# 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 the end-to-end message communication constraints are satisfied. We have 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

## 1. INTRODUCTION

Embedded real-time systems have to be designed such that they implement correctly the required functionality. In addition, they have to fulfill a wide range of competing constraints: development cost, unit cost, reliability, security, size, performance, power consumption, flexibility, time-to-market, maintainability, correctness, safety, etc. Very important for the correct functioning of such systems are their timing constraints: *"the correctness of the system behavior depends not only on the logical results of the computations, but also on the physical instant at which these results are produced*" [13].

Real-time systems have been classified as *hard* real-time and *soft* real-time systems [13]. Basically, hard real-time systems are systems where failing to meet a timing constraint can potentially have catastrophic consequences. For example, a brake-by-wire system in a car failing to react within a given time interval can result in a fatal accident. On the other hand, a multimedia system, which is a soft-real time system, can, under certain circumstances, tolerate a certain amount of delays resulting maybe in a patchier picture, without serious consequences besides some possible inconvenience to the user.

The techniques presented in this paper are aimed towards hard-real time systems that implement safety-critical applications where timing constraints are of utmost importance to the correct behavior of the application.

Many such applications, following physical, modularity or safety constraints, are implemented using *distributed architectures*. An increasing number of real-time applications are today implemented using distributed architectures consisting of interconnected clusters of processors (Figure 1). Each such cluster has its own communication protocol and two clusters communicate via a *gateway,* a node connected to both of them [21, 18]. This type of architectures is used in several application areas: vehicles, factory systems, networks on chip, etc.

Considering, for example, the automotive industry, the way functionality has been distributed on an architecture has evolved over time. Initially, distributed real-time systems were implemented using architectures where each node is dedicated to the implementation of a single function or class of functions, allowing the system integrators to purchase nodes implementing required functions from different vendors, and to integrate them into their system [7]. There are several problems related to this restricted mapping of functionality:

- The number of such nodes in the architecture has exploded, reaching, for example, more than 100 in a high-end car, incurring heavy cost and performance penalties.
- The resulting solutions are sub-optimal in many aspects, and do not use the available resources efficiently in order to reduce costs. For example, it is not possible to move a function from one node to another node where there are enough available resources (e.g., memory, computation power).
- Emerging functionality, such as brake-by-wire in the automotive industry, is inherently distributed, and achieving an efficient fault-tolerant implementation is very difficult in the current setting.

This has created a huge pressure to reduce the number of nodes by integrating several functions in one node and, at the same time, certain functionality has been distributed over several nodes (see

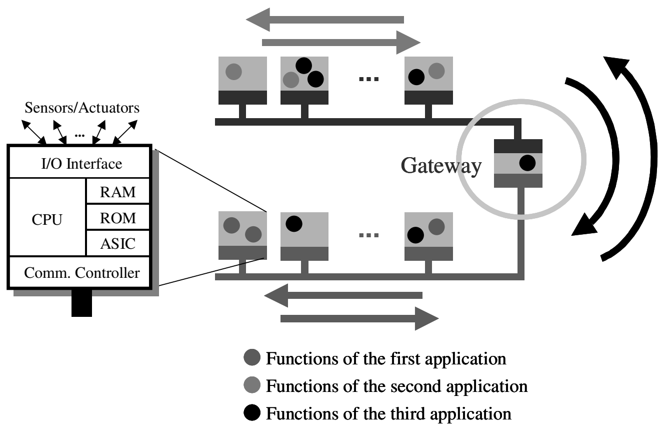
Sensors/Actuators
...
I/O Interface
CPU
RAM
ROM
ASIC
Comm. Controller
Gateway
...
...
● Functions of the first application
● Functions of the second application
● Functions of the third application

**Figure 1. Distributed Safety-Critical Applications**

Figure 1). Although an application is typically distributed over one single network, we begin to see applications that are distributed across several networks. For example, in Figure 1, the third application, represented as black dots, is distributed over two networks.

This trend is driven by the need to further reduce costs, improve resource usage, but also by application constraints like having to be physically close to particular sensors and actuators. Moreover, not only are these applications distributed across networks, but their functions can exchange critical information through the gateway nodes.

Due to the complexity of embedded systems, hardware/software cosynthesis environments are developed to assist the designer in finding the most cost effective solution that, at the same time, meets the design requirements [45].

Preemptive scheduling of independent processes with static priorities running on single-processor architectures has its roots in the work of Liu and Layland [19]. The approach has been later extended to accommodate more general computational models and has also been applied to distributed systems [39]. The reader is referred to [1, 3, 36] for surveys on this topic. Static cyclic scheduling of a set of data dependent software processes on a multiprocessor architecture has also been intensively researched [13, 44].

In [17] an earlier deadline first strategy is used for non-preemptive scheduling of processes with possible data dependencies. Preemptive and non-preemptive static scheduling are combined in the cosynthesis environment described in [5, 6]. In many of the previous scheduling approaches researchers have assumed that processes are scheduled independently. However, this is not the case in reality, where process sets can exhibit both data and control dependencies. Moreover, knowledge about these dependencies can be used in order to improve the accuracy of schedulability analyses and the quality of produced schedules. One way of dealing with data dependencies between processes with static priority based scheduling has been indirectly addressed by the extensions proposed for the schedulability analysis of distributed systems through the use of the *release jitter* [39]. Release jitter is the worst case delay between the arrival of a process and its release (when it is placed in the run-queue for the processor) and can include the communication delay due to the transmission of a message on the communication channel.

In [37] and [45] time offset relationships and phases, respectively, are used in order to model data dependencies. *Offset* and *phase* are similar concepts that express the existence of a fixed interval in time between the arrivals of sets of processes. The authors show that by introducing such concepts into the computational model, the pessimism of the analysis is significantly reduced when bounding the time behaviour of the system. The concept of dynamic offsets has been later introduced in [23] and used to model data dependencies [24].

Currently, more and more real-time systems are used in physically distributed environments and have to be implemented on distributed architectures in order to meet reliability, functional, and performance constraints.

Researchers have often ignored or very much simplified the communication infrastructure. One typical approach is to consider communications as processes with a given execution time (depending on the amount of information exchanged) and to schedule them as any other process, without considering issues like communication protocol, bus arbitration, packaging of messages, clock synchronization, etc. [45].

Many efforts dedicated to the synthesis of communication infrastructure parameters do not consider hard real-time constraints and system level scheduling aspects [10, 42]. We have to mention here some results obtained in extending real-time schedulability analysis so that network communication aspects can be handled. In [38], for example, the controller area network (CAN) protocol is investigated while the work reported in [39] deals with a simple time-division multiple access (TDMA) protocol.

There are two basic approaches for handling tasks in real-time applications [13]. 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 TT and ET approaches [2, 13, 43]. An interesting comparison, from a more industrial, in particular automotive, perspective, can be found in [20]. The conclusion there is that one has to choose the right approach depending on the characteristics 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 processes being TT and others ET.

In [25] we have addressed design problems for systems where the TT and ET activities *share the same* processor and TT/ET bus [11]. A fundamentally different architectural approach to heterogeneous TT/ET systems is that of heterogeneous multi-clusters, where each 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 a TDMA protocol such as, for example, the time-triggered protocol (TTP) [40].
- On *event-triggered clusters* (ETC) the processes are scheduled according to a priority based preemptive approach, while messages are transmitted using the priority-based CAN bus [4].

In this context, in [28] we have proposed an approach to schedulability analysis for multi-cluster distributed embedded systems. Starting from such an analysis, in this paper we address 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 [14, 33, 35]. 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, 65 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 [33, 35] CAN frames are created based on the properties of the messages, while in [14] 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 4.2 shows, the issue of frame packing cannot be addressed separately for each type of cluster, since the inter-cluster communication creates a circular dependency.

## 1.1 Contributions

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 a frame configuration that would produce a schedulable system.

The contributions of this paper are:

- We have addressed the issue of frame-packing in the context of multi-cluster architectures consisting of time-triggered clusters and event-triggered clusters.
- We have developed two new optimization heuristics that use the schedulability analysis as a driver towards a frame configuration that leads to a schedulable system.
- The schedulabiltiy of an application mapped on a multi-cluster system cannot be addressed separately for each type of cluster. Hence, we have proposed a multi-cluster scheduling algorithm that handles the circular dependency due to the inter-cluster communication.
- We have updated our schedulability analysis presented in [28] to account for the frame packing.

The paper is organized in six 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 frame-packing optimization strategy, driven by the analysis presented in Section 4.2. 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. Let $\mathcal{N}_T$ ($\mathcal{N}_E$) be the set of nodes on the TTC (ETC). Every *node* $N_i \in \mathcal{N}_T \cup \mathcal{N}_E$ includes a communication controller and a CPU, along with other components. 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 4.

Communication between the nodes on a TTC is based on the TTP [40]. 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.

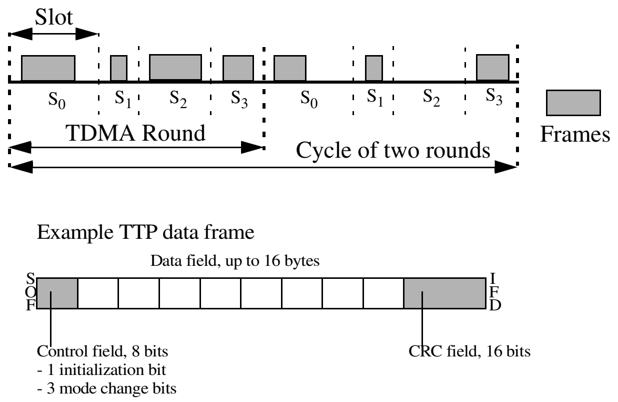

**Figure 2. Time-Triggered Protocol**

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 2) consists of the following fields: start of frame bit (SOF), control field, a data field of up to 16 bytes containing one or more messages, and a cyclic redundancy check (CRC) field. Frames are delimited by the inter-frame delimiter (IDF, 3 bits).

For example, the data efficiency of 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-bit frame, without considering the details of a

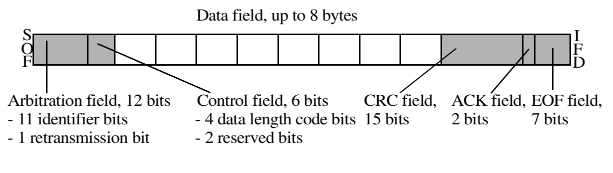

**Figure 3. Controller Area Network Data Frame (CAN 2.0A)**

particular physical layer). 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. In general, the protocol efficiency is in the range of 60–80% [41].

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 2.0A, 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 2. 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).

In this case, for a frame that carries 8 bytes of application data, we will have an efficiency of 47.4% [22]. The typical CAN protocol efficiency is in the range of 25–35% [41].

## 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 4 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 [26], while the infrastructure needed for communications on an ETC has been detailed in [38].

Let us consider the example in Figure 4, where we have an application consisting of four processes and four messages 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 4). $P_1$, 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 the 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 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 4 as a grey rectangle labeled "$m_3$") 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

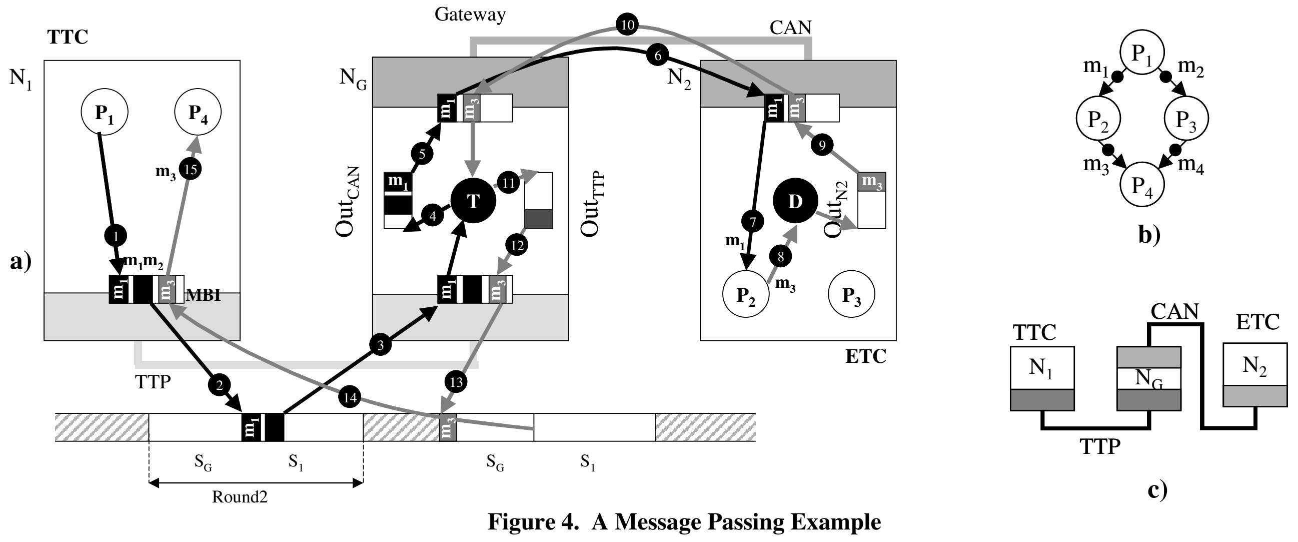

**Figure 4. A Message Passing Example**

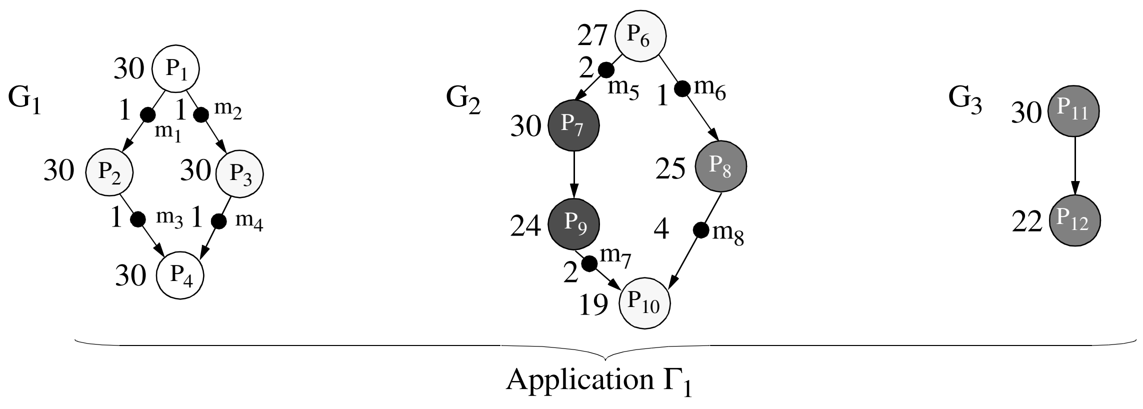

**Figure 5. Application Model**

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 4.2, 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

There is a lot of research in the area of system modeling and specification, and an impressive number of representations have been proposed. An overview, classification and comparison of different design representations and modeling approaches is given in [8, 15].

Researchers have used, for example, *dataflow process networks* (also called *task graphs*, or *process graphs*) [16] to describe interacting processes, and have represented them using directed acyclic graphs, where a node is a process and the directed arcs are dependencies between processes.

In this paper, we model an application $\Gamma$ as a set of process graphs $G_i \in \Gamma$ (see Figure 5). Nodes in the graph represent processes and arcs represent dependency between the connected processes. A *process* is a sequence of computations (corresponding to several building blocks in a programming language) which starts when all its inputs are available. When it finishes executing, the process produces its output values. Processes can be pre-emptable or non pre-emptable. *Non pre-emptable* processes are processes that cannot be interrupted during their execution, and are mapped on the TTC. *Pre-emptable* processes can be can be interrupted during their execution, and are mapped on the ETC. For example, a higher priority process has to be activated to service an event, in this case, the lower priority process will be temporary pre-empted until the higher priority process finishes its execution.

A process graph is polar, which means that there are two nodes, called source and sink, that conventionally represent the first and last process. If needed, these nodes are introduced as dummy processes so that all other nodes in the graph are successors of the source and predecessors of the sink, respectively.

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 performed 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 5).

Potential communication between processes in different applications is not part of the model. Technically, such a communication is implemented by the kernels based on asynchronous non-blocking send and receive primitives. Such messages are considered non-critical and are not affected by real-time constraints. Therefore, communications of this nature will not be addressed in this paper.

Each process $P_i$ is mapped on a processor $\mathcal{M}(P_i)$ (mapping represented by hashing in Figure 5), and has a worst case execution time $C_i$ on that processor (depicted to the left of each node). The designer can provide manually such worst-case times, or tools can be used in order to determine the worst-case execution time of a piece of code on a given processor [32].

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 $D_{G_i}$ is imposed on each process graph $G_i$. Deadlines can also be placed locally on processes. Release times of some processes as well as multiple deadlines can be easily modelled by inserting dummy nodes between certain processes and the source or the sink node, respectively. These dummy nodes represent processes with a certain execution time but which are not allocated to any processing element.

## 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" (captured by Equation (1)), 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).

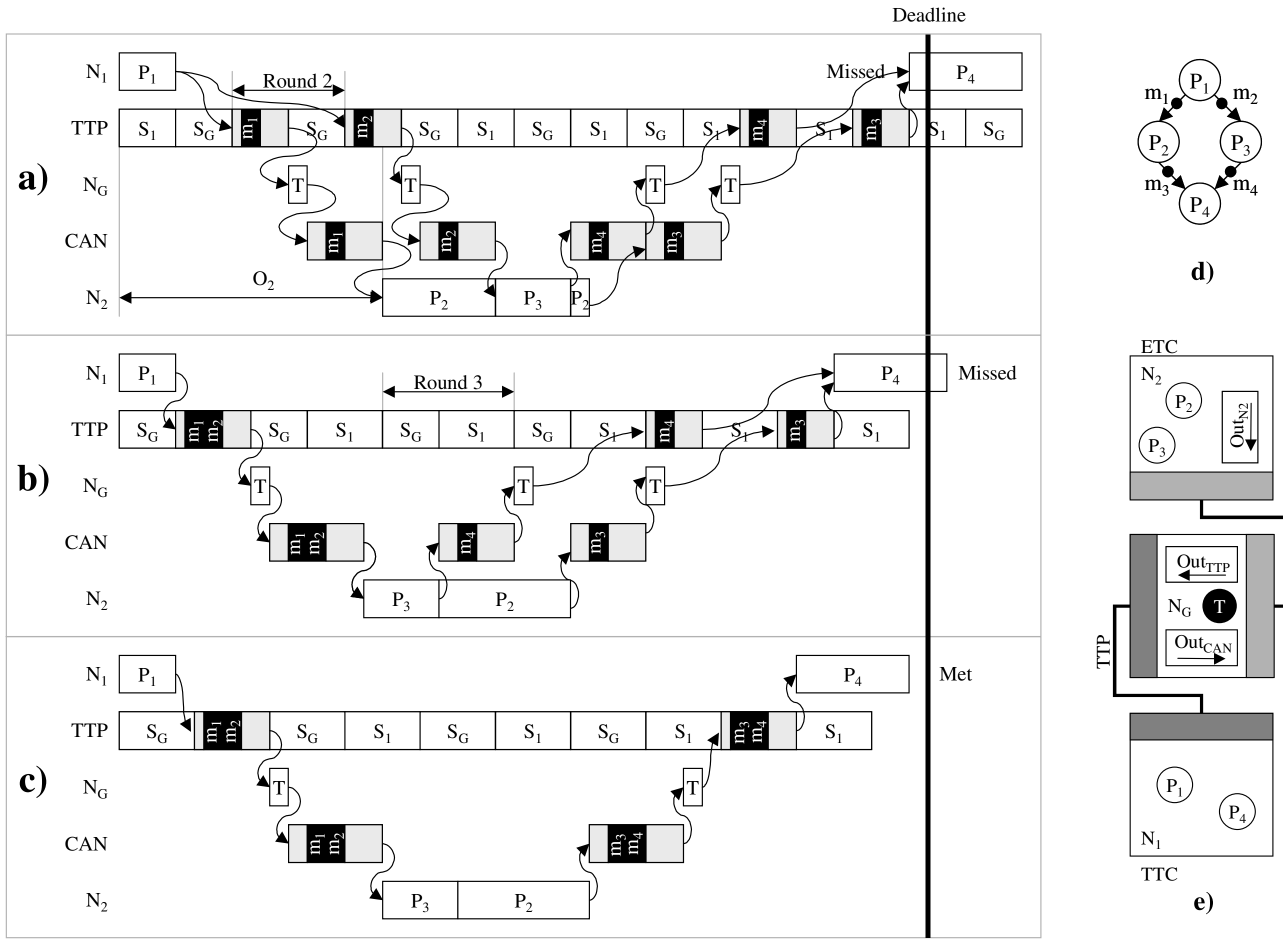

**Figure 6. Frame-Packing Optimization Example**

Let us consider the motivational example in Figure 6, where we have the process graph from Figure 6d mapped on the two-cluster system from Figure 6e: $P_1$ and $P_4$ are mapped on node $N_1$ from the TTC, while $P_2$ and $P_3$ are mapped on $N_2$ from ETC. The data filed of the frames is represented with a black rectangle, while the other frame fields are depicted in with a grey color. We consider a physical implementation of the buses such that the frames will take the time indicated in the figure by the length of their rectangles.

We are interested to find a frame configuration such that the application is schedulable.

In the system configuration of Figure 6a 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 ($S_G$). Process $P_3$ has a higher priority than process $P_2$, hence $P_2$ will be interrupted by $P_3$ when it receives message $m_2$. In such a setting, $P_4$ will miss its deadline, which is depicted as a thick vertical line in Figure 6. Changing the frame configuration as in Figure 6b, 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 worst-case response time of the process graph, which is still larger than the deadline. In Figure 6c, 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 worst-case response time of the application is further reduced, which means that the deadline is met, thus the system is schedulable.

However, packing more messages will not necessarily reduce the worst-case response times further, as it might increase too much the worst-case response times of messages that have to wait for the frame to be assembled, like is the case with message $m_3$ in Figure 6c. We are interested to find a frame packing that leads to a schedulable system.

Several details related to the schedulability analysis were omitted from the discussion of the example. These details will be discussed in Section 5.

## 4. FRAME PACKING STRATEGY

The general multi-cluster optimization strategy is outlined in Figure 7. The MultiClusterConfiguration strategy has two steps:

1. In the first step, line 3, the application is partitioned on the TTC and ETC clusters, and processes are mapped to the nodes of the architecture using the PartitioningAndMapping function. The partitioning and mapping can be done with an optimization heuristic like the one presented in [31]. As part of the partitioning and mapping process, an initial frame configuration $\psi^0 = <\alpha^0, \pi^0, \beta^0, \sigma^0>$ is derived. Messages exchanged by processes partitioned to the TTC will be mapped to TTC frames, while messages exchanged on the ETC will be mapped to ETC frames. For each message sent from a TTC process to an ETC process, we create an additional message on the ETC, and we map this message to an ETC frame. The sequence $\sigma^0$ of slots for the TTC is decided by assigning in order nodes to the slots ($S_i = N_i$). One message is assigned per frame in the initial set $\beta^0$ of TTC frames. For the ETC, the frames in the set $\alpha^0$ initially hold each one single message, and we calculate the message priorities $\pi^0$ based on the deadlines of the receiver processes.

```
MultiClusterConfiguration(Γ)
1  -- determine an initial partitioning and mapping $\mathcal{M}$,
2  -- and an initial frame configuration $\psi^0$
3  $<\mathcal{M}, \psi^0>$ = PartitioningAndMapping(Γ)
4  -- the frame packing optimization algorithm
5  $\psi$ = FramePackingOptimization(Γ, $\mathcal{M}$, $\psi^0$)
6  -- test if the resulted configuration leads to a schedulable application
7  if MultiClusterScheduling(Γ, $\mathcal{M}$, $\psi$) returns schedulable then
8      return $\mathcal{M}$, $\psi$
9  else
10     return unschedulable
11 endif
end MultiClusterConfiguration
```

**Figure 7. The General Frame Packing Strategy**

2. The frame packing optimization, is performed as the second step (line 7 in Figure 7). The FramePackingOptimization function receives as input the application $\Gamma$, the mapping $\mathcal{M}$ of processes to resources and the initial frame configuration $\psi^0$, and it produces as output the optimized frame packing configuration $\psi$. Such an optimization problem is NP complete [34], thus obtaining the optimal solution is not feasible. In this paper, we propose two frame packing optimization strategies, one based on a simulated annealing approach, presented in Section 4.3, while the other, outlined in Section 4.4, is based on a greedy heuristic that uses intelligently the problem-specific knowledge in order to explore the design space.

If after these steps the application is unschedulable, we conclude that no satisfactory implementation could be found with the available amount of resources.

Testing if the application $\Gamma$ is schedulable is done using the MultiClusterScheduling (MCS) algorithm (line 7 in Figure 7). The multi-cluster scheduling algorithm, presented in Section 4.2, takes as input an application $\Gamma$, a mapping $\mathcal{M}$ and an initial frame configuration $\psi^0$, builds the TT schedule tables, sets the ET priorities for processes, and provides the global analysis.

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 *response time analysis*, presented in the next section.

In order to drive our frame-packing optimization algorithms towards schedulable solutions, we characterize a given frame packing configuration using the degree of schedulability of the application. The *degree of schedulability* [27] is calculated as:

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

where $n$ is the number of processes in the application, $r_i$ is the worst-case response time of a process $P_i$, and $D_i$ its deadline. The worst-case response times are calculated by the MultiClusterScheduling algorithm using the response time analysis presented in the next section.

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 worst-case response times of the processes, so the application can potentially be implemented on a cheaper hardware architecture (with slower processors and/or buses). Improving the degree of schedulability can also lead to an improvement in the quality of control for control applications.

## 4.1 Schedulability Analysis for the ETC

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 [23, 37, 45], and has been also extended to consider the CAN protocol [38]. 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 $P_i$. Such an offset is, for example, $O_2$ in Figure 6a, as process $P_2$ cannot start before receiving $m_1$. The same is true for messages, their offset indicating the earliest possible transmission time.

In [28] 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 [28], showing how it can be extended to handle frames. The analysis presented in this section works under the following assumptions:

- All the processes belonging a process graph $G$ have the same period $T_G$. However, process graphs can have different periods.
- The offsets are *static* (as opposed to *dynamic* [23]), and are smaller than the period.
- The deadlines are arbitrary can be higher than the period.

In addition, our MultiClusterScheduling approach presented in Section 4.2 assumes that the nodes of an ETC are synchronized. On the TTC, the time-triggered protocol offers clock synchronization. If the clocks are not synchronized on the ETC, overheads will have to be added to account for the drifting of the clocks.

In this context, the worst-case response time of a process $P_i$ on the ETC[1] is [37]:

$$r_i = \max_{q = 0, 1, 2\ldots}\left(\max_{\forall P_j \in G}\left(w_i(q) + O_j + J_j - T_G\left(q + \left\lceil \frac{O_j + J_j - O_i - J_i}{T_G} \right\rceil\right) - O_i\right)\right) \quad (2)$$

where $T_G$ the period of the process graph $G$, $O_i$ and $O_j$ are offsets of processes $P_i$ and $P_j$, respectively, and $J_i$ and $J_j$ are the jitters of $P_i$ and $P_j$. The jitter is the worst-case delay between the arrival of a process and its release. In Equation (2), $q$ is the number of busy periods being examined, and $w_i(q)$ is the width of the level-$i$ busy period starting at time $qT_G$ [37]:

$$w_i(q) = B_i + (q+1)C_i + I_i \cdot \quad (3)$$

In the previous equation, the blocking term $B_i$ represents interference from lower priority processes that are in their critical section and cannot be interrupted, and $C_i$ represents the worst-case execution time of process $P_i$. The last term captures the interference $I_i$ from higher priority processes. The reader is directed to [37] for the details of the interference calculation.

Although this analysis is exact (both necessary and sufficient), it is computationally infeasible to evaluate. Hence, [37] proposes a feasible[2] but not exact analysis (sufficient but not necessary) for solving Equation (2). Our MultiClusterScheduling algorithm presented in Section 4.2 uses the feasible analysis provided in [37] for deriving the worst-case response time of a process $P_i$.

Regarding the worst-case response time of messages, we have extended the analysis from [38] and applied it for frames on the CAN bus:

$$r_f = \max_{q = 0, 1, 2\ldots}(J_f + W_f(q) + C_f) \,. \quad (4)$$

In the previous equation $J_f$ 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 $S(m)$ which sends message $m$ packed into frame $f$:

$$J_f = \max_{\forall m \in f}(r_{S(m)}) \,. \quad (5)$$

In Equation (4), $W_f$ is the *worst-case queuing delay* experienced by $f$ at the communication controller, and is calculated as:

$$W_f(q) = w_f(q) - qT_f \quad (6)$$

where $q$ is the number of busy periods being examined, and $w_f(q)$ is the width of the level-$f$ busy period starting at time $qT_f$.

Moreover, in Equation (4), $C_f$ 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.

The worst-case response time of message $m$ packed into a frame $f$ can be determined by observing that $r_m = r_f$.

The worst-case queueing delay for a frame ($W_f$ in Equation (4)) is calculated differently for each type of queue:

[1] Processes mapped on the ETC are pre-emptable, and are scheduled using fixed-priority pre-emptive scheduling.

[2] The implementation of this feasible analysis is available at: ftp://ftp.cs.york.ac.uk/pub/realtime/programs/src/offsets/

1. The output queue of an 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 $N_i$. An example of such a frame is the one containing message $m_3$ in Figure 6a, which is sent by process $P_2$ from the ETC node $N_2$ to the gateway node $N_G$, and has to wait in the $Out_{N_2}$ queue.
2. The TTP-to-CAN queue of the gateway node, in which case $W_f^{CAN}$ is the worst-case time a frame $f$ has to spend in the $Out_{CAN}$ queue of node $N_G$. In Figure 6a, the frame containing $m_1$ is sent from the TTC node $N_1$ to the ETC node $N_2$, and has to wait in the $Out_{CAN}$ queue of gateway node $N_G$ before it is transmitted on the CAN bus.
3. The CAN-to-TTP queue of the gateway node, where $W_f^{TTP}$ captures the time $f$ has to spend in the $Out_{TTP}$ queue node $N_G$. Such a situation is present in Figure 6a, where the frame with $m_3$ is sent from the ETC node $N_2$ to the TTC node $N_1$ through the gateway node $N_G$ where it has to wait in the $Out_{TTP}$ queue before it is transmitted on the TTP bus, in the $S_G$ slot of node $N_G$.

On the TTC, the synchronization between processes and the TDMA bus configuration is solved through the proper synthesis of schedule tables, hence no output queues are needed. The frames sent from a TTC node to another TTC node are taken into account when determining the offsets, and 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 Worst-case queuing delays in the $Out_{N_i}$ and $Out_{CAN}$ queues*

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:

$$w_f(q) = B_f + \sum_{\forall f_j \in hp(f)} \left\lceil \frac{w_f(q) + J_j}{T_j} \right\rceil C_j . \tag{7}$$

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 $f_j \in hp(f)$ 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 frame $f_k \in lp(f)$ to be transmitted to its destination is:

$$B_f = \max_{\forall f_k \in lp(f)} (C_k) . \tag{8}$$

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 Worst-case queuing delay in the $Out_{TTP}$ queue*

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 period $T_{TDMA}$ with which this slot $S_G$ is circulating on the bus [30]:

$$w_f^{TTP}(q) = B_f + \left\lfloor \frac{(q+1)s_f + I_f(w_f(q))}{S_G} \right\rfloor T_{TDMA} \tag{9}$$

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

$$I_f(w) = \sum_{\forall f_j \in hp(f)} \left\lceil \frac{w_f + J_j}{T_j} \right\rceil s_j \tag{10}$$

where the frame jitter $J_j$ is given by Equation (5).

The blocking term $B_f$ is the time interval in which $f$ cannot be transmitted because the slot $S_G$ of the TDMA round has not arrived yet. In the worst case (i.e., the frame $f$ has just missed the slot $S_G$), the frame has to wait an entire round $T_{TDMA}$ for the slot $S_G$ in the next TDMA round.

## 4.2 Multi-Cluster Scheduling

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 worst-case response times of the processes on the ETC, which on their turn influence the schedule table construction on the TTC. In Figure 6b 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 delay $P_2$ (which would not be the case if $m_2$ sent to $P_3$ would be scheduled in round 3, for example) 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 worst-case 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 8 receives as input the application $\Gamma$, the frame configuration $\psi$, and produces the offsets $\phi$ and worst-case response times $\rho$.

```
MultiClusterScheduling($\Gamma$, $\mathcal{M}$, $\psi$)
-- determines the set of offsets $\phi$ and worst-case response times $\rho$
1  for each $O_i \in \phi$ do $O_i = 0$ end for -- initially all offsets are zero
2  -- determine initial values for the worst-case response times
3  -- according to the analysis in Section 4.1
4  $\rho$ = ResponseTimeAnalysis($\Gamma$, $\mathcal{M}$, $\psi$, $\phi$)
5  -- determine new values for the offsets, based on the response times $\rho$
6  $\phi^{new}$ = ListScheduling($\Gamma$, $\mathcal{M}$, $\psi$, $\rho$)
7  $\delta_\Gamma = \infty$ -- consider the system unschedulable at first
8  repeat -- iteratively improve the degree of schedulability $\delta_\Gamma$
9    for each $O_i^{new} \in \phi^{new}$ do -- for each newly calculated offset
10     $O_i^{old} = \phi.O_i$; $\phi.O_i = \phi^{new}.O_i^{new}$ -- set the new offset, remember old
11     $\rho^{new}$ = ResponseTimeAnalysis($\Gamma$, $\mathcal{M}$, $\psi$, $\phi$)
12     $\delta_\Gamma^{new}$ = SchedulabilityDegree($\Gamma$, $\rho$)
13     if $\delta_\Gamma^{new} < \delta_\Gamma$ then -- the schedulability has improved
14       -- offsets are recalculated using $\rho^{new}$
15       $\phi^{new}$ = ListScheduling($\Gamma$, $\mathcal{M}$, $\psi$, $\rho^{new}$)
16       break -- exit the for-each loop
17     else -- the schedulability has not improved
18       $\phi.O_i = O_i^{old}$-- restore the old offset
19   end for
20 until $\delta_\Gamma$ has not changed or a limit is reached
21 return $\rho$, $\phi$, $\delta_\Gamma$
end MultiClusterScheduling
```

**Figure 8. The MultiClusterScheduling Algorithm**

The algorithm sets initially all the offsets to 0 (line 1). Then, the worst-case response times are calculated using the ResponseTime-Analysis function (line 4) using the feasible analysis provided in [37]. The fixed-point iterations that calculate the response times at line 3 will converge if processor and bus loads are smaller than 100% [37]. Based on these worst-case response times, we determine new values $\phi^{new}$ for the offsets using a list scheduling algorithm (line 6).

The multi-cluster scheduling algorithm loops until the degree of schedulability $\delta_\Gamma$ of the application $\Gamma$ cannot be further reduced (lines 8–20). In each loop iteration, we select a new offset from the set of $\phi^{new}$ offsets (line 10), and run the response time analysis (line 11) to see if the degree of schedulability has improved (line 12). If $\delta_\Gamma$ has not improved, we continue with the next offset in $\phi^{new}$.

When a new offset $O_i^{new}$ leads to an has improved $\delta_\Gamma$ we exit the for-each loop 9–19 that examines offsets from $\phi^{new}$. The loop iteration 8–20 continues with a new set of offsets, determined by ListScheduling at line 15, based on the worst-case response times $\rho^{new}$ corresponding to the previously accepted offset.

In the multi-cluster scheduling algorithm, the calculation of offsets is performed by the list scheduling algorithm presented in Figure 9. In each iteration, the algorithm visits the processes and messages in the ReadyList. A process or a message in the application is placed in the ReadyList if all its predecessors have been already scheduled. The list is ordered based on the priorities presented in [9]. The algorithm terminats when all processes and messages have been visited.

In each loop iteration, the algorithm calculates the earliest time moment *offset* when the process or message $node_i$ can start (lines 5–7). There are four situations:

1. The visited node is an ET message. The message $m_i$ is packed into its frame $f$ (line 9), and the offset $O_f$ of the frame is updated. The frame can only be transmitted after all the sender processes that pack messages in this frame have finished executing. The offset of message $m_i$ packed to frame $f$ is equal to the frame offset $O_f$.

2. The node is a TT message. In this case, when the frame is ready for transmission, it is scheduled using the ScheduleTTFrame function (presented in Figure 10), which returns the *round* and the *slot* where the frame has been placed (line 6 in Figure 9). In Figure 10, the round immediately following *offset* is the initial candidate to be considered (line 2). However, it can be too late to catch the allocated slot, in which case the next round is considered (line 4). For this candidate round, we have to check if the slot is not occupied by another frame. If so, the communication has to be delayed for another round (line 7). Once a frame has been scheduled, we can determine the offsets and worst-

```
ListScheduling($\Gamma$, $\mathcal{M}$, $\psi$, $\rho$) -- determines the set of offsets $\phi$
1  ReadyList = source nodes of all process graphs in the application
2  while ReadyList $\neq \emptyset$ do
3    $node_i$ = Head(ReadyList)
4    offset = 0 -- determine the earliest time when an activity can start
5    for each direct predecessor $node_j$ of $node_i$ do
6      offset = max(offset, $O_j + r_j$)
7    end for
8    if $node_i$ is a message $m_i$ then
9      PackFrame($m_i$, $f$) -- pack each ready message $m$ into its frame $f$
10     $O_f$ = max($O_f$, offset) -- update the frame offset
11     if $f$ is complete then -- the frame is complete for transmission
12       if $f \in \alpha$ then -- $f$ is an ET frame
13         -- the offset of messages is equal to the frame offset
14         for each $m_j \in f$ do $O_j = O_f$ end for
15       else -- $f$ is a TT frame
16         <round, slot> = ScheduleTTFrame($f$, offset, $\psi$)
17         -- set the TT message offsets based on the round and slot
18         for each $m_j \in f$ do $O_j = round * T_{TDMA} + O_{slot}$ end for
19     endif; endif
20   else -- $node_i$ is a process $P_i$
21     if $\mathcal{M}(P_i) \in \mathcal{N}_E$ then -- if process $P_i$ is mapped on the ETC
22       $O_i$ = offset -- the ETC process can start immediately
23     else -- process $P_i$ is mapped on the TTC
24       -- $P_i$ has to wait also for the processor $\mathcal{M}(P_i)$ to become available
25       $O_i$ = max(offset, ProcessorAvailable($\mathcal{M}(P_i)$))
26   end if; end if;
27   Update(ReadyList)
28 end while
29 return offsets $\phi$
end ListScheduling
```

**Figure 9. ListScheduling Algorithm**

```
ScheduleTTFrame (f, offset, $\psi$)
-- returns the slot and the round assigned to frame f
1  slot = the slot assigned to the node sending f -- the frame slot
2  round = offset / $T_{TDMA}$ -- the first round which could be a candidate
3  if offset – round * $T_{TDMA}$ > $O_{slot}$ then -- the slot in this is missed
4      round = round + 1 -- if yes, take the next round
5  end if
6  while slot is occupied do
7      round = round + 1
8  end while
9  return round, slot
end ScheduleTTFrame
```

**Figure 10. Frame Scheduling on the TTC**

case response times (Figure 9, line 18). For all the messages in the frame the offset is equal to the start of the slot in the TDMA round, and the worst-case response time is the slot length.

3. The algorithm visits a process $P_i$ mapped on an ETC node. A process on the ETC can start as soon as its predecessors have finished and its inputs have arrived, hence $O_i$ = *offset* (line 22). However, $P_i$ might experience interference from higher priority processes.

4. Process $P_i$ is mapped on a TTC node. In this case, besides waiting for the predecessors to finish executing, $P_i$ will also have to wait for its processor $\mathcal{M}(P_i)$ to become available (line 25). The earliest time when the processor is available is returned by the ProcessorAvailable function.

Let us now turn the attention back to the multi-cluster scheduling algorithm in Figure 8. The algorithm stops when the $\delta_\Gamma$ of the application $\Gamma$ is no longer improved, or when a limit imposed on the number of iterations has been reached. Since in a loop iteration we do not accept a solution with a larger $\delta_\Gamma$, the algorithm will terminate when in a loop iteration we are no longer able to improve $\delta_\Gamma$ by modifying the offsets. For the experimental results in Section 5 we have imposed a *limit* of ten iterations (line 20, Figure 8).

## 4.3 Frame Packing with Simulated Annealing

The first algorithm we have developed is based on a simulated annealing (SA) strategy [34], and is presented in Figure 11. The algorithm takes as input the application $\Gamma$, a mapping $\mathcal{M}$ and an initial frame configuration $\psi^0$, and determines the frame configuration $\psi$ which leads to the smallest degree of schedulability $\delta_\Gamma$ (the smaller the value, the more schedulable the system).

Determining a frame configuration $\psi$ means finding the set of ETC frames $\alpha$ and their relative priorities $\pi$, and the set of TTC frames $\beta$, including the sequence $\sigma$ of slots in a TDMA round.

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 (lines 9–10 of the algorithm in Figure 11). 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 (lines 12–13).

In Figure 11 we give a short description of this algorithm. An essential component of the algorithm is the generation of a new solution $\psi_{new}$ starting from the current one $\psi_{current}$. The neighbors of the current solution $\psi_{current}$ are obtained by performing transformations (called moves) on the current frame configuration $\psi_{current}$ (line 8). 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 $\alpha$;
- swapping two slots in the sequence $\sigma$ of slots in a TDMA round.

For the implementation of this algorithm, the parameters *TI* (initial temperature), *TL* (temperature length), $\varepsilon$ (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 $\varepsilon$ that will guarantee the finding of good quality solutions in a short time.

We performed long runs of up to 48 hours with the SA algorithm, for ten synthetic process graphs (two for each graph dimension of 80, 160, 240 320, 400, see Section 5) 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 $\varepsilon$ is 0.98. The algorithm stops if for three consecutive temperatures no new solution has been accepted.

```
SimulatedAnnealing($\Gamma$, $\mathcal{M}$, $\psi^0$)
1  -- given an application $\Gamma$ finds out if it is schedulable and produces
2  -- the configuration $\psi$ = <$\alpha$, $\pi$, $\beta$, $\sigma$> leading to the smallest $\delta_\Gamma$
3  -- initial frame configuration
4  $\psi_{current}$ = $\psi^0$
5  temperature = initial temperature TI
6  repeat
7      for i = 1 to temperature length TL do
8          generate randomly a neighboring solution $\psi_{new}$ of $\psi_{current}$
9          $\delta$ = MultiClusterScheduling($\Gamma$, $\mathcal{M}$, $\psi_{new}$) -
               MultiClusterScheduling($\Gamma$, $\mathcal{M}$, $\psi_{current}$)
10         if $\delta$ < 0 then $\psi_{current}$ = $\psi_{new}$
11         else
12             generate q = Random (0, 1)
13             if q < $e^{-\delta / temperature}$ then $\psi_{current}$ = $\psi_{new}$ end if
14         end if
15     end for
16     temperature = $\varepsilon$ * temperature
17 until stopping criterion is met
18 return SchedulabilityTest($\Gamma$, $\mathcal{M}$, $\psi_{best}$), solution $\psi_{best}$
       corresponding to the best degree of schedulablity $\delta_\Gamma$
end SimulatedAnnealing
```

**Figure 11. The Simulated Annealing Algorithm**

## 4.4 Frame Packing Greedy Heuristic

The OptimizeFramePacking greedy heuristic (Figure 12) 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 worst-case response times known so far and the application structure (i.e., the dependencies in the process graph). Thus, our greedy heuristic outlined in Figure 12, 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 algorithm tries to determine, using the for-each loops in Figure 12 the best frame configuration. The algorithm starts from the initial frame configuration $\psi^0$, and progressively determines the best change to the current configuration. The quality of a frame configuration is measured using the MultiClusterScheduling algorithm, which calculates the degree of schedulability $\delta_\Gamma$ (line 13). Once a configuration parameter has been fixed in the outer loops it is used by the inner loops:

- Lines 10–15: The innermost loops determine the best size $S_\alpha$ for the currently investigated frame $f_\alpha$ in the ETC frame configuration $\alpha_{current}$. Thus, several frame sizes are tried (line 11), each with a size returned by 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.
- Lines 9–16: This loop determines the best frame configuration $\alpha$. This means deciding on how many frames to include in $\alpha$ (line 9), and which are the best sizes for them. In $\alpha$ there can be any number of frames, from one single frame to $n_\alpha$ frames (in which case each frame carries one single message). Once a configuration $\alpha_{best}$ for the ETC, minimizing $\delta_\Gamma$, has been determined (saved in line 16), the algorithm looks for the frame configuration $\beta$ which will further improve $\delta_\Gamma$.
- Lines 7–17: The best size for a frame $f_\beta$ is determined similarly to the size for a frame $f_\alpha$.
- Lines 6–18: The best frame configuration $\beta_{best}$ is determined. For each frame configuration $\beta$ tried, the algorithm loops again through the innermost loops to see if there are better frame configurations $\alpha$ in the context of the current frame configuration $\beta_{current}$.

```
OptimizeFramePacking($\Gamma$, $\mathcal{M}$, $\psi^0$) -- produces the frame configuration $\psi$ leading to the smallest degree of schedulability $\delta_\Gamma$
1  $\pi^0$ = HOPA -- the initial priorities $\pi^0$ are updated using the HOPA heuristic
2  -- build the message lists ordered ascending on their offsets
3  $messages_\beta$ = ordered list of $n_\beta$ messages on the TTC; $messages_\alpha$ = ordered list of $n_\alpha$ messages on the ETC
4  for each $slot_i \in \sigma_{current}$ do for each $slot_j \in \sigma_{current} \wedge slot_i \neq slot_j$ do -- determine the best TTP slot sequence $\sigma$
5        Swap($slot_i$, $slot_j$) -- tentatively swap slots $slot_i$ with $slot_j$
6        for each $\beta_{current}$ with 1 to $n_\beta$ frames do -- determine the best frame packing configuration $\beta$ for the TTC
7           for each frame $f_\beta \in \beta_{current}$ do for each frame size $S_\beta \in$ RecomendedSizes($messages_\beta$) do -- determine the best frame size for $f_\beta$
8              $\beta_{current}.f_\beta.S = S_\beta$
9              for each $\alpha_{current}$ with 1 to $n_\alpha$ frames do -- determine the best frame packing configuration $\alpha$ for the ETC
10                for each frame $f_\alpha \in \alpha_{current}$ do for each frame size $S_\alpha \in$ RecomendedSizes($messages_\alpha$) do -- determine the best frame size for $f_\alpha$
11                   $\alpha_{current}.f_\alpha.S = S_\alpha$
12                   $\psi_{current}$ = <$\alpha_{current}$, $\pi^0$, $\beta_{current}$, $\sigma_{current}$>; PackMessages($\psi_{current}$, $messages_\beta \cup messages_\alpha$)
13                   $\delta_\Gamma$ = MultiClusterScheduling($\Gamma$, $\mathcal{M}$, $\psi_{current}$)
14                   if $\delta_\Gamma(\psi_{current})$ is best so far then $\psi_{best} = \psi_{current}$ end if -- remember the best configuration so far
15                end for; end for; if $\exists\ \psi_{best}$ then $\alpha_{current}.f_\alpha.S = \alpha_{best}.f_\alpha.S$ end if -- remember the best frame size for $f_\alpha$
16             end for; if $\exists\ \psi_{best}$ then $\alpha_{current} = \alpha_{best}$ end if -- remember the best frame packing configuration $\alpha$
17          end for; end for; if $\exists\ \psi_{best}$ then $\beta_{current}.f_\beta.S = \beta_{best}.f_\beta.S$ end if -- remember the best frame size for $f_\beta$
18       end for; if $\exists\ \psi_{best}$ then $\beta_{current} = \beta_{best}$ end if -- remember the best frame packing configuration $\beta$
19 end for; if $\exists\ \psi_{best}$ then $\sigma_{current}.slot_i = \sigma_{current}.slot$ end if; -- remember the best slot sequence $\sigma$; end for
20 return SchedulabilityTest($\Gamma$, $\mathcal{M}$, $\psi_{best}$), $\psi_{best}$
end OptimizeFramePacking
```

**Figure 12. The OptimizeFramePacking Algorithm**

- Lines 4–19: After a $\beta_{best}$ has been decided, the algorithm looks for a slot sequence $\sigma$, starting with the first slot and tries to find the node which, when transmitting in this slot, will reduce $\delta_\Gamma$. Different slot sequences are tried by swapping two slots within the TDMA round (line 5).

For the initial message priorities $\pi^0$ (initially, there is one message per frame) we use the "heuristic optimized priority assignment" (HOPA) approach in [12], where priorities 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 (line 1). The ETC message priorities set at the beginning of the algorithm are not changed by our greedy optimization loops. The priority of a frame $f_\alpha \in \alpha$ is given by the message $m \in f_\alpha$ with the highest priority.

The algorithm continues in this fashion, recording the best ever $\psi_{best}$ configurations obtained, in terms of $\delta_\Gamma$, and thus the best solution ever is reported when the algorithm finishes.

## 5. 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. Forty processes were assigned to each node, resulting in applications of 80, 160, 240, 320 and 400 processes.

We generated both graphs with random structure and graphs based on more regular structures like trees and groups of chains. We generated a random structure graph deciding for each pair of two processes if they should be connected or not. Two processes in the graph were connected with a certain probability (between 0.05 and 0.15, depending on the graph dimension) on the condition that the dependency would not introduce a loop in the graph. The width of the tree-like structures was controlled by the maximum number of direct successors a process can have in the tree (from 2 to 6), while the graphs consisting of groups of chains had 2 to 12 parallel chains of processes. Furthermore, the regular structures were modified by adding a number of 3 to 30 random cross-connections.

The mapping of the applications to the architecture has been done using a simple heuristic that tries to balance the utilization of processors while minimizing communication. Execution times and message lengths were assigned randomly using both uniform and exponential distribution within the 10 to 100 ms, and 1 bit to 2 bytes ranges, respectively. 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 $\delta_\Gamma$ obtained from our OptimizeFramePacking (OFP) heuristic (Figure 12) with the near-optimal values obtained by SA (Figure 11). Obtaining solutions that have a higher degree of schedulability means obtaining tighter worst-case response times, increasing the chances of meeting the deadlines.

Table 1 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 8.

In Table 1 we have one row for each application dimension of 80 to 400 processes, and a header for each optimization algorithm considered. For each of the SF and OFP algorithms we have three columns in the table. In the first column, we present the average percentage deviation of the algorithm from the results obtained by SA. The percentage deviation is calculated according to the formula:

$$deviation = \frac{\delta_\Gamma^{approach} - \delta_\Gamma^{SA}}{\delta_\Gamma^{SA}} \cdot 100 . \qquad (11)$$

The second column presents the maximum percentage deviation from the SA result, and the third column presents the average execution time of the algorithm, in seconds. For the SA algorithm we present only its average execution times.

**Table 1. Evaluation of the Frame-Packing Optimzation Algorithms**

| No. of Processes | Straightforward solution (SP) | | | Optimize Frame Packing (OFP) | | | Simulated Annealing (SA) |
|---|---|---|---|---|---|---|---|
| | avgerage (%) | max (%) | time (sec.) | average (%) | max (%) | time (sec.) | time (sec.) |
| 80 | 2.42 | 17.89 | 0.09 | 0.40 | 1.59 | 4.35 | 235.95 |
| 160 | 16.10 | 42.28 | 0.22 | 2.28 | 8.32 | 12.09 | 732.40 |
| 240 | 40.49 | 126.4 | 0.54 | 6.59 | 21.80 | 49.62 | 2928.53 |
| 320 | 70.79 | 153.08 | 0.74 | 13.70 | 30.51 | 172.82 | 7585.34 |
| 400 | 97.37 | 244.31 | 0.95 | 31.62 | 95.42 | 248.30 | 22099.68 |

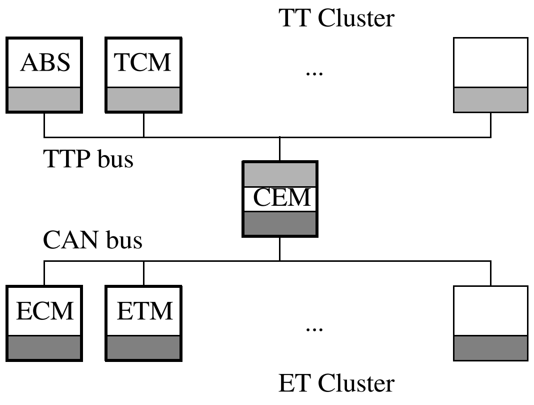

**Figure 13. Hardware Architecture for the Cruise Controller**

Table 1 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 under 100 seconds on average.

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.

## 5.1 The Vehicle Cruise Controller

A typical safety critical application with hard real-time constraints, is a vehicle cruise controller (CC). We have considered a CC system derived from a requirement specification provided by the industry. The CC delivers the following functionality: it maintains a constant speed for speeds over 35 Km/h and under 200 Km/h, offers an interface (buttons) to increase or decrease the reference speed, and is able to resume its operation at the previous reference speed. The CC operation is suspended when the driver presses the brake pedal.

The specification assumes that the CC will operate in an environment consisting of two clusters. There are four nodes which functionally interact with the CC system: the Anti-lock Braking System (ABS), the Transmission Control Module (TCM), the Engine Control Module (ECM), and the Electronic Throttle Module (ETM) (see Figure 13).

It has been decided to map the functionality (processes) of the CC over these four nodes. The ECM and ETM nodes have an 8-bit Motorola M68HC11 family CPU with 128 Kbytes of memory, while the ABS and TCM are equipped with a 16-bit Motorola M68HC12 CPU and 256 Kbytes of memory. The 16-bit CPUs are twice as fast than the 8-bit ones. The transmission speed of the communication channel is 256 Kbps and the frequency of the TTP controller was chosen to be 20 MHz.

We have modeled the specification of the CC system using a set of 32 processes and 17 messages as described in [29], where the mapping of processes to the nodes is also given. The period was chosen 250 ms, equal to the deadline.

In this setting, the straightforward approach SF produced an end-to-end worst-case 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.

This shows that the optimization heuristic proposed, driven by our schedulability analysis, is able to identify that frame packing configuration which increases the schedulability degree of an application, allowing the developers to reduce the implementation cost of a system.

## 6. 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 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. Although the SA takes a very long time to execute, it finds very good quality results, and thus can be used for the synthesis of the final implementation of the system.

## ACKNOWLEDGMENTS

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

## REFERENCES

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

[2] N. Audsley, K. Tindell, A. Burns, “The End of Line for Static Cyclic Scheduling?,” in *Proceedings of the 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,” in *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] B. P. Dave, N. K. Jha, “COHRA: Hardware-Software Cosynthesis of Hierarchical Heterogeneous Distributed Systems,” in *IEEE Transactions on CAD*, 17(10), 900-919, 1998.

[6] B. P. Dave, G. Lakshminarayana, N. J. Jha, “COSYN: Hardware-Software Co-Synthesis of Heterogeneous Distributed Embedded Systems,” in *IEEE Transactions on VLSI Systems*, 7(1), 92-104, 1999.

[7] EAST-EEA project, *ITEA Full Project Proposal,* http://www.itea-office.org, 2002.

[8] S. Edwards, *Languages for Digital Embedded Systems*, Kluwer Academic Publishers, 2000.

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

[10] R. Ernst, “Codesign of Embedded Systems: Status and Trends,” in *IEEE Design & Test of Computers*, April-June, 1998.

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

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

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

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

[15] L. Lavagno, A. Sangiovanni-Vincentelli, and E. Sentovich, “Models of Computation for Embedded System Design,” in *System-Level Synthesis*, Kluwer Academic Publishers, pages 45–102, 1999.

[16] E. A. Lee, T. M. Parks, “Dataflow process networks,” in *Proceedings of the IEEE*, volume 83, pages 773–801, May 1995.

[17] C. Lee, M. Potkonjak, W. Wolf, “Synthesis of Hard Real-Time Application Specific Systems,” in *Design Automation for Embedded Systems*, 4(4), 215-241, 1999.

[18] G. Leen, D. Heffernan, “Expanding automotive electronic systems,” in *Computer*, Volume: 35, Issue: 1, Pages 88-93, 2002.

[19] C. L. Liu, J. W. Layland, “Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment,” in *Journal of the ACM*, 20(1), 46-61, 1973.

[20] H. Lönn, J. Axelsson, “A Comparison of Fixed-Priority and Static Cyclic Scheduling for Distributed Automotive Control Applications,” in *Proceedings of the Euromicro Conference on Real-Time Systems*, 142-149, 1999.

[21] K. Melin, *Volvo S80: Electrical System of the Future*, Volvo Technology Report, 1998.

[22] T. Nolte, H. Hansson, C. Norström, S. Punnekkat, “Using bit-stuffing distributions in CAN analysis,” in *Proceedings of the IEEE/IEE Real-Time Embedded Systems Workshop*, 2001

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

[24] J. C. Palencia, M. González Harbour, “Exploiting Precedence Relations in the Schedulability Analysis of Distributed Real-Time Systems,” in *Proceedings of the 20th IEEE Real-Time Systems Symposium*, pages 328–339, 1999.

[25] T. Pop, P. Eles, Z. Peng, “Holistic Scheduling and Analysis of Mixed Time/Event-Triggered Distributed Embedded Systems,” in *Proceedings of the International Symposium on Hardware/Software Codesign*, 187-192, 2002.

[26] P. Pop, P. Eles, Z. Peng, “Scheduling with Optimized Communication for Time Triggered Embedded Systems,” in *Proceedings of the International Workshop on Hardware-Software Codesign*, 178-182, 1999.

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

[28] P. Pop, P. Eles, Z. Peng, “Schedulability Analysis and Optimization for the Synthesis of Multi-Cluster Distributed Embedded Systems,” in *Proceedings of Design Automation and Test in Europe Conference*, 184-189, 2003.

[29] P. Pop, *Analysis and Synthesis of Communication-Intensive Heterogeneous Real-Time Systems*, Linköping Studies in Science and Technology, Ph.D. Dissertation No. 833, 2003, available at http://www.ida.liu.se/~paupo/thesis

[30] P. Pop, P. Eles, Z. Peng, “Schedulability-Driven Communication Synthesis for Time-Triggered Embedded Systems,” in *Real-Time Systems Journal*, No. 24, 297–325, 2004.

[31] P. Pop, P. Eles, Z. Peng, V. Izosimov, M. Hellring, O. Bridal, “Design Optimization of Multi-Cluster Embedded Systems for Real-Time Applications,” in *Proceedings of Design, Automation and Test in Europe Conference*, 1028–1033, 2004.

[32] P. Puschner, A. Burns, “A Review of Worst-Case Execution-Time Analyses,” in *Real-Time Systems Journal*, Vol. 18, No. 2/3, May 2000.

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

[34] C. R. Reevs, *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell Scientific Publications, 1993.

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

[36] J. A. Stankovic, K. Ramamritham, *Advances in Real-Time Systems*, IEEE Computer Society Press, 1993.

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

[38] K. Tindell, A. Burns, A. Wellings, “Calculating CAN Message Response Times,” in *Control Engineering Practice*, 3(8), 1163-1169, 1995.

[39] K. Tindell, J. Clark, “Holistic Schedulability Analysis for Distributed Hard Real-Time Systems,” in *Microprocessing & Microprogramming*, Vol. 50, No. 2-3, 1994.

[40] TTTech, *TTP/C Specification Version 0.5*, 1999, availabe at http://www.tttech.com/

[41] TTTech, *Comparison CAN–Byteflight–FlexRay–TTP/C*, Technical Report, availabe at http://www.tttech.com/

[42] W. Wolf, “A Decade of Hardware/Software Codesign,” in *Computer*, 36/4, 38–43, 2003.

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

[44] J. Xu, D. L. Parnas, “Priority Scheduling Versus Pre-Run-Time Scheduling,” in *Journal of Real Time Systems*, volume 18, issue 1, pages 7–24, 2000.

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