Schedulability Analysis and Optimization for the Synthesis of Multi-Cluster Distributed Embedded Systems

Paul Pop, Petru Eles, Zebo Peng
Computer and Information Science Dept., Linköping University, Sweden
{paupo, petel, zebpe}@ida.liu.se

Abstract[1]

We present an approach to schedulability analysis for the synthesis of multi-cluster distributed embedded systems consisting of time-triggered and event-triggered clusters, interconnected via gateways. We have also proposed a buffer size and worst case queuing delay analysis for the gateways, responsible for routing inter-cluster traffic. Optimization heuristics for the priority assignment and synthesis of bus access parameters aimed at producing a schedulable system with minimal buffer needs have been proposed. Extensive experiments and a real-life example show the efficiency of our approaches.

1. Introduction

There are two basic approaches for handling tasks in real-time applications [8]. In the event-triggered approach (ET), activities are initiated whenever a particular event is noted. In the time-triggered (TT) approach, activities are initiated at predetermined points in time. There has been a long debate in the real-time and embedded systems communities concerning the advantages of each approach [2, 8, 17]. Several aspects have been considered in favour of one or the other approach, such as flexibility, predictability, jitter control, processor utilization, testability, etc.

The same duality is reflected at the level of the communication infrastructure, where communication activities can be triggered either dynamically, in response to an event, as with the controller area network (CAN) bus [4], or statically, at predetermined moments in time, as in the case of time-division multiple access (TDMA) protocols and, in particular, the time-triggered protocol (TTP) [8].

Process scheduling and schedulability analysis have been intensively studied for the past decades [1, 3]. A few approaches have been proposed for the schedulability analysis of distributed real-time systems, taking into consideration both process and communication scheduling. In [15, 16] Tindell provided a framework for the analysis of ET process sets interconnected through an infrastructure based on either the CAN protocol or a generic TDMA protocol. In [5] and [12] we have developed an analysis allowing for either TT or ET process sets communicating over the TTP.

An interesting comparison of the ET and TT approaches, from a more industrial, in particular automotive, perspective, can be found in [9]. The conclusion there is that one has to choose the right approach depending on the particularities of the processes. This means not only that there is no single "best" approach to be used, but also that inside a certain application the two approaches can be used together, some tasks being TT and others ET. The fact that such an approach is suitable for automotive applications is demonstrated by the following two trends which are currently considered to be of foremost importance not only for the automotive industry, but also for other categories of industrial applications:

1. The development of bus protocols which support both static and dynamic communication [6]. This allows for ET and TT processes to share the same processor as well as dynamic and static communications to share the same bus. In [11] we have addressed the problem of timing analysis for such systems.
2. Complex systems are designed as interconnected clusters of processors. Each such cluster can be either TT or ET. In a *time-triggered cluster* (TTC), processes and messages are scheduled according to a static cyclic policy, with the bus implementing the TTP. On an *event-triggered cluster* (ETC), the processes are scheduled according to a priority based preemptive approach, while messages are transmitted using the priority-based CAN protocol. Depending on their particular nature, certain parts of an application can be mapped on processors belonging to an ETC or a TTC. The critical element of such an architecture is the *gateway*, which is a node connected to both types of clusters, and is responsible for the inter-cluster routing of hard real-time traffic.

In this paper we propose an approach to schedulability analysis for the synthesis of multi-cluster distributed embedded systems, including also buffer need analysis and worst case queuing delays of inter-cluster traffic. We have also developed optimization heuristics for the synthesis of bus access parameters as well as process and message priorities aimed at producing a schedulable system such that buffer sizes are minimized.

Efficient implementation of new, highly sophisticated automotive applications, entails the use of TT process sets together with ET ones implemented on top of complex distributed architectures. In this context, this paper is the first one to address the analysis and optimization of heterogeneous TT and ET systems implemented on multi-cluster embedded networks.

The paper is organized in 7 sections. The next section presents the application model as well as the hardware and software architecture of our systems. Section 3 introduces more precisely the problems that we are addressing in this paper. Section 4 presents our proposed schedulability analysis for multi-cluster systems, and section 5 uses this analysis to drive the optimization heuristics used for system synthesis.

2. Application Model and System Architecture

2.1 Application Model

We model an application Γ as a set of process graphs $G_i \in \Gamma$ (see Figure 1). Nodes in the graph represent processes and arcs represent dependency between the connected processes. The communication time between processes mapped on the same processor is considered to be part of the process worst-case execution time and is not modeled explicitly. Communication between processes mapped to different processors is preformed by message passing over the buses and, if needed, through the gateway. Such message passing is modeled as a communication process inserted on the arc connecting the sender and the receiver process (the black dots in Figure 1).

Each process P_i is mapped on a processor $processor_{P_i}$ (mapping represented by hashing in Figure 1), and has a worst case execution time C_i on that processor (depicted to the left of each node). For each message we know its size (in bytes, indicated to its left), and its period, which is identical with that of the sender process. Processes and messages activated based on events also have a uniquely assigned priority, $priority_{P_i}$ for processes and $priority_{m_i}$ for messages.

All processes and messages belonging to a process graph G_i have the same period $T_i = T_{G_i}$ which is the period of the process graph. A deadline $D_{G_i} \leq T_{G_i}$ is imposed on each process graph G_i. Deadlines can also be placed locally on processes. If communicating processes are of different periods, they are combined into a hyper-graph capturing all process activations for the hyper-period (LCM of all periods).

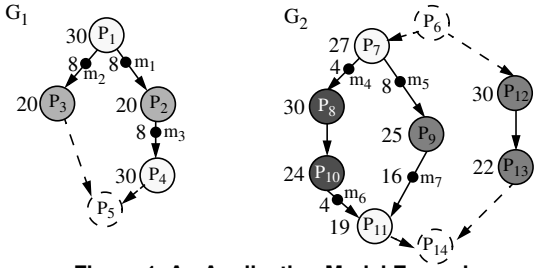


Figure 1. An Application Model Example

1. The authors are grateful to the industrial partners at Volvo Technological Development in Gothenburg, for their close involvement and precious feedback during this work.

2.2 Hardware Architecture

We consider architectures consisting of several clusters, interconnected by gateways (Figure 2 depicts a two-cluster example). A *cluster* is composed of nodes which share a broadcast communication channel. Every *node* consists, among others, of a communication controller, and a CPU. The gateways, connected to both types of clusters, have two communication controllers, for TTP and CAN. The communication controllers implement the protocol services, and run independently of the node's CPU. Communication with the CPU is performed through a *message base interface* (MBI) which is usually implemented as a dual ported RAM (Figure 3).

Communication between the nodes on a TTC is based on the TTP [8]. The bus access scheme is TDMA, where each node N_i, including the gateway node, can transmit only during a predetermined time interval, the so called TDMA slot S_i. In such a slot, a node can send several messages packaged in a frame. A sequence of slots corresponding to all the nodes in the TTC is called a TDMA round. A node can have only one slot in a TDMA round. Several TDMA rounds can be combined together in a cycle that is repeated periodically. The TDMA access scheme is imposed by a message descriptor list (MEDL) that is located in every TTP controller. The MEDL serves as a schedule table for the TTP controller which has to know when to send/receive a frame to/from the communication channel.

On an ETC the CAN [4] protocol is used for communication. The CAN bus is a priority bus that employs a collision avoidance mechanism, whereby the node that transmits the message with the highest priority wins the contention. Message priorities are unique and are encoded in the frame identifiers, which are the first bits to be transmitted on the bus.

2.3 Software Architecture

A real-time kernel is responsible for activation of processes and transmission of messages on each node. On a TTC, the processes are activated based on the local schedule tables, and messages are transmitted according to the MEDL. On an ETC, we have a scheduler that decides on activation of ready processes and transmission of messages, based on their priorities.

In Figure 3 we illustrate our message passing mechanism. Here we concentrate on the communication between processes located on different clusters. For message passing details within a TTC the reader is directed to [13], while the infrastructure needed for communications on an ETC has been detailed in [15].

Let us consider the example in Figure 3, where we have the process graph G_1 from Figure 1 mapped on the two clusters. Processes P_1 and P_4 are mapped on node N_1 of the TTC, while P_2 and P_3 are mapped on node N_2 of the ETC. Process P_1 sends messages m_1 and m_2 to processes P_2 and P_3, respectively, while P_2 sends message m_3 to P_4.

The transmission of messages from the TTC to the ETC takes place in the following way (see Figure 3). P_1, which is statically scheduled, is activated according to the schedule table, and when it finishes it calls the send kernel function in order to send m_1 and m_2, indicated in the figure by number (1). Messages m_1 and m_2 have to be sent from node N_1 to node N_2. At a certain time, known from the schedule table, the kernel transfers m_1 and m_2 to the TTP controller by packaging them into a frame in the MBI. Later on, the TTP controller knows from its MEDL when it has to take the frame from the MBI, in order to broadcast it on the bus. In our example, the timing information in the schedule table of the kernel and the MEDL is determined in such a way that the broadcasting of the frame is done in the slot S_1 of round 2 (2). The TTP controller of the gateway node N_G knows from its MEDL that it has to read a frame from slot S_1 of round 2 and to transfer it into its MBI (3). Invoked periodically, having the highest priority on node N_G,

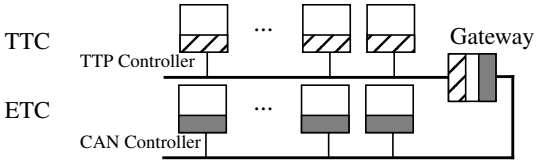


Figure 2. A System Architecture Example

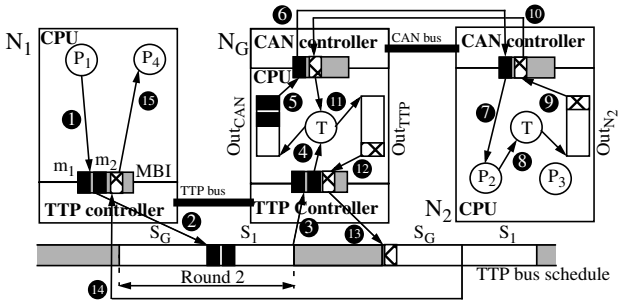


Figure 3. A Message Passing Example

and with a period which guarantees that no messages are lost, the gateway process T copies messages m_1 and m_2 from the MBI to the TTP-to-CAN priority-ordered message queue Out_{CAN} (4). The highest priority message in the queue, in our case m_1, will tentatively be broadcast on the CAN bus (5). Whenever message m_1 will be the highest priority message on the CAN bus, it will successfully be broadcast and will be received by the interested nodes, in our case node N_2 (6). The CAN communication controller of node N_2 receiving m_1 will copy it in the transfer buffer between the controller and the CPU, and raise an interrupt which will activate a delivery process, responsible to activate the corresponding receiving process, in our case P_2, and hand over message m_1 that finally arrives at the destination (7).

Message m_3 (depicted in Figure 3 as a hashed rectangle) sent by process P_2 from the ETC will be transmitted to process P_4 on the TTC. The transmission starts when P_2 calls its send function and enqueues m_3 in the priority-ordered Out_{N_2} queue (8). When m_3 has the highest priority on the bus, it will be removed from the queue (9) and broadcast on the CAN bus (10), arriving at the gateway's CAN controller where it raises an interrupt. Based on this interrupt, the gateway transfer process T is activated, and m_3 is placed in the Out_{TTP} FIFO queue (11). The gateway node N_G is only able to broadcast on the TTC in the slot S_G of the TDMA rounds circulating on the TTP bus. According to the MEDL of the gateway, a set of messages not exceeding $size_{S_G}$ of the slot S_G will be removed from the front of the Out_{TTP} queue in every round, and packed in the S_G slot (12). Once the frame is broadcast (13) it will arrive at node N_1 (14), where all the messages in the frame will be copied in the input buffers of the destination processes (15). Process P_4 is activated according to the schedule table, which has to be constructed such that it accounts for the worst-case communication delay of message m_3, bounded by the analysis in section 4, and thus when P_4 starts executing it will find m_3 in its input buffer.

As part of our timing analysis and synthesis approach, we generate all the local schedule tables and MEDLs on the TTC, the message and process priorities for the activities on the ETC, as well as buffer sizes and bus configurations such that the global system is schedulable.

3. Problem Formulation

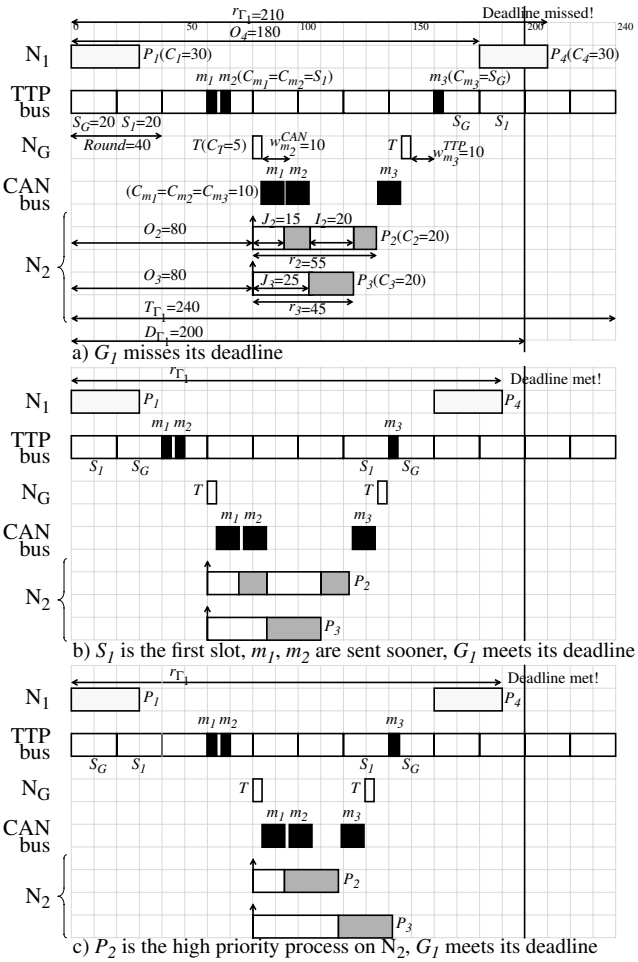
As input to our problem we have an application Γ given as a set of process graphs mapped on an architecture consisting of a TTC and an ETC interconnected through a gateway.

We are interested first to find a system configuration denoted by a 3-tuple $\psi=<\phi, \beta, \pi>$ such that the application Γ is schedulable. Determining a system configuration ψ means deciding on:

- The set ϕ of the offsets corresponding to each process and message in the system (see section 4). The offsets of processes and messages on the TTC practically represent the local schedule tables and MEDLs.
- The sequence and size of the slots in a TDMA round on the TTC (β).
- The priorities of the processes and messages on the ETC (π).

Once a configuration leading to a schedulable application is found, we are interested to find a system configuration that minimizes the total queue sizes needed to run a schedulable application. The approach presented in this paper can be easily extended to cluster configurations where there are several ETCs and TTCs interconnected by gateways.

Let us consider the example in Figure 4, where we have the process graph G_1 from Figure 1 mapped on the two-cluster system as indicated in Figure 3. In the system configuration of Figure 4a we consider that,



a) G_1 misses its deadline

b) S_1 is the first slot, m_1, m_2 are sent sooner, G_1 meets its deadline

c) P_2 is the high priority process on N_2, G_1 meets its deadline

Figure 4. Scheduling Examples

on the TTP bus, the gateway transmits in the first slot (S_G) of the TDMA round, while node N_1 transmits in the second slot (S_1). The priorities inside the ETC have been set such that $priority_{m1} > priority_{m2}$ and $priority_{P3} > priority_{P2}$. In such a setting, G_1 will miss its deadline, which was set at 200 ms. However, changing the system configuration as in Figure 4b, so that slot S_1 of N_1 comes first, we are able to send m_1 and m_2 sooner, and thus reduce the response time and meet the deadline. The response times and resource usage do not, of course, depend only on the TDMA configuration. In Figure 4c, for example, we have modified the priorities of P_2 and P_3 so that P_2 is the higher priority process. In such a situation, P_2 is not interrupted when the delivery of message m_2 was supposed to activate P_3 and, thus, eliminating the interference, we are able to meet the deadline, even with the TTP bus configuration of Figure 4a.

4. Multi-Cluster Scheduling

In this section we propose an analysis for hard real-time applications mapped on multi-cluster systems. The aim of such an analysis is to find out if a system is schedulable, i.e. all the timing constraints are met. In addition to this, we are also interested in bounding the queue sizes.

On the TTC an application is schedulable if it is possible to build a schedule table such that the timing requirements are satisfied. On the ETC, the answer wether or not a system is schedulable is given by a *schedulability analysis*.

In this paper, for the ETC we use a *response time analysis*, where the schedulability test consists of the comparison between the worst-case response time r_i of a process P_i and its deadline D_i. Response time analysis of data dependent processes with static priority preemptive scheduling has been proposed in [10, 14, 18], and has been also extended to consider the CAN protocol [15]. The authors use the concept of *offset* in order to handle data dependencies. Thus, each process P_i is characterized by an offset O_i, measured from the start of the process graph, that indicates the earliest possible start time of P_i. For example, in Figure 4a, O_2=80, as process P_2 cannot start before receiving m_1 which is available at the end of slot S_1 in round 2. The same is true for messages, their offset indicating the earliest possible transmission time.

```
MultiClusterScheduling(Γ, β, π)
   -- assign initial values to offsets
   for each Oi ∈ ϕ do Oi =initial value end for
   -- iteratively improve the offsets and response times
   repeat
      -- determine the response times based on the current values for the offsets
      ρ=ResponseTimeAnalysis(Γ, ϕ, π)
      -- determine the offsets based on the current values for the response times
      ϕ=StaticScheduling(Γ, ρ, β)
   until ϕ not changed
   return ϕ, ρ
end MultiClusterScheduling
```

Figure 5. The MultiClusterScheduling Algorithm

Determining the schedulability of an application mapped on a multi-cluster system cannot be addressed separately for each type of cluster, since the inter-cluster communication creates a circular dependency: the static schedules determined for the TTC influence through the offsets the response times of the processes on the ETC, which on their turn influence the schedule table construction on the TTC. In Figure 4a placing m_1 and m_2 in the same slot leads to equal offsets for P_2 and P_3. Because of this, P_3 will interfere with P_2 (which would not be the case if m_2 sent to P_3 would be scheduled in round 4) and thus the placement of P_4 in the schedule table has to be accordingly delayed to guarantee the arrival of m_3.

In our response time analysis we consider the influence between the two clusters by making the following observations:

- The start time of process P_i in a schedule table on the TTC is its offset O_i.
- The worst-case response time r_i of a TT process is its worst case execution time, i.e. r_i=C_i (TT processes are not preemptable).
- The response times of the messages exchanged between two clusters have to be calculated according to the schedulability analysis described in section 4.1.
- The offsets have to be set by a scheduling algorithm such that the precedence relationships are preserved. This means that, if process P_B depends on process P_A, the following condition must hold: $O_B \geq O_A+r_A$. Note that for the processes on a TTC receiving messages from the ETC this translates to setting the start times of the processes such that a process is not activated before the worst-case arrival time of the message from the ETC. In general, offsets on the TTC are set such that all the necessary messages are present at the process invocation.

The MultiClusterScheduling algorithm in Figure 5 receives as input the application Γ, the TTC bus configuration β and the ET process and message priorities π, and produces the offsets ϕ and response times ρ. The algorithm starts by assigning to all offsets an initial value obtained by a static scheduling algorithm applied on the TTC without considering the influence from the ETC. The response times of all processes and messages in the ETC are then calculated according to the analysis in section 4.1 by using the ResponseTimeAnalysis function. Based on the response times, offsets of the TT processes can be defined such that all messages received from the ETC cluster are present at process invocation. Considering these offsets as constraints, a static scheduling algorithm can derive the schedule tables and MEDLs of the TTC cluster. For this purpose we use a list scheduling based approach presented in [5]. Once new values have been determined for the offsets, they are fed back to the response time calculation function, thus obtaining new, *tighter* (i.e., smaller, less pessimistic) values for the worst-case response times. The algorithm stops when the response times cannot be further tightened and, consequently, the offsets remain unchanged. Termination is guaranteed if processor and bus loads are smaller than 100% (see section 4.2) and deadlines are smaller than the periods.

4.1 Schedulability and Resource Analysis

The analysis in this section is used in the ResponseTimeAnalysis function in order to determine the response times for processes and messages on the ETC. It receives as input the application Γ, the offsets ϕ and the priorities π, and it produces the set ρ of worst case response times.

We have extended the framework provided by [14, 15] for an ETC. Thus, the response time of a process P_i on the ETC is r_i=J_i+w_i+C_i, where J_i is the *jitter* of process P_i (the worst case delay between the

activation of the process and the start of its execution), and C_i is its worst case execution time. The *interference* w_i from other processes running on the same processor is given by:

$$w_i = B_i + \sum_{\forall j \in hp(P_i)} \left\lceil \frac{w_i + J_j - O_{ij}}{T_j} \right\rceil C_j .$$

In the previous equation, the blocking factor B_i represents interference from lower priority processes that are in their critical section and cannot be interrupted. The second term captures the interference from higher priority processes $P_j \in hp(P_i)$, where O_{ij} is a positive value representing the relative offset of process P_j to P_i.

The same analysis can be applied for messages on the CAN bus: $r_m=J_m+w_m+C_m$, where J_m is the jitter of message m which in the worst case is equal to the response time $r_{S(m)}$ of the sender process $P_{S(m)}$, w_m is the *worst-case queuing delay* experienced by m at the communication controller, and C_m is the worst-case time it takes for a message m to reach the destination controller. On CAN, C_m depends on the frame configuration and message size s_m, while on TTP it is equal to the slot size where m is transmitted.

The response time analysis for processes and messages are combined by realizing that the jitter of a destination process depends on the communication delay between sending and receiving a message. Thus, for a process $P_{D(m)}$ that receives a message m from a sender process $P_{S(m)}$, the release jitter is $J_{D(m)}=r_m$.

The worst-case queueing delay for a message is calculated differently depending on the type of message passing employed:

1. From an ETC node to another ETC node (in which case $w_m^{N_i}$ represents the worst-case time a message m has to spend in the Out_{N_i} queue on ETC node N_i),
2. From a TTC node to an ETC node (w_m^{CAN} is the worst-case time a message m has to spend in the Out_{CAN} queue).
3. From an ETC node to a TTC node (where w_m^{TTP} captures the time m has to spend in the Out_{TTP} queue).

The messages sent from a TTC node to another TTC node are taken into account when determining the offsets (StaticScheduling, Figure 5), and thus are not involved directly in the ETC analysis.

The next sections show how the worst queueing delays and the bounds on the queue sizes are calculated for each of the previous three cases.

4.1.1 From ETC to ETC and from TTC to ETC

The analyses for $w_m^{N_i}$ and w_m^{CAN} are similar. Once m is the highest priority message in the Out_{CAN} queue, it will be sent by the gateway's CAN controller as a regular CAN message, therefore the same equation for w_m can be used:

$$w_m = B_m + \sum_{\forall j \in hp(m)} \left\lceil \frac{w_m + J_j - O_{mj}}{T_j} \right\rceil C_j .$$

The intuition is that m has to wait, in the worst case, first for the largest lower priority message that is just being transmitted (B_m) as well as for the higher priority $j \in hp(m)$ messages that have to be transmitted ahead of m (the second term). In the worst case, the time it takes for the largest lower priority message $k \in lp(m)$ to be transmitted to its destination is:

$$B_m = \max_{\forall k \in lp(m)} (C_k) .$$

Note that in our case, $lp(m)$ and $hp(m)$ also include messages produced by the gateway node, transferred from the TTC to the ETC.

We are also interested to bound the size s_m^{CAN} of the Out_{CAN} and $s_m^{N_i}$ of the Out_{N_i} queue. In the worst case, message m, and all the messages with higher priority than m will be in the queue, awaiting transmission. Summing up their sizes, and finding out what is the most critical instant we get the worst-case queue size:

$$s_{Out} = \max_{\forall m} \left(s_m + \sum_{\forall j \in hp(m)} \left\lceil \frac{w_m + J_j - O_{mj}}{T_j} \right\rceil s_j \right)$$

where s_m and s_j are the sizes of message m and j, respectively.

4.1.2 From ETC to TTC

The time a message m has to spend in the Out_{TTP} queue in the worst case depends on the total size of messages queued ahead of m (Out_{TTP} is a FIFO queue), the size S_G of the gateway slot responsible for carrying the CAN messages on the TTP bus, and the frequency T_{TDMA} with which this slot S_G is circulating on the bus:

$$r_2 = J_2 + w_2 + C_2, w_2 = B_2 + \left\lceil \frac{w_2 + J_3 - O_{2,3}}{T} \right\rceil C_3$$
$$r_3 = J_3 + w_3 + C_3, w_3 = B_3$$
$$r_{m_1} = J_{m_1} + w_{m_1}^{CAN} + C_{m_1}, w_{m_1}^{CAN} = B_{m_1}$$
$$r_{m_2} = J_{m_2} + w_{m_2}^{CAN} + C_{m_2}, w_{m_2}^{CAN} = B_{m_2} + \left\lceil \frac{w_{m_2}^{CAN} + J_{m_1} - O_{m_2,m_1}}{T} \right\rceil C_3$$
$$r_{m_3} = J_{m_3} + w_{m_3}^{N_2} + C_{m_3}$$
$$r_{m_3'} = J_{m_3'} + w_{m_3'}^{TTP} + C_{m_3'}, w_{m_3'}^{TTP} = B_{m_3'} + \left\lceil \frac{s_{m_3}}{T} \right\rceil T_{TDMA}, B_{m_3'} = T_{TDMA} - O_{m_3} \bmod T_{TDMA} + O_{S_G}$$

Figure 6. Response Time Analysis Example

$$w_m^{TTP} = B_m + \left\lceil \frac{S_m + I_m}{S_G} \right\rceil T_{TDMA},$$

where I_m is the total size of the messages queued ahead of m. Those messages $j \in hp(m)$ are ahead of m, which have been sent from the ETC to the TTC, and have higher priority than m:

$$I_m = \sum_{\forall j \in hp(m)} \left\lceil \frac{w_m^{TTP} + J_m - O_{mj}}{T_j} \right\rceil s_j$$

where the message jitter J_m is in the worst case the response time of the sender process, $J_m=r_{S(m)}$.

The blocking factor B_m is the time interval in which m cannot be transmitted because the slot S_G of the TDMA round has not arrived yet, and is determined as T_{TDMA}-O_m *mod* T_{TDMA}+O_{S_G}, where O_{S_G} is the offset of the gateway slot in a TDMA round.

Determining the size of the queue needed to accommodate the worst case burst of messages sent from the CAN cluster is done by finding out the worst instant of the following sum:

$$s_{Out}^{TTP} = \max_{\forall m} (S_m + I_m) .$$

4.2 Response Time Analysis Example

Figure 6 presents the equations for our system in Figure 4a. The jitter of P_2 depends on the response time of the gateway transfer process T and the response time of message m_1, $J_2=r_{m1}$. Similarly, $J_3=r_{m2}$. We have considered that $J_{m1}=J_{m2}=r_T$. The response time r_{m_3} denotes the response time of m_3 sent from process P_2 to the gateway process T, while $r_{m_3'}$ is the response time of the same message m_3 sent now from T to P_4.

The equations are recurrent, and they will converge if the processor and bus utilization are under 100% [16]. Considering a TDMA round of 40 ms, with two slots each of 20 ms as in Figure 4a, r_T=5 ms, 10 ms for the transmission times on CAN for m_1 and m_2, and using the offsets in the figure, the equations will converge to the values indicated in Figure 4a (all values are in milliseconds). Thus, the response time of graph G_1 will be $r_{G_1}=O_4+r_4=210$, which is greater than $D_{G_1}=200$, thus the system is not schedulable.

5. Scheduling and Optimization Strategy

Once we have a technique to determine if a system is schedulable, we can concentrate on optimizing the total queue sizes. Our problem is to synthesize a system configuration ψ such that the application is schedulable, i.e. the condition[1]

$$r_{G_j} \le D_{G_j}, \forall G_j \in \Gamma_i,$$

holds, and the total queue size s_{total} is minimized[2]:

$$s_{total} = s_{Out}^{CAN} + s_{Out}^{TTP} + \sum_{\forall (N_i \in ETC)} s_{Out}^{N_i} .$$

Such an optimization problem is NP complete, thus obtaining the optimal solution is not feasible. We propose a resource optimization strategy based on a hill-climb heuristic that uses an intelligent set of initial solutions in order to efficiently explore the design space.

5.1 Scheduling and Buffer Optimization Heuristic

Our optimization heuristic is outlined in Figure 7. The basic idea of our OptimizeResources heuristic is to find, as a first step, a solution with the smallest possible response times, without considering the buffer sizes, in the hope of finding a schedulable system. This is achieved through the OptimizeSchedule function. Then, a hill-climb-

1. The worst-case response time a process graph G_i is calculated based on its sink node as $r_{G_i} = O_{sink}+r_{sink}$. If local deadlines are imposed, they will also have to be tested in the schedulability condition.
2. On the TTC, the synchronization between processes and the TDMA bus configuration is solved through the proper synthesis of schedule tables, thus no output queues are needed. Input buffers on both TTC and ETC nodes are local to processes. There is one buffer per input message and each buffer can store one message instance (see explanation to Figure 3).

ing heuristic iteratively performs moves intended to minimize the total buffer size while keeping the resulted system schedulable.

The OptimizeSchedule function outlined in Figure 8 is a greedy approach which determines an ordering of the slots and their lengths, as well as priorities of messages and processes in the ETC, such that the degree of schedulability of the application is maximized. The *degree of schedulability* [12] is calculated as:

$$\delta_\Gamma = \begin{cases} f_1 = \sum_{i=1}^{n} max(0, R_{G_i} - D_{G_i}), \text{ if } f_1 > 0 \\ f_2 = \sum_{i=1}^{n} (R_{G_i} - D_{G_i}), \text{ if } f_1 = 0 \end{cases}$$

where n is the number of process graphs in the application. If the application is not schedulable, the term f_1 will be positive, and in this case the cost function is equal to f_1. However, if the process set is schedulable, $f_1 = 0$ and we use f_2 as a cost function, as it is able to differentiate between two alternatives, both leading to a schedulable process set. For a given set of optimization parameters leading to a schedulable process set, a smaller f_2 means that we have improved the response times of the processes.

As an initial TTC bus configuration β, OptimizeSchedule assigns in order nodes to the slots and fixes the slot length to the minimal allowed value, which is equal to the length of the largest message generated by a process assigned to N_i, S_i=<N_i, $size_{smallest}$>. Then, the algorithm starts with the first slot and tries to find the node which, when transmitting in this slot, will maximize the degree of schedulability δ_Γ.

Simultaneously with searching for the right node to be assigned to the slot, the algorithm looks for the optimal slot length. Once a node was selected for the first slot and a slot length fixed (S_i=S_{best}), the algorithm continues with the next slots, trying to assign nodes (and to fix slot lengths) from those nodes which have not yet been assigned. When calculating the length of a certain slot we consider the feedback from the MultiClusterScheduling algorithm which recommends slot sizes to be tried out. Before starting the actual optimization process for the bus access scheme, a scheduling of the initial solution is performed which generates the recommended slot lengths. We refer the reader to [5] for details concerning the generation of the recommended slot lengths.

In the OptimizeSchedule function the degree of schedulability δ_Γ is calculated based on the response times produced by the MultiClusterScheduling algorithm. For the priorities used in the response time calculation we use the "heuristic optimized priority assignment" (HOPA) approach in [7], where priorities for processes and messages in a distributed real-time system are determined, using knowledge of the factors that influence the timing behaviour, such that the degree of schedulability is improved.

The OptimizeSchedule function also records the best solutions in terms of δ_Γ and s_{total} in the seed_solutions list in order to be used as the starting point for the second step of our OptimizeResources heuristic.

Once a schedulable system is obtained, our goal is to minimize the buffer space. Our design space exploration in the second step of OptimizeResources is based on successive design transformations (generating the neighbors of a solution) called *moves*. For our heuristics, we consider the following types of moves:

- moving a process or a message belonging to the TTC inside its [*ASAP, ALAP*] interval calculated based on the current values for the offsets and response times;
- swapping the priorities of two messages transmitted on the ETC, or of two processes mapped on the ETC;
- increasing or decreasing the size of a TDMA slot with a certain value;
- swapping two slots inside a TDMA round.

```
OptimizeResources(Γ)
  -- Step 1: try to find a schedulable system
  seed_solutions=OptimizeSchedule(Γ)
  -- if no schedulable configuration has been found, modify mapping and/or architecture
  if Γ is not schedulable for ψbest then modify mapping; go to step 1; end if
  -- Step 2: try to reduce the resource need, minimize stotal
  for each ψ in seed_solutions do
    repeat
      -- find moves with highest potential to minimize stotal
      move_set=GenerateNeighbors(ψ)
      -- select move which minimizes stotal
      -- and does not result in an un-schedulable system
      move = SelectMove(move_set); Perform(move)
    until stotal has not changed or limit reached
  end for
  return system configuration ψ, queue sizes
end OptimizeResources
```

Figure 7. The OptimizeResources Algorithm

```
OptimizeSchedule(Γ)
  -- given an application Γ produces the configuration ψ=<φ, β, π> leading to the smallest δΓ
  -- start by determining an initial TTC bus configuration β
  for each slot Si ∈ β do Si=<Ni, sizesmallest> end for
  -- find the best allocation of slots, the TDMA slot sequence
  for each slot Si ∈ β do
    for each node Nj ∈ TTC do
      Si=<Nj, sizeSi>; Sj=<Ni, sizeSj> -- allocate Nj tentatively to Si, Ni gets slot Sj
      -- determine best size for slot Si
      for each slot size ∈ recomended_lengths(Si) do
        π=HOPA -- calculate the priorities according to the HOPA heuristic
        -- determine the offsets φ, thus obtaining a complete system configuration ψ
        Si=<Nj, size>; φ=MultiClusterScheduling(Γ, β, π); ψcurrent=<φ, β, π>
        -- remember the best configuration so far, add it to the seed configurations
        if δΓ(ψcurrent) is best so far then
          ψbest = ψcurrent; Sbest=Si;
          add ψbest to seed_solutions
        end if
        determine stotal for ψcurrent
        if stotal is best so far and Γ is schedulable
        then add ψcurrent to seed_solutions end if
      end for
    end for
    -- make binding permanent, use the Sbest corresponding to ψbest
    if a Sbest exists then Si=Sbest end if
  end for
  return ψbest, δΓ(ψbest), seed_solutions
end OptimizeSchedule
```

Figure 8. The OptimizeSchedule Algorithm

The second step of the OptimizeResources heuristic start from the seed solutions produced in the previous step, and iteratively preforms moves in order to reduce the total buffer size, s_{total}. The heuristic tries to improve on the total queue sizes, without producing un-schedulable systems. The neighbors of the current solution are generated in the GenerateNeighbours functions, and the move with the smallest s_{total} is selected using the SelectMove function. Finally, the move is performed, and the loop reiterates. The iterative process ends when there is no improvement achieved on s_{total}, or a limit imposed on the number of iterations has been reached.

In order to improve the chances to find good values for s_{total}, the algorithm has to be executed several times, starting with a different initial solution. The intelligence of our OptimizeResources heuristic lies in the selection of the initial solutions, recorded in the seed_solutions list. The list is generated by the OptimizeSchedule function which records the best solutions in terms of δ_Γ and s_{total}. Seeding the hill climbing heuristic with several solutions of small s_{total} will guarantee that the local optima are quickly found. However, during our experiments, we have observed that another good set of seed solutions are those that have high degree of schedulability δ_Γ. Starting from a highly schedulable system will permit more iterations until the system degrades to an un-schedulable configuration, thus the exploration of the design space is more efficient.

6. Experimental Results

For evaluation of our algorithms we first used process graphs generated for experimental purpose. We considered two-cluster architectures consisting of 2, 4, 6, 8 and 10 nodes, half on the TTC and the other half on the ETC, interconnected by a gateway. 40 processes were assigned to each node, resulting in applications of 80, 160, 240, 320 and 400 processes. Message sizes were randomly chosen between 8 and 32 bytes. 30 examples were generated for each application dimension, thus a total of 150 applications were used for experimental evaluation. Worst-case execution times and message lengths were assigned randomly using both uniform and exponential distribution. All experiments were run on a SUN Ultra 10.

In order to provide a basis for the evaluation of our heuristics we have developed two simulated annealing (SA) based algorithms. Both are based on the moves presented in the previous section. The first one, named SA Schedule (SAS), was set to preform moves such that δ_Γ is minimized. The second one, SA Resources (SAR), uses s_{total} as the cost function to be minimized. Very long and expensive runs have been performed with each of the SA algorithms, and the best ever so-

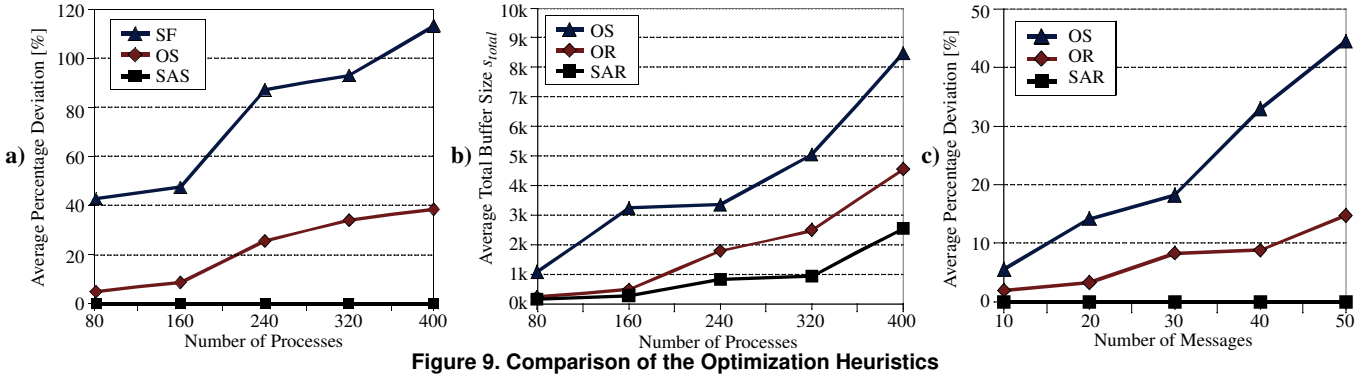


Figure 9. Comparison of the Optimization Heuristics

lution produced has been considered a close to the optimum value.

The first result concerns the ability of our heuristics to produce schedulable solutions. We have compared the degree of schedulability δ_Γ obtained from our OptimizeSchedule (OS) heuristic (Figure 8) with the near-optimal values obtained by SAS. Figure 9a presents the average percentage deviation of the degree of schedulability produced by OS from the near-optimal values obtained with SAS. Together with OS, a straightforward approach (SF) is presented. For SF we considered a TTC bus configuration consisting of a straightforward ascending order of allocation of the nodes to the TDMA slots; the slot lengths were selected to accommodate the largest message sent by the respective node, and the scheduling has been performed by the MultiClusterScheduling algorithm in Figure 5.

Figure 9a shows that when considering the optimization of the access to the communication channel, and of priorities, the degree of schedulability improves dramatically compared to the straightforward approach. The greedy heuristic OptimizeSchedule performs well for all the graph dimensions, having run-times which are more than two orders of magnitude smaller than with SAS. In the figure, only the examples where all the algorithms have obtained schedulable systems were presented. The SF approach failed to find a schedulable system in 26 out of the total 150 applications.

Next, we are interested to evaluate the heuristics for minimizing the buffer sizes needed to run a schedulable application. Thus, we compare the total buffer need s_{total} obtained by the OptimizeResources (OR) function with the near-optimal values obtained when using simulated annealing, this time with the cost function s_{total}. To find out how relevant the buffer optimization problem is, we have compared these results with the s_{total} obtained by the OS approach, which is interested only to obtain a schedulable system, without any other concern. As shown in Figure 9b, OR is able to find schedulable systems with a buffer need half of that needed by the solutions produced with OS. The quality of the solutions obtained by OR is also comparable with the one obtained with simulated annealing (SAR).

Another important aspect of our experiments was to determine the difficulty of resource minimization as the number of messages exchanged over the gateway increases. For this, we have generated applications of 160 processes with 10, 20, 30, 40, and 50 messages exchanged between the TTC and ETC clusters. 30 applications were generated for each number of messages. Figure 9c shows the average percentage deviation of the buffer sizes obtained with OR and OS from the near-optimal results obtained by SAR. As the number of inter-cluster messages increases, the problem becomes more complex. The OS approach degrades very fast, in terms of buffer sizes, while OR is able to find good quality results even for intense inter-cluster traffic.

When deciding on which heuristic to use for design space exploration or system synthesis, an important issue is the execution time. In average, our optimization heuristics needed a couple of minutes to produce results, while the simulated annealing approaches (SAS and SAR) had an execution time of up to three hours.

Finally, we considered a real-life example implementing a vehicle cruise controller. The process graph that models the cruise controller has 40 processes, and it was mapped on an architecture consisting of a TTC and an ETC, each with 2 nodes, interconnected by a gateway. The "speedup" part of the model has been mapped on the ETC while the other processes were mapped on the TTC. We considered one mode of operation with a deadline of 250 ms. The straightforward approach SF produced an end-to-end response time of 320 ms, greater than the deadline, while both the OS and SAS heuristics produced a schedulable system with a worst-case response time of 185 ms. The total buffer need of the solution determined by OS was 1020 bytes. After optimization with OR a still schedulable solution with a buffer need reduced by 24% has been generated, which is only 6% worse than the solution produced with SAR.

7. Conclusions

We have presented in this paper an approach to schedulability analysis for the synthesis of multi-cluster distributed embedded systems consisting of time-triggered and event-triggered clusters, interconnected via gateways. The main contribution is the development of a schedulability analysis for such systems, including determining the worst-case queuing delays at the gateway and the bounds on the buffer size needed for running a schedulable system.

Optimization heuristics for system synthesis have been proposed, together with simulated annealing approaches tuned to find near-optimal results. The first heuristic, OS, was concerned with obtaining a schedulable system, by maximizing the degree of schedulability. Our second heuristic, OR, aimed at producing schedulable systems with a minimal buffer size need.

References

[1] N. Audsley, A. Burns, et. al., "Fixed Priority Preemptive Scheduling: An Historical Perspective", Real-Time Systems, 8(2/3), 173-198, 1995.

[2] N. Audsley, K. Tindell, A. et. al., "The End of Line for Static Cyclic Scheduling?", Euromicro Workshop on Real-Time Systems, 36-41, 1993.

[3] F. Balarin, L. Lavagno, et. al., "Scheduling for Embedded Real-Time Systems", IEEE Design and Test of Computers, Jan.-Mar., 71-82, 1998.

[4] R. Bosch GmbH, "CAN Specification Version 2.0", 1991.

[5] P. Eles et al., "Scheduling with Bus Access Optimization for Distributed Embedded Systems", IEEE Trans. on VLSI Systems, 472-491, 2000.

[6] FlexRay Requirements Specification, http://www.flexray-group.com/.

[7] J. J.G. Garcia, M. G. Harbour, "Optimized Priority Assignment for Tasks and Messages in Distributed Hard Real-Time Systems", Proc. Workshop on Parallel and Distributed R-T Systems, 124-132, 1995.

[8] H. Kopetz, "Real-Time Systems - Design Principles for Distributed Embedded Applications", Kluwer Academic Publishers, 1997.

[9] H. Lönn, J. Axelsson, "A Comparison of Fixed-Priority and Static Cyclic Scheduling for Distributed Automotive Control Applications", Euromicro Conference on Real-Time Systems, 142-149, 1999.

[10] J. C. Palencia, M. G. Harbour, "Schedulability Analysis for Tasks with Static and Dynamic Offsets", Proc. Real-Time Systems Symp., 1998.

[11] T. Pop, P. Eles, Z. Peng, "Holistic Scheduling and Analysis of Mixed Time/Event-Triggered Distributed Embedded Systems", Intl. Symposium on Hardware/Software Codesign, 187-192, 2002.

[12] P. Pop, P. Eles, Z. Peng, "Bus Access Optimization for Distributed Embedded Systems Based on Schedulability Analysis", Proc. Design Automation and Test in Europe, 567-574, 2000.

[13] P. Pop, P. Eles, Z. Peng, "Scheduling with Optimized Communication for TT Embedded Systems", Intl. Workshop on Hw-Sw Codesign, 178-182, 1999.

[14] K. Tindell, "Adding Time-Offsets to Schedulability Analysis", Department of Computer Science, University of York, Report No. YCS-94-221, 1994.

[15] K. Tindell, A. Burns, A. J. Wellings, "Calculating CAN Message Response Times", Control Engineering Practice, 3(8), 1163-1169, 1995.

[16] K. Tindell, J. Clark, "Holistic Schedulability Analysis for Distributed Hard Real-Time Systems", Microprocessing & Microprogramming, Vol. 50, No. 2-3, 1994.

[17] J. Xu, D. L. Parnas, "On satisfying timing constraints in hard-real-time systems", IEEE Transactions on Software Engineering, 19(1), 1993.

[18] T. Y. Yen, W. Wolf, "Hardware-Software Co-Synthesis of Distributed Embedded Systems", Kluwer Academic Publishers, 1997.