Schedulability-Driven Frame Packing for Multi-Cluster Distributed Embedded Systems

Paul Pop, Petru Eles, Zebo Peng

Computer and Information Science Dept.,
Linköping University,
SE-581 83 Linköping, Sweden
{paupo, petel, zebpe}@ida.liu.se

ABSTRACT

We present an approach to frame packing for multi-cluster distributed embedded systems consisting of time-triggered and event-triggered clusters, interconnected via gateways. In our approach, the application messages are packed into frames such that the application is schedulable. Thus, we have also proposed a schedulability analysis for applications consisting of mixed event-triggered and time-triggered processes and messages, and a worst case queuing delay analysis for the gateways, responsible for routing inter-cluster traffic. Optimization heuristics for frame packing aiming at producing a schedulable system have been proposed. Extensive experiments and a real-life example show the efficiency of our frame-packing approach.

Categories and Subject Descriptors

D.4.1 [**Operating Systems**]: Process Management–*scheduling*

General Terms

Algorithms, Performance, Design, Theory

Keywords

Frame Packing, Schedulability Analysis, Multi-Clusters

1. INTRODUCTION

Depending on the particular application, an embedded system has certain requirements on performance, cost, dependability, size, etc. For hard real-time applications the timing requirements are extremely important. Thus, in order to function correctly, an embedded system implementing such an application has to meet its deadlines.

Process scheduling and schedulability analysis has been intensively studied for the past decades. The reader is referred to [1, 3] for surveys on this topic. There are two basic approaches for handling activities 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, 21]. 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].

A few approaches have been proposed for the schedulability analysis of distributed real-time systems, taking into consideration both process and communication scheduling. In [18] and [19] 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 [13] 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 [10]. 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 ET and TT processes to share the same processor as well as dynamic and static communications to share the same bus. In [12] 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 [14] we have proposed an approach to schedulability analysis for such systems, including also buffer need analysis and worst case queuing delay analysis of inter-cluster traffic. There we concentrated on optimization heuristics aimed at producing a schedulable system such that *communication buffer sizes are minimized*.

We have, however, not addressed the issue of *frame packing*, which is of utmost importance in cost-sensitive embedded systems where resources, such as communication bandwidth, have to be fully utilized [9, 16, 20]. In both TTP and CAN protocols messages are not sent independently, but several messages having similar timing properties are usually packed into frames. In many application areas, like automotive electronics, messages range from one single bit (e.g., the state of a device) to a couple of bytes (e.g., vehicle speed, etc.). Transmitting such small messages one per frame would create a high communication overhead, which can cause long delays leading to an unschedulable system. For example, 48 bits have to be transmitted on CAN for delivering one single bit of application data. Moreover, a given frame configuration defines the exact behavior of a node on the network, which is very important when integrating nodes from different suppliers.

The issue of frame packing (sometimes referred to as frame compiling) has been previously addressed separately for the CAN and the TTP. In [16, 20] CAN frames are created based on the properties of the messages, while in [9] a "cluster compiler" is used to derive the frames for a TT system which uses TTP as the communication protocol. However, researchers have not addressed frame packing on multi-cluster systems implemented using both ET and TT clusters, where the interaction between the ET and TT processes of a hard real-time application has to be very carefully considered in order to guarantee the timing constraints. As our multi-cluster scheduling strategy in section 4 shows, the issue of frame packing cannot be addressed separately for each type of cluster, since the inter-cluster communication creates a circular dependency.

Therefore, in this paper, we concentrate on the issue of packing messages into frames, for multi-cluster distributed embedded systems consisting of time-triggered and event-triggered clusters, interconnected via gateways. We are interested to obtain that frame configuration which would produce a schedulable system. We have updated our schedulability analysis presented in [14] to account for the frame packing, and we have proposed two optimization heuristics that use the schedulability analysis as a driver towards a frame configuration that leads to a schedulable system.

The paper is organized in 7 sections. The next section presents the hardware and software architectures as well as the application model of our systems. Section 3 introduces more precisely the problem 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 frame generation. The last two sections present the experimental results and conclusions.

2. APPLICATION MODEL AND SYSTEM ARCHITECTURE

2.1 Hardware Architecture

We consider architectures consisting of several clusters, interconnected by gateways (Figure 1 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), see Figure 2.

Communication between the nodes on a TTC is based on the TTP [8]. The TTP integrates all the services necessary for fault-tolerant real-time systems. The bus access scheme is time-division multiple-access (TDMA), meaning that each node N_i on the TTC, including the gateway node, can transmit only during a predetermined time interval, the TDMA slot S_i. In such a slot, a node can send several messages packed in a frame. A sequence of slots corresponding to all the nodes in the architecture 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 sequence and length of the slots are the same for all the TDMA rounds. However, the length and contents of the frames may differ.

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.

There are two types of frames in the TTP. The initialization frames, or I-frames, which are needed for the initialization of a node, and the normal frames, or N-frames, which are the data frames containing, in their data field, the application messages. A TTP data frame (Figure 1) consists of the following fields: start of frame bit (SOF), control field, a data field of up to 8 bytes containing one or more messages, and a cyclic

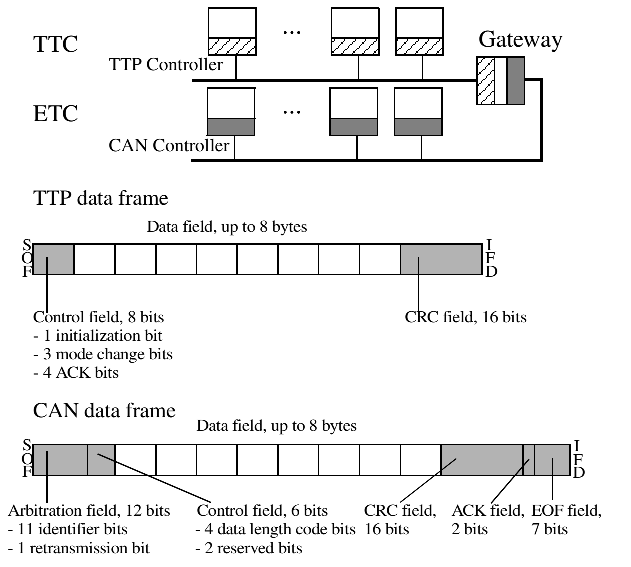


Figure 1. A System Architecture Example

redundancy check (CRC) field. Frames are delimited by the inter-frame delimiter (IDF, 3 bits). Thus, the data efficiency for such a frame that carries 8 bytes of application data, i.e., the percentage of transmitted bits which are the actual data bits needed by the application, is 69.5% (64 data bits transmitted in a 92 bits frame). Note that no identifier bits are necessary, as the TTP controllers know from their MEDL what frame to expect at a given point in time.

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 frame with the highest priority wins the contention. Frame priorities are unique and are encoded in the frame identifiers, which are the first bits to be transmitted on the bus.

In the case of CAN, there are four frame types: data frame, remote frame, error frame, and overload frame. We are interested in the composition of the data frame, depicted in Figure 1. A data frame contains seven fields: SOF, arbitration field that encodes the 11 bits frame identifier, a control field, a data field up to 8 bytes, a CRC field, an acknowledgement (ACK) field, and an end of frame field (EOF), thus having a data efficiency of 57.6%.

2.2 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 2 we illustrate our message passing mechanism. Here we concentrate on the communication between processes located on different clusters. For message passing within a TTC the reader is directed to [15], while the infrastructure needed for communications on an ETC has been detailed in [18].

Let us consider the example in Figure 2, where we have an application consisting of four processes mapped on 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 and P_3 send messages m_3 and m_4 to P_4. All messages have a size of one byte.

The transmission of messages from the TTC to the ETC takes place in the following way (see Figure 2). 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 packing them into

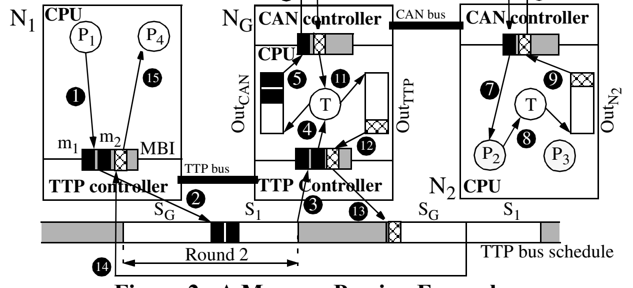


Figure 2. A Message Passing Example

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, 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). Let us assume that on the ETC messages m_1 and m_2 are sent independently, one per frame. The highest priority frame in the queue, in our case the frame f_1 containing m_1, will tentatively be broadcast on the CAN bus (5). Whenever f_1 will be the highest priority frame 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 f_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 2 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 the frame f_3 containing m_3 has the highest priority on the bus, it will be removed from the queue (9) and broadcast on the CAN bus (10). Several messages can be packed into a frame in order to increase the efficiency of data transmission. For example, m_3 can wait in the queue until m_4 is produced by P_3, in order to be packed together with m_4 in a frame. When f_3 arrives at the gateway's CAN controller it raises an interrupt. Based on this interrupt, the gateway transfer process T is activated, and m_3 is unpacked from f_3 and 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 data field of the frame traveling in 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 frame packing approach, we generate all the MEDLs on the TTC (i.e., the TT frames and the sequence of the TDMA slots), as well as the ET frames and their priorities on the ETC such that the global system is schedulable.

2.3 Application Model

We model an application Γ as a set of process graphs $G_i \in \Gamma$ (see Figure 3). 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

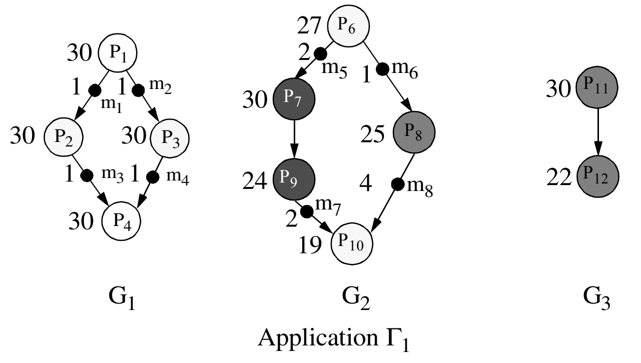


Figure 3. Application Model

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 3).

Each process P_i is mapped on a processor $processor_{P_i}$ (mapping represented by hashing in Figure 3), 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 the periods).

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.

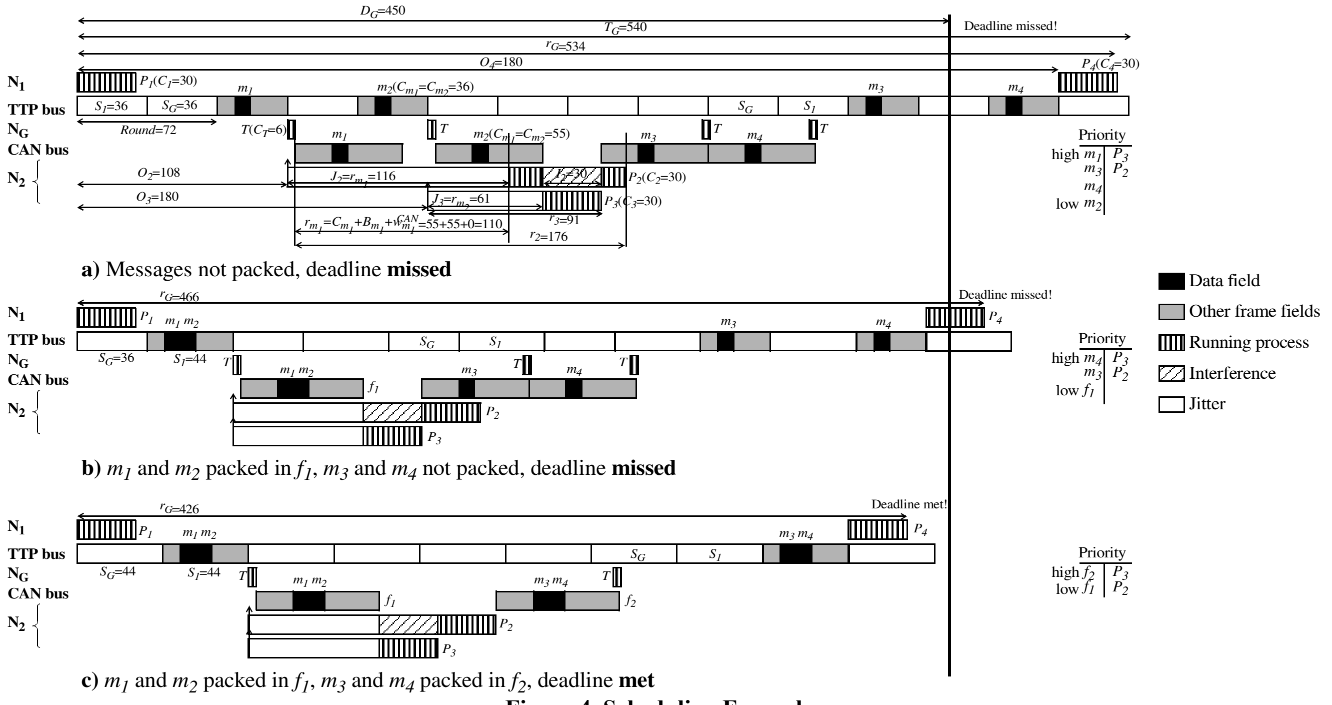
We are interested to find a mapping of messages to frames (a frame packing configuration) denoted by a 4-tuple $\psi = <\alpha, \pi, \beta, \sigma>$ such that the application Γ is schedulable. Once a schedulable system is found, we are interested to further improve the "degree of schedulability", so the application can potentially be implemented on a cheaper hardware architecture (with slower buses and processors).

Determining a frame configuration ψ means deciding on:

- The mapping of application messages transmitted on the ETC to frames (the set of ETC frames α), and their relative priorities, π. Note that the ETC frames α have to include messages transmitted from an ETC node to a TTC node, messages transmitted inside the ETC cluster, and those messages transmitted from the TTC to the ETC.
- The mapping of messages transmitted on the TTC to frames, denoted by the set of TTC frames β, and the sequence σ of slots in a TDMA round. The slot sizes are determined based on the set β, and are calculated such that they can accommodate the largest frame sent in that particular slot. We consider that messages transmitted from the ETC to the TTC are not statically allocated to frames. Rather, we will dynamically pack messages originating from the ETC into the "gateway frame", for which we have to decide the data field length (see section 2.2).

Although we consider only a two cluster system, 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 mapped on the two-cluster system as indicated



a) Messages not packed, deadline **missed**

b) m_1 and m_2 packed in f_1, m_3 and m_4 not packed, deadline **missed**

c) m_1 and m_2 packed in f_1, m_3 and m_4 packed in f_2, deadline **met**

Figure 4. Scheduling Examples

in Figure 2. In the system configuration of Figure 4a we consider that, on the TTP bus, the node N_1 transmits in the first slot (S_1) of the TDMA round, while the gateway transmits in the second slot (S_G). The priorities of processes and messages in the ETC are illustrated in the figure. In such a setting, G will miss its deadline, which was set at 450 ms. Changing the frame configuration as in Figure 4b, so that messages m_1 and m_2 are packed into frame f_1 and slot S_G of the gateway comes first, processes P_2 and P_3 will receive m_1 and m_2 sooner and thus reduce the response time to 466, which is still larger than the deadline. In Figure 4c, we also pack m_3 and m_4 into f_2. In such a situation, the sending of m_3 will have to be delayed until m_4 is queued by P_2. Nevertheless, the response time of the application is further reduced to 426, which means that the deadline is met, thus the system is schedulable.

However, packing more messages will not necessarily reduce the response times further, as it might increase too much the response times of messages that have to wait for the frame to be assembled, like is the case with message m_3 in Figure 4c. We are interested to find that frame packing which would lead to a schedulable system.

4. MULTI-CLUSTER SCHEDULING

In [14] we have proposed an analysis for hard real-time applications mapped on multi-cluster systems. We have, however, not addressed the issue of frame packing. In this section we briefly present the analysis developed in [14], showing how it can be extended to handle frames.

The aim of such an analysis is to find out if a system is schedulable, i.e., all the timing constraints are met. 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 whether or not a system is schedulable is given by a *schedulability analysis*.

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 [11, 17, 22], and has been also extended to consider the CAN protocol [18]. 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=108, 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

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

Figure 5. The MultiClusterScheduling Algorithm

is true for messages, their offset indicating the earliest possible transmission time.

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 4b packing m_1 and m_2 in the same frame 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 arrivals of m_3 and m_4.

In our 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 which receive 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 frame configuration ψ, 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 and the TTC frame configuration (the mapping β of TTC messages to frames and the slot sequence σ) 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 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, the ETC frame configuration (the mapping α of ETC messages to frames, and the frame priorities π), and it produces the set ρ of the worst case response times. For the priorities of processes used in the response time calculation we use the "heuristic optimized priority assignment" (HOPA) approach in [7], where priorities for processes in a distributed real-time system are determined, using knowledge of the factors that influence the timing behavior, such that the "degree of schedulability" of the system is improved.

We have extended the framework provided by [17, 18] 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:

$$w_i = B_i + \sum_{\forall j \in hp(P_i)} \left\lceil \frac{w_i + J_j - O_{ij}}{T_j} \right\rceil_0 C_j .$$

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 $\lceil x \rceil_0$ operator is the positive ceiling, which returns the smallest integer greater than x, or 0 if x is negative.

The same analysis can be applied for frames on the CAN bus: $r_f=J_f+w_f+C_f$, where

$$J_f = \max_{\forall m \in f} (r_{S(m)})$$

is the jitter of frame f which in the worst case is equal to the largest worst case response time $r_{S(m)}$ of a sender process $P_{S(m)}$ which sends message m packed into frame f, w_f is the *worst-case queuing delay* experienced by f at the communication controller, and C_f is the worst-case time it takes for a frame f to reach the destination controller. On CAN, C_f depends on the frame configuration and the size of the data field, s_f, while on TTP it is equal to the slot size in which f is transmitted. Moreover, the response time of message m packed into a frame f can be determined by observing that $r_m=r_f$.

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 frame is calculated depending on the type of message passing employed:

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

The frames 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 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_f^{Ni} and w_f^{CAN} are similar. Once f is the highest priority frame in the Out_{CAN} queue, it will be sent by the gateway's CAN controller as a regular CAN frame, therefore the same equation for w_f can be used:

$$w_f = B_f + \sum_{\forall j \in hp(f)} \left\lceil \frac{w_f + J_j - O_{fj}}{T_j} \right\rceil_0 C_j .$$

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

$$B_f = \max_{\forall k \in lp(f)} (C_k) \quad .$$

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

4.1.2 From ETC to TTC

The time a frame f has to spend in the Out_{TTP} queue in the worst case depends on the total size of messages queued ahead of f (Out_{TTP} is a FIFO queue), the size S_G of the data field of the frame fitting into 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:

$$w_f^{TTP} = B_f + \left\lfloor \frac{S_f + I_f}{S_G} \right\rfloor T_{TDMA}$$

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

$$I_f = \sum_{\forall j \in hp(f)} \left\lceil \frac{w_f^{TTP} + J_f - O_{fj}}{T_j} \right\rceil_0 s_j$$

where the frame jitter

$$J_f = \max_{\forall m \in f} (r_{S(m)})$$

is the largest worst case the response time among the processes that send a message m packed in f.

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

4.2 Response Time Analysis Example

Figure 4a presents the values of the analysis parameters (worst case response time, offset, jitter, interference) for the system in Figure 2. The jitter of P_2 depends on the response time of message m_1, J_2=r_{m_1}. Similarly, J_3=r_{m_2}. We have considered that J_{m_1}=J_{m_2}=r_T=6 (where r_T is the response time of the gateway transfer process T). Note that despite the fact that m_1 is the highest priority message in Figure 4a, it can be blocked by lower priority messages which are just being sent, and thus B_{m_1}=55, resulting r_{m_1}=6+55+55=116.

The response time equations are recurrent, and they will converge if the processor and bus utilization are under 100% [19]. Considering a TDMA round of 72 ms, with two slots each of 36 ms as in Figure 4a, r_T=6 ms, 55 ms for the transmission times on CAN for each message, 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 application G will be r_G=O_4+r_4=534, which is greater than D_G=450, thus the system is not schedulable.

5. FRAME PACKING STRATEGY

Once we have a technique to determine if a system is schedulable, we can concentrate on optimizing the packing of messages to frames.

Such an optimization problem is NP complete [16], thus obtaining the optimal solution is not feasible. We propose two frame packing optimization strategies, one based on a simulated annealing approach, while the other is based on a greedy heuristic that uses intelligently the problem-specific knowledge in order to explore the design space.

In order to drive our optimization algorithms towards schedulable solutions, we characterize a given frame packing configuration using the degree of schedulability of the application. The *degree of schedulability* [13] is calculated as:

$$\delta_\Gamma = \begin{cases} c_1 = \sum_{i=1}^{n} max(0, R_i - D_i), \text{ if } c_1 > 0 \\ c_2 = \sum_{i=1}^{n} (R_i - D_i), \text{ if } c_1 = 0 \end{cases}$$

where n is the number of processes in the application. If the application is not schedulable, the term c_1 will be positive, and, in this case, the cost function is equal to c_1. However, if the process set is schedulable, c_1 = 0 and we use c_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 c_2 means that we have improved the response times of the processes, so the application can potentially be implemented on a cheaper hardware architecture (with slower processors and/or buses).

5.1 Frame Packing with Simulated Annealing

The first algorithm we have developed is based on a simulated annealing (SA) strategy. The main feature of a SA strategy is that it tries to escape from a local optimum by randomly selecting a new solution from the neighbors of the current solution. The new solution is accepted if it is an improved solution. However, a worse solution can also be accepted with a certain probability that depends on the deterioration of the cost function and on a control parameter called temperature.

In Figure 6 we give a short description of this algorithm. An essential component of the algorithm is the generation of a new solution x' starting from the current one x_{now}. The neighbors of the current solution x_{now} are obtained by performing transformations (called moves) on the current frame configuration ψ. We consider the following moves:

- moving a message m from a frame f_1 to another frame f_2 (or moving m into a separate single-message frame);
- swapping the priorities of two frames in α;
- swapping two slots in the sequence σ of slots in a TDMA round.

For the implementation of this algorithm, the parameters *TI* (initial temperature), *TL* (temperature length), ε (cooling ratio), and the stopping criterion have to be determined. They define the "cooling schedule" and have a decisive impact on the quality of the solutions and the CPU time consumed. We are interested to obtain values for *TI*, *TL* and ε that will guarantee the finding of good quality solutions in a short time.

We performed very long and expensive runs with the SA algorithm, and the best ever solution produced has been considered as the optimum. Based on further experiments we have determined the parameters of the SA algorithm so that the optimization time is reduced as much as possible but the near-optimal result is still produced. For example, for the graphs with 320 nodes, *TI* is 700, *TL* is 500 and ε is 0.98. The algorithm stops if for three consecutive temperatures no new solution has been accepted.

```
SimulatedAnnealing(Γ)
  -- given an application Γ finds out if it is schedulable and produces
  -- the configuration ψ=<α, π, β, σ> leading to the smallest δ_Γ
  construct an initial frame configuration x^now
  temperature = initial temperature TI
  repeat
    for i = 1 to temperature length TL do
      generate randomly a neighboring solution x' of x^now
      δ = MultiClusterScheduling(Γ, x') -
        MultiClusterScheduling(Γ, x^now)
      if δ < 0 then x^now = x'
      else
        generate q = random (0, 1)
        if q < e^(- δ / temperature) then x^now = x' end if
      end if
    end for
    temperature = ε * temperature;
  until stopping criterion is met
  return SchedulabilityTest(Γ, ψ_best), solution ψ_best
    corresponding to the best degree of schedulablity δ_Γ
end SimulatedAnnealing
```

Figure 6. The Simulated Annealing Algorithm

5.2 Frame Packing Greedy Heuristic

The OptimizeFramePacking greedy heuristic (Figure 7) constructs the solution by progressively selecting the best candidate in terms of the degree of schedulability.

We start by observing that all activities taking place in a multi-cluster system are ordered in time using the offset information, determined in the StaticScheduling function based on the response times known so far and the application structure (i.e., the dependencies in the process graphs). Thus, our greedy heuristic outlined in Figure 7, starts with building two lists of messages ordered according to the ascending value of their offsets, one for the TTC, $messages_\beta$, and one for ETC, $messages_\alpha$. Our heuristic is to consider for packing in the same frame messages which are adjacent in the ordered lists. For example, let us consider that we have three messages, m_1 of 1 byte, m_2 2 bytes and m_3 3 bytes, and that messages are ordered as m_3, m_1, m_2 based on the offset information. Also, assume that our heuristic has suggested two frames, frame f_1 with a data field of 4 bytes, and f_2 with a data field of 2 bytes. The PackMessages function will start with m_3 and pack it in frame f_1. It continues with m_2, which is also packed into f_1, since there is space left for it. Finally, m_3 is packed in f_2, since there is no space left for it in f_1. The OptimizeFramePacking tries to determine, using the for-each loops in Figure 7, the allocation of frames, i.e., the number of frames and their sizes, for each cluster. The actual mapping of messages to frames will be performed by the PackMessages function as described.

As an initial TDMA slot sequence $\sigma_{initial}$ on the TTC, OptimizeFramePacking assigns 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, $size_{S_i}=size_{largest\ message}$.

Then, the algorithm looks, in the innermost for-each loops, for the optimal frame configuration α. This means deciding on how many frames to include in α, and which are the best sizes for them. In α there can be any number of frames, from one single frame to n_α frames (in which case each frame carries one single message). Thus, several numbers of frames are tried, each tested for RecomendedSizes to see if it improves the current configuration. The RecomendedSizes($messages_\alpha$) list is built recognizing that only messages adjacent in the $messages_\alpha$ list will be packed into the same frame. Sizes of frames are determined as a sum resulted from adding the sizes of combinations of adjacent messages, not exceeding 8 bytes. For the previous example, with m_1, m_2 and m_3, of 1, 2 and 3 bytes, respectively, the frame sizes recommended will be of 1, 2, 3, 4, and 6 bytes. A size of 5 bytes will not be recommended since there are no adjacent messages that can be summed together to obtain 5 bytes of data.

Once a configuration α_{best} for the ETC, minimizing δ_Γ, has been determined (considering for π, β, σ the initial values

```
OptimizeFramePacking($\Gamma$)
  -- given an application $\Gamma$ finds out if it is schedulable and produces the configuration $\psi$=<$\alpha$, $\pi$, $\beta$, $\sigma$> leading to the smallest $\delta_\Gamma$
  -- build the message lists ordered ascending on their offsets
  $messages_\beta$=ordered list of $n_\beta$ messages on the TTC; $messages_\alpha$=ordered list of $n_\alpha$ messages on the ETC
  -- build an initial frame configuration $\psi$=<$\alpha$, $\pi$, $\beta$, $\sigma$>
  $\beta$=$messages_\beta$; $\alpha$=$messages_\alpha$ -- initially, each frame carries one message
  for each slot $S_i \in \sigma$ do $S_i$=$N_i$; $size_{S_i}$=$size_{largest\ message}$ end for -- determine an initial TDMA slot sequence $\sigma$
  $\pi_{initial}$=HOPA -- calculate the priorities $\pi$ according to the HOPA heuristic

  for each slot $S_i \in \sigma_{current}$ do -- find the best allocation of slots, the TDMA slot sequence $\sigma_{current}$
    for each node $N_j \in$ TTC do
      $\sigma_{current}.S_i$=$N_j$; $\sigma_{current}.S_j$=$N_i$-- allocate $N_j$ tentatively to $S_i$, $N_i$ gets slot $S_j$
      for each $\beta_{current}$ having a number of 1 to $n_\beta$ frames do -- determine the best frame packing configuration $\beta$ for the TTC
        for each frame $f_i \in \beta_{current}$ do
          for each frame size $S_f \in$ RecomendedSizes($messages_\beta$) do -- determine the best frame size for $f_i$
            $\beta_{current}.f_i.S$=$S_f$
            for each $\alpha_{current}$ having a number of 1 to $n_\alpha$ frames do -- determine the best frame packing configuration $\alpha$ for the ETC
              for each frame $f_j \in \alpha_{current}$ do
                for each frame size $S_f \in$ recomended_sizes($messages_\alpha$) do -- determine the best frame size for $f_j$
                  $\alpha_{current}.f_j.S$=$S_f$
                  $\psi_{current}$=<$\alpha_{current}$, $\pi_{initial}$, $\beta_{current}$, $\sigma_{current}$>
                  PackMessages($\psi_{current}$, $messages_\beta \cup messages_\alpha$)
                  $\delta_\Gamma$=MultiClusterScheduling($\Gamma$, $\psi_{current}$)
                  -- remember the best configuration so far
                  if $\delta_\Gamma(\psi_{current})$ is best so far then $\psi_{best}$ = $\psi_{current}$ end if
                end for
              end for; if $\psi_{best}$ exists then $\alpha_{current}.f_j.S$=size of frame $f_j$ in the configuration $\psi_{best}$ end if
            end for; if $\psi_{best}$ exists then $\alpha_{current}$=frame set $\alpha$ in the configuration $\psi_{best}$ end if
          end for
        end for; if $\psi_{best}$ exists then $\beta_{current}.f_i.S$=size of frame $f_i$ in the configuration $\psi_{best}$ end if
      end for; if $\psi_{best}$ exists then $\beta_{current}$=frame set $\beta$ in the configuration $\psi_{best}$ end if
    end for; if $\psi_{best}$ exists then $\sigma_{current}.S_i$=node in the slot sequence $\sigma$ in the configuration $\psi_{best}$ end if
  end for
  return SchedulabilityTest($\Gamma$, $\psi_{best}$), $\psi_{best}$
end OptimizeFramePacking
```

Figure 7. The OptimizeFramePacking Algorithm

determined at the beginning of the algorithm), the algorithm looks for the frame configuration β which will further improve δ_Γ. The degree of schedulability δ_Γ (the smaller the value, the more schedulable the system) is calculated based on the response times produced by the MultiClusterScheduling algorithm. After a β_{best} has been decided, the algorithm looks for a slot sequence σ, starting with the first slot and tries to find the node which, when transmitting in this slot, will reduce δ_Γ. The algorithm continues in this fashion, recording the best ever ψ_{best} configurations obtained, in terms of δ_Γ, and thus the best solution ever is reported when the algorithm finishes. In the inner loops of the heuristic we will not change the frame priorities $\pi_{initial}$ set at the beginning of the algorithm.

6. EXPERIMENTAL RESULTS

For the 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 1 bit and 2 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. For the communication channels we considered a transmission speed of 256 Kbps and a length below 20 meters. All experiments were run on a SUN Ultra 10.

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 OptimizeFramePacking (OFP) heuristic (Figure 7) with the near-optimal values obtained by SA (Figure 6). Obtaining solutions that have a higher degree of schedulability means obtaining tighter response times, increasing the chances of meeting the deadlines.

Figure 8a presents the average percentage deviation of the degree of schedulability produced by OFP from the near-optimal values obtained with SA. Together with OFP, a straightforward approach (SF) is presented. The SF approach does not consider frame packing, and thus each message is transmitted independently in a frame. Moreover, 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 frame sent by the respective node, and the scheduling has been performed by the MultiClusterScheduling algorithm in Figure 5.

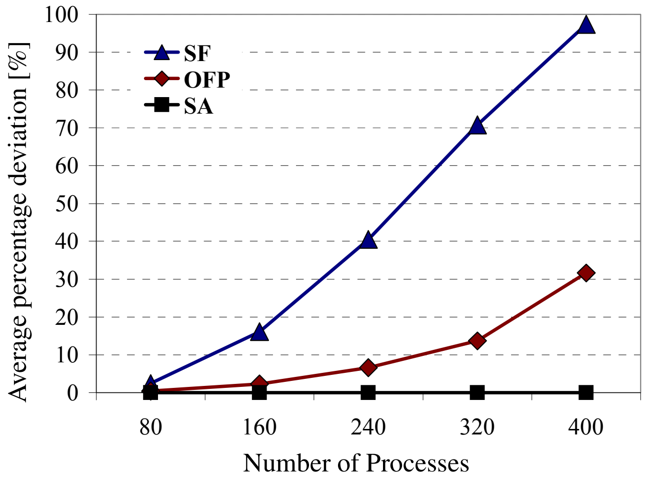
Figure 8a shows that when packing messages to frames, the degree of schedulability improves dramatically compared to the straightforward approach. The greedy heuristic OptimizeFramePacking performs well for all the graph dimensions, having run-times which are more than two orders of magnitude smaller than with SA.

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 approach had an execution time of up to 6 hours.

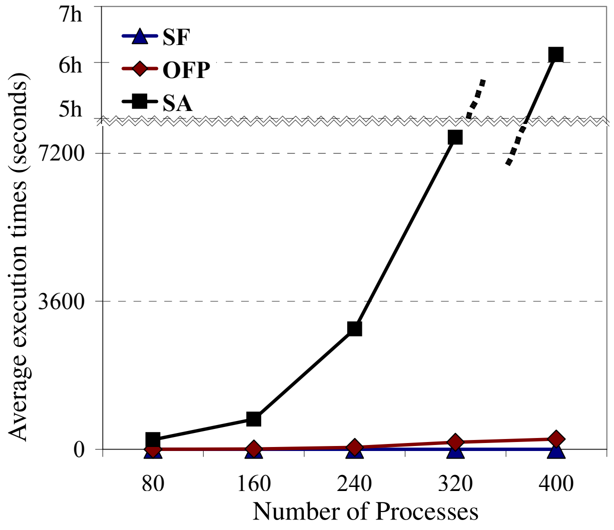
Finally, we considered a real-life example implementing a vehicle cruise controller. The process graph that models the cruise controller has 40 processes, and it was mapped on an architecture consisting of a TTC and an ETC, each with 2 nodes, interconnected by a gateway. The "speedup" part of the model 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 OFP and SA heuristics produced a schedulable system with a worst-case response time of 172 ms.

7. CONCLUSIONS

We have presented in this paper an approach to schedulability-driven frame packing for the synthesis of multi-cluster distributed embedded systems consisting of time-triggered and event-triggered clusters, interconnected via gateways. We have also presented an update of the schedulability analysis for



a) Average Percentage Deviation from SA



b) Average Execution Time

Figure 8. The Frame Packing Heuristics

multi-cluster systems to handle frames, including determining the worst-case queuing delays at the gateway nodes.

The main contribution is the development of two optimization heuristics for frame configuration synthesis which are able to determine frame configurations that lead to a schedulable system. We have shown that by considering the frame packing problem, we are able to synthesize schedulable hard-real time systems and to potentially reduce the overall cost of the architecture. The greedy approach is able to produce accurate results in a very short time, therefore it can be used for performance estimation as part of a larger design space exploration cycle. SA is able to find near-optimal results in reasonable time, and can be used for the synthesis of the final implementation of the system.

ACKNOWLEDGEMETNS

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

REFERENCES

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

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

[3] F. Balarin, L. Lavagno, P. Murthy, A. Sangiovanni-Vincentelli, "Scheduling for Embedded Real-Time Systems", IEEE Design and Test of Computers, January-March, 71-82, 1998.

[4] Robert Bosch GmbH, "CAN Specification, Version 2.0", http://www.can.bosch.com/, 1991.

[5] P. Eles, A. Doboli, P. Pop, Z. Peng, "Scheduling with Bus Access Optimization for Distributed Embedded Systems", IEEE Transactions on VLSI Systems, 472-491, 2000.

[6] The FlexRay Group, "FlexRay Requirements Specification, Version 2.0.2", http://www.flexray-group.com/, 2002.

[7] J. J. Gutiérrez García, M. González Harbour, "Optimized Priority Assignment for Tasks and Messages in Distributed Hard Real-Time Systems", Proceedings of the Workshop on Parallel and Distributed Real-Time Systems, 124-132, 1995.

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

[9] H. Kopez, R. Nossal, "The Cluster-Compiler – A Tool for the Design of Time Triggered Real-Time Systems", Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Real-Time Systems, 108-116, 1995.

[10] 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.

[11] J. C. Palencia, M. González Harbour, "Schedulability Analysis for Tasks with Static and Dynamic Offsets", Proceedings of the 19th IEEE Real-Time Systems Symposium, 26-37, 1998.

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

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

[14] P. Pop, P. Eles, Z. Peng, "Schedulability Analysis and Optimization for the Synthesis of Multi-Cluster Distributed Embedded Systems", Design Automation and Test in Europe Conference, 2003 (to be published).

[15] P. Pop, P. Eles, Z. Peng, "Scheduling with Optimized Communication for Time Triggered Embedded Systems", International Workshop on Hardware-Software Codesign, 178-182, 1999.

[16] K. Sandström, C. Norström, "Frame Packing in Real-Time Communication", Proceedings of the International Conference on Real-Time Computing Systems and Applications, 399-403, 2000.

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

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

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

[20] A. Rajnak, K. Tindell, L. Casparsson, "Volcano Communications Concept", Volcano Communication Technologies AB, 1998.

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

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