering step for the creation of the clusters :

- two communicating tasks may be clustered if and only if the bus utilization of their communication link is high (compared to the bus utilization of the other communication links of the architecture) :

$$\frac{c_{ijl}}{\tau_i} \geq CF \times \max_{e_{xyz} \in E_x, E_x \in G} \frac{c_{xyz}}{\tau_x} \quad (6)$$

CF is the **communication factor**.  $CF \in [0, 1]$ . It is a parameter of the clustering function. When  $CF = 0$ , all the communicating tasks may be clustered ; when  $CF = 1$ , no couple of communicating tasks satisfies condition 6.

- all cluster  $\mathcal{C}$  must meet condition 7 :

$$\sum_{s_{ij} \in \mathcal{C}} \frac{q_{ij}}{\tau_i} \leq 1 \quad (7)$$

with this second rule, the clustering function can not build clusters which need a computing power greater than the computing power offered by a single site. This rule is especially used when condition 6 allows to cluster two tasks which are already in clusters. If condition 7 is satisfied, the two clusters can be merged.

### 4.1.2 The allocation/scheduling step

This second step allocates tasks to site and tries to build a valid schedule. The allocation must verify the following constraints (produced by the clustering step) :

- all the tasks of a same cluster must be allocated to the same site ;
- two communicating tasks from two different clusters must be allocated to different sites.

To build the allocation and the scheduling, the algorithm uses a list of ready tasks (a task is ready when all its incoming messages are available and its starting time is over), a list of free sites and a variable to represent the time, as shown in algorithm above. At each decision point in time (when a site becomes free or when a message is

newly available), the algorithm tries to allocate and schedule the actually ready tasks to the actually free sites. This operation is based on a search tree where each node is an arrangement of tasks on sites. This exploration is guided by the latest start time of the tasks.

As soon as a task is scheduled, its outgoing messages are passed to the network management function which computes the availability dates of these messages for their consumers.

For the first try, the communication factor CF is set to 0. If the function fails to find a valid solution, the communication factor CF is increased by 0.1. This is repeated while  $CF \leq 1$ .

#### The Clustering Algorithm

```
// Init
to_do ← {t11, t12, ..., tmnm}
ready_tasks ← 0
free_sites ← {h1, h2, ..., hp}
time ← 0
// Clustering step
compute_clusters ()
// Allocation - Scheduling step
ready_tasks = update_ready_tasks (to_do, time)
while |to_do| ≠ 0 ∨ |free_sites| ≠ 0 do
  if time_ok (time) then
    if exists_alloc_sched (ready_tasks, free_sites) then
      alloc_sched (ready_tasks, free_sites)
      free_sites ← update_free_sites ()
      to_do ← to_do - ready_tasks
      time ← update_time ()
      ready_tasks ← update_ready_tasks (to_do, time)
    else
      exit (fail)
    end if
  else
    exit (fail)
  end if
end while
exit (success)
```

### 4.2. The genetic algorithm

A genetic algorithm (GA) is an optimization method developed by Holland [16] to mimic the adaptive mechanism of natural systems : a GA search for an optimum all over the solution space of the problem to be solved. Several points (the population) of that space are simultaneously explored. Each of these points is represented by an individual (of the population). An individual is a coded representation (a chromosome) of a solution. To find an optimum, the GA creates new individuals through evolution operators like crossover and mutation. The fitness of these new individuals is evaluated in order to select a new population (generally composed of the fittest individuals from the former population and the fittest new individuals).

The keys of a GA are :

- the coding
- the evolution operators
- the fitness function
- the population replacement strategy

We have worked with a GA designed and studied previously in [21, 22]. The pseudo-code for this algorithm is given below.

#### The Genetic Algorithm

```
// Init
generation ← 1
population ← generate_init_population ()
evaluate (population)
// Start of the optimization process
repeat
  children ← crossover (population, proba_cross)
  children ← mutation (children, proba_mut)
  evaluate (children)
  population ← population ∪ children
  population ← selection (population)
  generation ← generation +1
until (∃i ∈ population/fitness(i) = 0) ∨ (generation = max_generation)
// Did we found a solution ?
if ∃i ∈ population/fitness(i) = 0 then
  exit (success)
else
  exit (fail)
end if
```

#### 4.2.1 The coding

The coding chosen here is very close to a direct representation (a coding is a direct representation if an individual corresponds to exactly one solution of the search space). It can also be used with classical evolution operators. It is illustrated by figure 4.

Each task  $t_{ij}$  of the software architecture is represented by a complex gene which includes both its allocation and a priority for each of its instances during the schedule (ie. the least common multiple of the periods of the macro-tasks). This priority determines the execution sequence on each site (the task with the lowest priority is executed first).

On the chromosome represented in figure 4,  $\tau_2 = 2 \times \tau_1$ . In the solution described by this chromosome, task  $t_{11}$  is allocated to site 1, task  $t_{12}$  is allocated to site 3, etc. The execution sequence of site 1 starts with task  $t_{15}$ , the execution sequence of site 2 starts with task  $t_{29}$ , etc.

task :	$t_{11}$	$t_{12}$	$t_{13}$	$t_{14}$	$t_{15}$	$t_{16}$	$t_{21}$	$t_{22}$	$t_{23}$	$t_{24}$	$t_{25}$	$t_{26}$	$t_{27}$	$t_{28}$	$t_{29}$	$t_{30}$
allocation :	1	3	4	4	1	3	2	3	1	3	4	4	1	4	2	4
priority (1) :	12	5	19	3	7	4	16	1	9	20	18	2	15	11	6	13
priority (2) :	14	17	21	8	22	10										

macro-task G1
macro-task G2

Figure 4. A chromosome

#### 4.2.2 The crossover and the mutation operator

The coding allows the use of classical evolutionary operators. Thus, the crossover operator used is the one-point crossover operator, as illustrated by figure 5. Two new individuals (called children) are created from two chromosomes (called parents). This operator can create invalid individuals (the chromosome of this individual does not represent a feasible schedule). When created, these individuals are repaired. The parents are selected by the "roulette wheel" selection with linear normalization fitness [22]. This technique gives preference to the fittest individuals, without totally excluding the weakest ones.

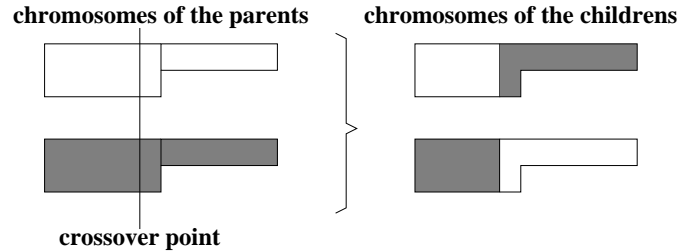


Figure 5. Crossover operator

The purpose of mutation is to operate a little change in the chromosome. The mutation operator used here randomly chooses a gene in the selected chromosome and changes its allocation for a new one (randomly chosen too). This operator can create invalid individuals and mutated individuals may need to be repaired.

#### 4.2.3 The fitness function

The optimization criteria used by our GA is the minimization of the tardiness of the execution sequence corresponding to a chromosome. With such a criteria, an individual with a null fitness is a valid solution : in the sequence corresponding to it, all the tasks complete their execution before their deadlines.

To measure this tardiness, the function builds the allocation and the schedule described in the chromosome. This step uses the network management function in the same way as the clustering algorithm : once a task is scheduled, its outgoing messages are given to that function. The function returns back the availability date of these messages.

#### 4.2.4 The population replacement strategy

The purpose of the replacement strategy is to keep the size of the population constant by eliminating individuals. Our GA uses a really simple replacement strategy : it discards the weakest individuals of the resulting population (parents together with their children).

### 5. Performance analysis

Whereas we have done extensive experimentation with the algorithms, due to space limitations, here we show only some of the salient results.

#### 5.1. The benchmark

In order to test the performance of the algorithms with various architectures, a workload generator has been constructed to randomly create software architectures. The generator takes a lot of parameters as inputs such as, for example, the number of macro-tasks, the possible numbers of tasks in a macro-task, the possible periods of a macro-task, the maximum number of tasks of the architecture, the laxity of the macro-tasks (here, the laxity is defined as the ratio between the period and the amount of processing time of a macro-task), the graph type, etc. For a given combination of all these parameters, the generator creates a software architecture accordingly. As for the reported performance assessment, the main values have been chosen in the following way :

- the number of macro-tasks varies from 2 to 5 ;
- the number of tasks in a macro-task varies from 6 to 10 ;
- the period of a macro-task is equal either to 6000  $\mu s$ , or to 7500  $\mu s$  , or to 15.000  $\mu s$ .

Each software architecture has then been combined with a physical architecture in order to produce a problem to be solved by the heuristic algorithms. A physical architecture has been characterized by its number of sites, here 3, 4, or 5 sites. Before to run the heuristic algorithms, the necessary unoverloading condition has been checked, i.e. the sum of the macro-task utilization ratios must not exceed the number of sites of the considered physical architecture. Thus it led to generate :

- 292 distinct software architectures for a 3-site physical architecture ;
- 528 distinct software architectures for a 4-site physical architecture ;
- 672 distinct software architectures for a 5-site physical architecture.

That is to say that a set of 1492 distinct problems were to be solved by each heuristic algorithm.

Lastly, different versions of these algorithms have been implemented and analyzed : one for each message scheduling strategy as explained in section 3.

#### 5.2. The results

Table 1 gives the number of problems for which a feasible task allocation and schedule has been found, according to the bus access control protocol considered. It can be seen that the bus access control protocol is a parameter that acts upon the ability for the heuristics to find feasible solutions. The purpose of this paper is to focus on this particular point but one can find another qualitative and quantitative performance trends in [6].

protocol	solved problems
CSMA/CA	1312/1492
TDMA	1226/1492

**Table 1. Absolute success ratio per MAC protocol.**

Table 2 summarizes the obtained results expressed as a relative success ratio. The relative success ratio of an algorithm is defined as the number of problems solved by this algorithm divided by the number of problems for which a solution has yet been found by any other algorithm. Indeed, as mentioned in Introduction, the decision problem of knowing if a feasible allocation and schedule exists for given software and physical architectures is NP-hard.

algo	success	algo	success
clus	93 %	clus	90 %
ga	80 %	ga	73 %

(a) CSMA/CA

(b) TDMA

**Table 2. Relative success ratio per algorithm.**

Table 2 shows that the clustering algorithm has a better performance than the genetic algorithm. However we think that those performance values are upper limits for the clustering algorithm that could not be improved whereas the genetic algorithm should have still potential if its numerous parameters (population size, crossover and mutation probabilities, number of generations, etc) are closely settled in relation with the problem. Actually, the clustering method is not convenient for some problems. As said in section 4.1, it introduces new assignment constraints which may reduce the search space to an empty set of feasible solutions. On the other hand, the same problems can be solved by the genetic algorithm which does not intrinsically limit the search space.

Besides the genetic algorithm is able to compute feasible solutions with almost great BU (84% maximum for CSMA/CA and 52% for TDMA). It suggests that by adding a coded representation of the bus schedule inside

the chromosomes, it should be possible to improve its results. Such an approach has yet been investigated in [23] to schedule communications on a TTP network (whose medium access control protocol is a TDMA one).

Moreover previous tables show that the heuristic algorithms both work best when the medium access control protocol is CSMA/CA. The results shown in Tables 3(a) and 3(b) reinforce this remark. These tables summarize the average BU for the feasible solutions, as well as the percentage of communications that have not been withdrawn due to allocation decisions. It can be seen that in case of TDMA protocol the bus is much less used than in case of CSMA/CA. It leads us to think that the experimented message scheduling strategies are not well-suited for the TDMA protocol. This drawback is bypassed by the allocation algorithms that compute (in that case) solutions which minimize the BU.

algo	solution avg. BU	$\frac{BU(\text{solution})}{BU(\text{problem})}$
clus	2.8 %	4.3%
ga	34.7 %	44.2%

(a) CSMA/CA

algo	solution avg. BU	$\frac{BU(\text{solution})}{BU(\text{problem})}$
clus	0.04 %	0.8 %
ga	13.4 %	20.2 %

(b) TDMA

**Table 3. Avg. reduction of the BU level by the allocation heuristic (per protocol)**

politic	success	politic	success
BU	99 %	BU	98 %
laxity	99 %	laxity	99 %
period	99 %	period	98 %
		emergency	98 %

(a) CSMA/CA

(b) TDMA

**Table 4. Influence of the bus scheduling strategy.**

The relative success ratios of the different message scheduling strategies are presented in tables 4(a) and 4(b). They are all excellent and their likeness is not surprising. All of them have the same goal : to give the preference to those data that are urgent. But, even if they do not base

their urgency evaluation on the same criteria, it is always the same messages that are identically favored. Furthermore, one must not forget that these message scheduling strategies have a minor influence on the feasibility of a solution. They are only considered after the task allocation and scheduling decisions have been made and they are taken into account only when some freedom is left by the bus access control protocol.

## 6. Conclusion

In this paper, we have addressed the problem of allocating and scheduling a set of periodic communicating macro-tasks to the processing nodes of a distributed real-time system. Task allocation is one of the most important issues in designing distributed real-time systems. However this problem is generally known to be NP-hard even without considering the precedence constraints resulting from inter-task communications. Thus we have solved our problem by heuristic techniques : a constructive one known as clustering algorithm, and a no-guided search one known as genetic algorithm.

Although a lot of research works have yet paid a great attention to the allocation problem, very few studies are conducted with a realistic model for the message communication protocol. This is the fundamental motivation of our work. Two medium access control protocols largely used in embedded systems have been considered : CSMA/CA (implemented by CAN) and TDMA (implemented by TTP). Moreover some message scheduling strategies have been taken into account.

The experimental analysis points out that the medium access control protocol has an influence on the allocation algorithm performance. In addition it illustrates the behavior of the heuristic algorithms : the clustering algorithm trends towards the minimization of the message-passing interactions between tasks whereas the genetic algorithm does not. Even if those experimental results show that the clustering algorithm is the best, we are convinced that the genetic algorithm can be improved. On the one hand, we have to pay much more attention to the adjustment of its various parameters. And, on the other hand, some specific communication scheduling information should be merged into the genetic representation so as to be involved in the optimization process. Our purpose is now to go into the matter more closely, being inspired for example by the approach developed in [23].

Our future work should also include research with regard to some extensions of the considered models. First, the model will be extended by allocation constraints. Such constraints are frequent in real-time control systems and express any specific resource requirement. Such resources include the CPU, sensors, input-output devices, data structures, files, or databases. Resource constraints restrict the sites to which a task can be assigned : these sites should have the resources required by the task.

In addition, fault-tolerance requirements will be added.

When fault-tolerance is achieved via replication, fault-tolerance requirements of a task is specified by the number of replicates needed for it. Hence exclusion constraints have to be taken into account since the instances of a replicated task (each instance is associated with the specifications of the original task) have to be executed at different sites.

## References

- [1] P. Altenbernd and H. Hansson. The Slack Method : A New Method for Static Allocation of Hard Real-Time Tasks. *Real-Time Systems*, 15:103–130, 1998.
- [2] L. Baccouche. *Un Mécanisme d'Ordonnement Distribué de Tâches Temps Réel*. PhD thesis, Institut National Polytechnique de Grenoble, 1995.
- [3] M. Bayart and F. Simonot-Lion. Impact de l'émergence des réseaux de terrain et de l'instrumentation intelligente dans la conception des architectures des systèmes d'automatisation de processus. projet MESR 2033 - Rapport final, 1995.
- [4] J. Beauvais and A. Déplanche. Heuristic for Scheduling Periodic Complex Real-Time Tasks in a Distributed System. In *DCCS'95, 13<sup>th</sup> IFAC Workshop*, pages 55–60, 1995.
- [5] J. Beauvais and A. Déplanche. Affectation de tâches dans un système temps réel réparti. *T.S.I.*, 17(1), Jan 1998.
- [6] J.-P. Beauvais. *Etude d'Algorithmes de Placement de Tâches Temps Réel Priodiques Complexes dans un Système Réparti*. PhD thesis, ECN, Université de Nantes, 1996.
- [7] Y. Belhamissi and M. Jégado. Scheduling in Distributed Systems : Survey and Questions. Technical Report 1478, INRIA, 1991.
- [8] A. Billionet, M. Costa, and A. Sutter. Les problèmes de placement dans les systèmes distribués. *T.S.I.*, AFCET:307–337, 1989.
- [9] M. Coli and P. Palazzari. A New Method for Optimization of Allocation and Scheduling in Real-Time Applications. In *Proc. of the 7<sup>th</sup> Euromicro Workshop on Real-Time Systems*, pages 262–269, 1995.
- [10] R. Conea, A. Ferreira, and P. Rebreyend. Scheduling Multiprocessor Tasks with Genetic Algorithms. *IEEE Trans. on Parallel and Distributed Systems*, 10(8):825–837, 1999.
- [11] M. Di Natale and J. Stankovic. Applicability of Simulated Annealing Methods to Real-Time Scheduling and Jitter Control. In *Proc. of the IEEE Real-Time Systems Symposium*, pages 216–227, 1994.
- [12] E. Durand. *Description et vérification d'architectures d'application temps réel : CLARA et les réseaux de Petri temporels*. PhD thesis, ECN, Université de Nantes, 1998.
- [13] S. Faucou. Prise en compte de la politique d'accès au réseau dans l'ordonnement de tâches temps réel. Master's thesis, IRCyN – ECN, EMN, Université de Nantes, 1999.
- [14] G. Fohler and C. Koza. Scheduling for Distributed Hard Real-Time Systems using Heuristic Search Strategies. Technical Report RR 12/90, Technische Universität Wien - Institut für technische informatik, 1990.
- [15] G. Greenwood, C. Lang, and S. Hurley. Applied Decision Technologies. pages 171–188, 1995.
- [16] J. H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, 1975.
- [17] H. Kopetz and G. Grünsteidl. TTP - A Protocol for Fault-Tolerant Real-Time Systems. *IEEE Computer*, 1995.
- [18] E. Larsson. VIA BASEMENT Real-Time System : the scheduling tool. Technical Report ProVIA - 94204, Dept. Of Computer Systems, Uppsala University, 1995.
- [19] G. LeLann and N. Rivière. Real-Time Communications over Broadcast Networks : the CSMA-DCR and the DOD/CSMA-CD Protocols. In *RTS'94*, pages 67–84, 1994.
- [20] Z. Mammeri. Réseaux et temps réel - Ordonnement de messages. In *Ecole d'été ETR'97 : Applications, Réseaux et Systèmes*, pages 62–79, 1997.
- [21] Y. Monnier. Un Algorithme Génétique pour le problème du Placement-Ordonnement de Tâches Temps Réel. Master's thesis, IRCyN – ECN, EMN, Université de Nantes, 1997.
- [22] Y. Monnier, J.-P. Beauvais, and A.-M. Déplanche. A Genetic Algorithm for Scheduling Tasks in a Real-Time Distributed System. In *24<sup>th</sup> Euromicro Conference*, volume 2, pages 708–714. IEEE, 1998.
- [23] R. Nossal and T. Galla. Solving NP-Complete Problems in Real-Time System Design by Multichromosome Genetic Algorithms. Technical Report RR 1-97, Institut für Technische Informatik - Technische Universität Wien, 1997.
- [24] D. Peng, K. Shin, and T. Abdelzاهر. Assignment and Scheduling Communicating Periodic Tasks in Distributed Real-Time Systems. *IEEE Trans. on Software Engineering*, 23(12), 1997.
- [25] K. Ramamritham. Allocation and Scheduling of Complex Periodic Tasks. In *10<sup>th</sup> International Conference on Distributed Computing Systems*, pages 108–115, 1990.
- [26] Rober Bosch GmbH. *CAN Specification - Version 2.0*, 1991.
- [27] V. Sarkar. *Partitioning and Scheduling Parallel Programs for Execution on Multiprocessors*. The MIT Press, 1989.
- [28] K. Tindell. Holistic Schedulability Analysis for Distributed Hard Real-Time Systems. Technical Report YCS 197, Real-Time Systems Research Group - Dept. Of Computer Science - University of York, 1993.
- [29] K. W. Tindell. Analysis of Hard Real-Time Communications. *Real-Time Systems*, 9(2):147–171, 1995.
- [30] K. W. Tindell, A. Burns, and A. Wellings. Allocating Hard Real-Time Tasks : An NP-Hard Problem made easy. *Real-Time Systems*, 4(2):145–165, 1992.
- [31] TTTech Computertechnik GmbH. *TTP/C Protocol - Specification version 0.5*, 1999.
- [32] J. Verhoosel, E. Luit, and D. Hammer. A Static Scheduling Algorithm for Distributed Hard Real-Time Systems. *Journal of Real-Time Systems*, 3(3):227–246, 1991.