Design Optimization of Multi-Cluster Embedded Systems for Real-Time Applications

Paul Pop[1], Petru Eles[1], Zebo Peng[1], Viacheslav Izosimov[1], Magnus Hellring[2], Olof Bridal[2]

[1]{paupo|petel|zebpe|viaiz}@ida.liu.se
Dept. of Computer and Information Science
Linköping University
SE–581 83 Linköping, Sweden

[2]{magnus.hellring|olof.bridal}@volvo.com
Dept. of Electronics and Software
Volvo Technology Corporation
SE–405 08 Göteborg, Sweden

Abstract

We present an approach to design optimization of multi-cluster embedded systems consisting of time-triggered and event-triggered clusters, interconnected via gateways. In this paper, we address design problems which are characteristic to multi-clusters: partitioning of the system functionality into time-triggered and event-triggered domains, process mapping, and the optimization of parameters corresponding to the communication protocol. We present several heuristics for solving these problems. Our heuristics are able to find schedulable implementations under limited resources, achieving an efficient utilization of the system. The developed algorithms are evaluated using extensive experiments and a real-life example.

1. Introduction

An increasing number of real-time applications are today implemented using distributed architectures consisting of interconnected clusters of processors. Each such cluster has its own communication protocol and two clusters communicate via a *gateway,* a node connected to both of them. 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, each function was running on a dedicated hardware component. However, in order to use the resources more efficiently and reduce costs, several functions have been integrated in one node and, at the same time, certain functionality has been distributed over several nodes (see Figure 1). Although an application is typically distributed over one single cluster, we begin to see applications that are distributed across several clusters, as illustrated in Figure 1 where the application represented as black dots, is distributed over the two clusters. 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.

Researchers have often ignored or very much simplified aspects concerning 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. [14].

Many efforts dedicated to communication synthesis have concentrated on the synthesis support for the communication infrastructure but without considering hard real-time constraints and system level scheduling aspects [4, 15]. We have to mention here some results obtained in extending real-time schedulability analysis so that network communication aspects can be handled. In [11], for example, the controller area network (CAN) protocol is investigated while the work reported in [12] deals with a simple time-division multiple access (TDMA) protocol.

There are two basic approaches for handling tasks in real-time applications [6]. 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 [1, 6, 13]. An interesting comparison, from a more industrial, in particular automotive, perspective, can be found in [7]. 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 processes being TT and others ET.

In [8] we have addressed design problems for systems where the TT and ET activities *share the same* processor and bus. 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) [6].
- 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 [2].

In this context, in [9] we have proposed an approach to schedulability analysis for multi-cluster distributed embedded systems. Starting from such an analysis, this paper is the first one to address specific design issues for multi-cluster systems. The proposed approaches solve the problems of partitioning an application between the TT and ET clusters, mapping the functionality of the application on the heterogeneous nodes of a cluster and adjusting the parameters of the communication protocols such that the timing constraints of the final implementation are guaranteed.

The paper is organized in 7 sections. The next section presents the application model as well as the hardware and software architecture of our systems. Section 3 presents the design optimization problems we are addressing in this paper, and section 4 presents our proposed heuristics for the design optimization of multi-cluster systems. The last two sections present the experimental results and conclusions.

2. Application Model and System Architecture

2.1 System Architecture

We consider architectures consisting of two interconnected clusters (see Figure 1). 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* $N_i \in \mathcal{N}_T \cup \mathcal{N}_E$ consists, among others, of a communication controller and a CPU. The gateway node N_G connected to both types of clusters, has two communication controllers, for TTP and CAN. The communication controllers implement the protocol services and run independently of the node's CPU.

Communication between the nodes on a TTC is based on the TTP [6]. The bus access scheme is TDMA, where each node N_i, including the gateway node, can transmit only during a predetermined time interval, the so called TDMA slot S_i. In such a slot, a node can send several messages

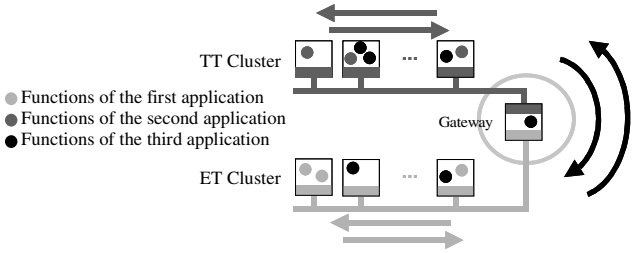

Figure 1. A System Architecture Example

packed in a frame. A sequence of slots corresponding to all the nodes in the TTC is called a TDMA round. A node can have only one slot in a TDMA round. Several TDMA rounds can be combined together in a cycle that is repeated periodically. The TDMA access scheme is imposed by a message descriptor list (MEDL) that is located in every TTP controller. The MEDL serves as a schedule table for the TTP controller which has to know when to send/receive a frame to/from the communication channel.

On an ETC the CAN [2] protocol is used for communication. The CAN bus is a priority bus that employs a collision avoidance mechanism, whereby the node that transmits the message with the highest priority wins the contention. Message priorities are unique and are encoded in the frame identifiers, which are the first bits to be transmitted on the bus.

Throughout the paper we will use the notation $\mathcal{B} = <\beta, \pi>$ to denote a certain communication configuration consisting of the sequence and size of the slots in a TDMA round on the TTC (β) and the priorities of the messages on the ETC (π).

We have designed a software architecture which runs on the CPU in each node, and which has a real-time kernel as its main component. 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. For more details about the software architecture and the message passing mechanism the reader is referred to [9].

The approaches presented in this paper can be easily extended to cluster configurations where there are several ETCs and TTCs interconnected by gateways.

2.2 Application Model

We model an application Γ as a set of directed, acyclic, polar graphs $G_i(V, E) \in \Gamma$. Each node $P_i \in V$ represents one process. An edge $e_{ij} \in E$ from P_i to P_j indicates that this output of P_i is an input to P_j. A process can be activated after all its inputs have arrived and it issues its outputs when it terminates. 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 mapping of a process graph $G(V, E)$ is given by a function $M: V \rightarrow \mathcal{N}$, where $\mathcal{N} = \mathcal{N}_T \cup \mathcal{N}_E$ is the set of nodes in the architecture. For a process $P_i \in V$, $M(P_i)$ is the node to which P_i is assigned for execution. Each process P_i can potentially be mapped on several nodes. Let $\mathcal{N}_{P_i} \subseteq \mathcal{N}$ be the set of nodes to which P_i can potentially be mapped. We consider that for each $N_k \in \mathcal{N}_{P_i}$, we know the worst-case execution time[1] $C_{P_i}^{N_k}$ of process P_i, when executed on N_k. We also consider that the size of the messages is given.

Processes and messages activated based on events also have a uniquely assigned priority, p_{P_i} for processes and p_{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. In addition, processes can have associated individual release times and deadlines. If communicating processes are of different periods, they are combined into a hyper-graph capturing all process activations for the hyper-period (LCM of all periods).

3. Design Optimization Problems

Considering the type of applications and systems described in section 2, and using the analysis proposed in [9] and briefly outlined in the section 3.2, several design optimization problems can be addressed. In this paper, we address problems which are characteristic to applications distributed across multi-cluster systems consisting of heterogeneous TT and ET networks. In particular, we are interested in the following issues:

1. partitioning of the processes of an application into time-triggered and event-triggered domains, and their mapping to the nodes of the clusters;
2. scheduling of processes and messages;
3. optimization of the access to the communication infrastructure.

The goal is to produce an implementation which meets all the timing constraints of the application.

3.1 Partitioning and Mapping

In this paper, by partitioning we denote the decision whether a certain process should be assigned to the TT or the ET domain (and, implicitly, to a TTC or an ETC, respectively). Mapping a process means assigning it to a particular node inside a cluster.

Very often, the partitioning decision is taken based on the experience and preferences of the designer, considering aspects like the functionality implemented by the process, the hardness of the constraints, sensitivity to jitter, legacy constraints, etc. Let $\mathcal{P}$ be the set of processes in the application Γ. We denote with $\mathcal{P}_T \subseteq \mathcal{P}$ the subset of processes which the designer has assigned to the TT cluster, while $\mathcal{P}_E \subseteq \mathcal{P}$ contains processes which are assigned to the ET cluster.

Many processes, however, do not exhibit certain particular features or requirements which obviously lead to their implementation as TT or ET activities. The subset $\mathcal{P}^+ = \mathcal{P} \setminus (\mathcal{P}_T \cup \mathcal{P}_E)$ of processes could be assigned to any of the TT or ET domains. Decisions concerning the partitioning of this set of activities can lead to various trade-offs concerning, for example, the schedulability properties of the system, the amount of communication exchanged through the gateway, the size of the schedule tables, etc.

For part of the partitioned processes, the designer might have already decided their mapping. For example, certain processes, due to constraints like having to be close to sensors/actuators, have to be physically located in a particular hardware unit. They represent the sets $\mathcal{P}_T^M \subseteq \mathcal{P}_T$ and $\mathcal{P}_E^M \subseteq \mathcal{P}_E$ of already mapped TT and ET processes, respectively. Consequently, we denote with $\mathcal{P}_T^* = \mathcal{P}_T \setminus \mathcal{P}_T^M$ the TT processes for which the mapping has not yet been decided, and similarly, with $\mathcal{P}_E^* = \mathcal{P}_E \setminus \mathcal{P}_E^M$ the unmapped ET processes. The set $\mathcal{P}^* = \mathcal{P}_T^* \cup \mathcal{P}_E^* \cup \mathcal{P}^+$ then represents all the unmapped processes in the application.

The mapping of messages is decided implicitly by the mapping of processes. Thus, a message exchanged between two processes on the TTC (ETC) will be mapped on the TTP bus (CAN bus) if these processes are allocated to different nodes. If the communication takes place between two clusters, two message instances will be created, one mapped on the TTP bus and one on the CAN bus. The first message is sent from the sender node to the gateway, while the second message is sent from the gateway to the receiving node.

Let us illustrate some of the issues related to partitioning in such a context. In the example presented in Figure 2 we have an application[2] with six processes, P_1 to P_6, and four nodes, N_1 and N_2 on the TTC, N_3 on the ETC and the gateway node N_G. The worst-case execution times on each node are given to the right of the application graph. Note that N_2 is faster than N_3, and an "X" in the table means that the process is not allowed to be mapped on that node. The mapping of P_1 is fixed on N_1, P_3 and P_6 are mapped on N_2, P_2 and P_5 are fixed on N_3, and we have to decide how to partition P_4 between the TT and ET domains. Let us also assume that process P_5 is the

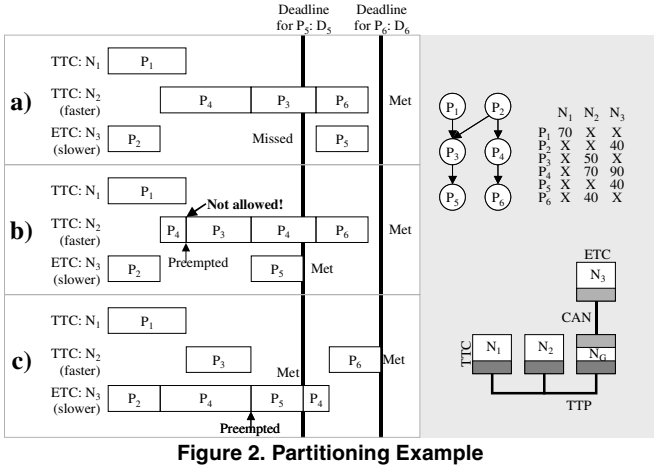

Figure 2. Partitioning Example

1. In this paper we consider hard real-time applications where violating a timing constraint is not acceptable.

2. Communications are ignored for this example.

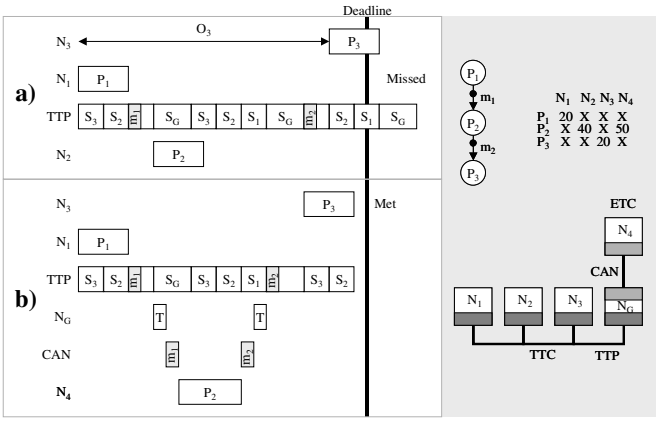

Figure 3. Mapping Example

highest priority process on N_3. In addition, P_5 and P_6 have each a deadline, D_5 and D_6, respectively, as illustrated in the figure by thick vertical lines.

We can observe that although P_3 and P_4 do not have individual deadlines, their mapping and scheduling has a strong impact on their successors, P_5 and P_6, respectively, which are deadline constrained. Thus, we would like to map P_4 such that not only P_3 can start on time, but P_4 also starts soon enough to allow P_6 to meet its deadline.

As we can see from Figure 2a, this is impossible to achieve by mapping P_4 on the TTC node N_2. It is interesting to observe that, if preemption would be allowed in the TT domain, as in Figure 2b, both deadlines could be met. This, however, is impossible on the TTC where preemption is not allowed. Both deadlines can be met only if P_4 is mapped on the slower ETC node N_3, as depicted in Figure 2c. In this case, although P_4 competes for the processor with P_5, due to the preemption of P_4 by the higher priority P_5, all deadlines are satisfied.

For a multi-cluster architecture the communication infrastructure has an important impact on the design and, in particular, the mapping decisions. Let us consider the example in Figure 3. We assume that P_1 is mapped on node N_1 and P_3 on node N_3 on the TTC, and we are interested to map process P_2. P_2 is allowed to be mapped on the TTC node N_2 or on the ETC node N_4, and its execution times are depicted in the table to the right of the application graph.

In order to meet the deadline, one would map P_2 on the node it executes fastest, N_2 on the TTC, see Figure 3a. However, this will lead to a deadline miss due to the TTP slot configuration which introduces communication delays. The application will meet the deadline only if P_2 is mapped on the slower node, i.e., node N_4 in the case in Figure 3b[1]. Not only is N_4 slower than N_2, but mapping P_2 on N_4 will place P_2 on a different cluster than P_1 and P_3, introducing extra communication delays through the gateway node. However, due to the actual communication configuration, the mapping alternative in Figure 3b is desirable.

3.2 Multi-Cluster Scheduling

Once a partitioning and a mapping is decided, and a communication configuration is fixed, the processes and messages have to be scheduled. For the TTC this means building the schedule tables, while for the ETC the priorities of the ET processes have to be determined and their schedulability has to be analyzed. In [9] we have proposed an analysis for hard real-time applications mapped on multi-cluster systems. The aim is to find out if a system is schedulable, i.e. all the timing constraints are met.

The basic idea is that 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*. In [9], for the ETC we used 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. We used 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 3, O_3 is the offset of P_3, determined to guarantee that when P_3 is activated, message m_2 is already available.

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.

The MultiClusterScheduling (MCS) algorithm proposed in [9] takes as input an application Γ, a mapping $\mathcal{M}$ and a bus configuration $\mathcal{B}$, builds the TT schedule tables, sets the ET priorities for processes, and provides the global analysis.

3.3 Bus Access Optimization

As shown in section 3.1, the communication has an important impact on the partitioning and mapping process. Hence, we are interested to determine the parameters $\mathcal{B}=<\beta, \pi>$ (see section 2.1) of the communication infrastructure such that the implementation is schedulable.

Figure 4 presents a bus access optimization example. An application consisting of processes P_1 to P_4 and messages m_1 to m_4 is mapped on a two-cluster architecture with node N_1 on the TTC and node N_2 on the ETC. Let us assume that the mapping is already determined such that processes P_1 and P_4 are mapped on N_1 and P_2 and P_3 are mapped on N_2, with the execution times given in the table in Figure 4. Moreover, we assume that all messages have the same size, but the transmission times on the CAN bus are two times faster than on the TTP.

In the situation depicted in Figure 4a we have the TDMA configuration with slot S_1 able to hold one message, followed by slot S_G which can hold two messages. The priority on the CAN bus is $p_{m_2} < p_{m_3}$. In this case, P_3's input message m_2 will be delayed by the transmission of m_3, and hence P_3's output message m_4 will miss the slot S_G in round 4, and will have to take the next round, leading to a deadline miss.

However, if we set the priority p_{m_2} greater than p_{m_3}, the transmission of the message m_4 on the CAN bus will not be delayed, and will catch round 4, reducing the response time of the application, which, however, is still unschedulable.

Further improvements can be achieved by changing the order of the slots and their length in the TDMA round as indicated in Figure 4c where S_G is now the first slot, and S_1 is the second slot. S_G has been shortened to hold only one message, while S_1 has been enlarged to hold two messages. Let us consider that $p_{m_2} > p_{m_1}$ on the CAN bus. With such a configuration, the implementation in Figure 4c meets the deadline.

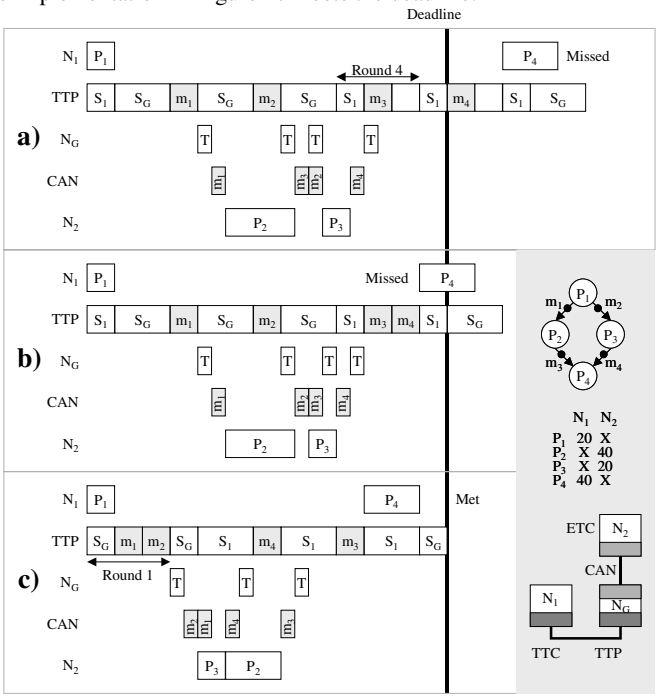

Figure 4. Bus Access Optimization Example

1. Process T in Figure 3b executing on the gateway node N_G is responsible for transferring messages between the TTP and CAN controllers.

```
MultiClusterConfiguration(Γ)
1 Step 1: B^0 = InitialBusAccess(Γ)
2         M^0 = InitialPM(Γ, B^0)
3         if MultiClusterScheduling(Γ, M^0, B^0) returns schedulable then stop end if
4 Step 2: M = PMHeuristic(Γ, M^0, B^0)
5         if MultiClusterScheduling(Γ, M, B^0) returns schedulable then stop end if
6 Step 3: B = BusAccessOptimization(Γ, M)
7         MultiClusterScheduling(Γ, M, B)
end MultiClusterConfiguration
```

Figure 5. The General Strategy

3.4 Exact Problem Formulation

As an input we have an application Γ given as a set of process graphs (section 2.2) and a two-cluster system consisting of a TT and an ET cluster. As introduced previously, $\mathcal{P}_T$ and $\mathcal{P}_E$ are the sets of processes already partitioned into TT and ET, respectively. Also, $\mathcal{P}_T^M \subseteq \mathcal{P}_T$ and $\mathcal{P}_E^M \subseteq \mathcal{P}_E$ are the sets of already mapped TT and ET processes.

We are interested to:

1. find a partitioning for processes in $\mathcal{P}^+ = \mathcal{P} \setminus (\mathcal{P}_T \cup \mathcal{P}_E)$ and decide a mapping for processes in $\mathcal{P}^* = \mathcal{P}_T^* \cup \mathcal{P}_E^* \cup \mathcal{P}^+$, where $\mathcal{P}_T^* = \mathcal{P}_T \setminus \mathcal{P}_T^M$, and $\mathcal{P}_E^* = \mathcal{P}_E \setminus \mathcal{P}_E^M$;
2. determine a bus configuration $\mathcal{B} = <\beta, \pi>$, where β is the sequence and size of the slots in a TDMA round on the TTC, and π is the set of message priorities on the ETC

such that imposed deadlines are guaranteed to be satisfied.

4. Design Optimization Strategy

The design problem formulated in the previous section is NP complete. Our strategy, outlined in Figure 5, is to divide the problem into several, more manageable, subproblems. The MultiClusterConfiguration strategy (MCC) has three steps:

In the first step (lines 1–3) we decide very quickly on an initial bus access configuration $\mathcal{B}^0$, and an initial partitioning and mapping $\mathcal{M}^0$. The initial bus access configuration (line 1) is determined, for the TTC, by assigning in order nodes to the slots ($S_i = N_i$) and fixing the slot length to the minimal allowed value, which is equal to the length of the largest message in the application. For the ETC we calculate the message priorities π based on the deadlines of the receiver processes. The initial partitioning and mapping algorithm (line 2 in Figure 5) is described in section 4.1. Once an initial partitioning and bus configuration are obtained, the application is scheduled using the MultiClusterScheduling algorithm outlined in section 3.2 (line 3).

If the application is schedulable the optimization strategy stops. Otherwise, it continues with the second step by using an iterative improvement PMHeuristic (line 4), presented in section 4.2, to improve the partitioning and mapping obtained in the first step.

If the application is still not schedulable, we use, in the third step, the algorithm in section 4.3 to optimize the access to the communication infrastructure (line 6). If the application is still unschedulable, we conclude that no satisfactory implementation could be found with the available amount of resources.

4.1 Initial Partitioning and Mapping (IPM)

Our IPM algorithm (Figure 6) receives as input the merged graph $\mathcal{G}$ and the bus configuration $\mathcal{B}$. $\mathcal{G}$ is obtained by merging all the graphs of the application, and has a period equal to the LCM of all constituent graphs [10].

The IPM algorithm uses a list scheduling based greedy approach. A process P_i is placed in the ready list $\mathcal{L}$ if all its predecessors have been already scheduled. In each iteration of the loop (lines 2–7), all ready processes from the list $\mathcal{L}$ are investigated, and that process P_i is selected for mapping by the SelectProcess function, which has the largest delay $\delta_i = r_i + l_i$. In the previous equation, r_i is the response time of process P_i on the fastest node in $\mathcal{N}_{P_i}$, and l_i is the critical path starting from process P_i, defined as:

```
InitialPM(G, B) -- Initial Partitioning and Mapping
1 L = {source of G} -- start initial mapping with the first node of the merged graph
2 while L ≠ ∅ do -- visits ready processes in the order of list scheduling
3       P = SelectProcess(L)
4       N = SelectNode(N_P)
5       M(P) = N -- map process P on node N
6       L = UpdateReadyList(L)
7 end while
end InitialPM
```

Figure 6. The Initial Partitioning and Mapping

$$l_i = \max_k \sum_{\forall \tau_j \in \pi_{ik}} r_{\tau_j} \quad (1)$$

where π_{ik} is the k^{th} path from process P_i to the sink node of $\mathcal{G}$ (not including P_i), and r_{τ_j} is the response time of a process or message on π_{ik}. The response times are calculated using the MultiClusterScheduling function, under the following assumptions:

- Every yet unpartitioned/unmapped process $P_i \in \mathcal{P}^*$ is considered mapped on the fastest node from the list of potential nodes $\mathcal{N}_{P_i}$.
- The worst-case response time for messages sent or received by yet unpartitioned/unmapped processes is considered equal to zero.

Let us consider the design example in Figure 7 where we have five processes, P_1 to P_5, and three nodes, N_1 on the TTC, N_2 on the ETC and the gateway node N_G. The initial bus configuration, consisting of the slots order and size, together with the ET message priorities, is also given. The mapping of P_3 is fixed on N_1, P_5 is fixed on N_2, and we have to decide where to partition and map P_1, P_2 and P_4. In the first iteration of IPM, SelectProcess has to decide between P_1 and P_2 which are ready for execution. The critical path of P_1 is $l_1 = \max(r_{m_1} + r_3 + r_{m_4} + r_5, r_{m_2} + r_4 + r_{m_5} + r_5) = \max(0 + 40 + 40 + 40, 0 + 30 + 0 + 40) = 120$, while $l_2 = r_{m_3} + r_4 + r_{m_5} + r_5 = 0 + 30 + 0 + 40 = 70$. Thus, the delay of P_1 is $\delta_1 = C_1^{N_2} + l_1 = 30 + 120 = 150$, and the delay of P_2 is $\delta_2 = C_2^{N_2} + l_2 = 60 + 70 = 130$[1]. Therefore, SelectProcess will select P_1 because it has a larger delay.

Once a process P_i is selected, all mapping alternatives of P_i to nodes[2] in $\mathcal{N}_{P_i}$ are tested by the SelectNode function. Out of these alternatives, SelectNode returns that node N_k which leads to the smallest end-to-end delay $\delta_i^{N_k}$ on the application graph:

$$\delta_i^{N_k} = O_i^{N_k} + r_i^{N_k} + l_i^{N_k}. \quad (2)$$

In the previous equation, $O_i^{N_k}$ is the offset of process P_i when mapped on node N_k (i.e., the earliest possible starting time taking into account the predecessors and the communication delay of the incoming messages) calculated by our scheduling algorithm.

The worst-case response time $r_i^{N_k}$ is equal to the worst-case execution time $C_i^{N_k}$ if N_k is in the TTC ($N_k \in \mathcal{N}_T$). If N_k is in the ETC ($N_k \in \mathcal{N}_E$), the worst-case response time is calculated according to the equations below:

$$r_i^{N_k} = J_i + w_i + C_i^{N_k} \quad (3)$$

$$w_i = B_i + \sum_{\forall P_j \in hp(P_i)} \left\lceil \frac{w_i + J_j - O_{ij}^{N_k}}{T_j} \right\rceil_0 C_j^{N_k} \quad (4)$$

where J_i is the worst-case jitter of process P_i, and w_i represents the worst-case interference on P_i caused by lower priority processes in their critical section (the term B_i in Equation 4) and by higher priority processes $P_j \in hp(P_i)$ running on the same node N_k with P_i (the second term[3] in Equation 4). T_j represents the period of process P_j and O_{ij} is a positive value representing the relative offset of processes P_i and P_j.

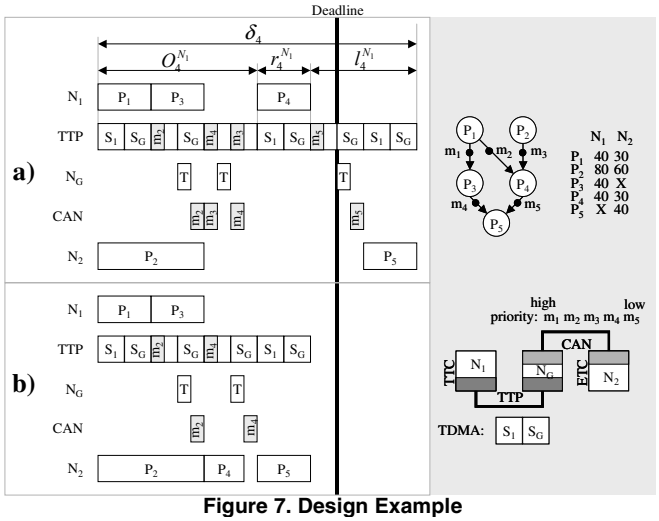

Figure 7. Design Example

1. According to the first assumption, both P_1 and P_2 are considered mapped on the fastest node.
2. $\mathcal{N}_{P_i}$ is the set of nodes on which process P_i could, potentially, be executed. If the process is already partitioned to a certain cluster, only nodes in that cluster are considered.
3. $\lceil x \rceil_0$ is the positive ceiling, returning the smallest integer greater than x, 0 if x is negative.

The third term of the delay $\delta_i^{N_k}$ represents the critical path from process P_i to the sink node (as introduced in Equation 1) in the case P_i is mapped on node N_k. The delay $\delta_i^{N_k}$ is calculated by the MultiClusterScheduling function, under the same assumptions mentioned earlier.

Let us go back to the example in Figure 7. IPM has decided in the first two iterations of the while loop (lines 2–7 in Figure 6) that P_1 should be mapped on N_1 and P_2 on N_2. In the third iteration, P_4 has been selected by SelectProcess, and now the mapping alternatives on N_1 and N_2 are tested by the SelectNode function. According to Equation 2, if P_4 is mapped on N_1 we have $\delta_4^{N_1} = O_4^{N_1} + r_4^{N_1} + l_4^{N_1} = 120 + 40 + (40 + 40) = 240$ (see Figure 7a). Similarly, for the alternative on N_2 we have $\delta_4^{N_2} = O_4^{N_2} + r_4^{N_2} + l_4^{N_2} = 80 + 30 + (0 + 40) = 150$ (see Figure 7b). Thus, P_4 will be mapped on N_2 which produces the smallest delay of 150. IPM will finally produce the schedulable solution presented in Figure 7b.

4.2 Partitioning and Mapping Heuristic (PMH)

If, after the initial partitioning, mapping and bus setup we do not obtain a schedulable application, we apply an iterative improvement algorithm, the PMHeuristic in Figure 8. The algorithm receives as input the application Γ, the initial partitioning and mapping $\mathcal{M}^0$ produced by IPM, and the bus configuration $\mathcal{B}^0$, and produces a partitioning and mapping for processes in $\mathcal{P}^*$.

We investigate each unschedulable graph $G_i \in \Gamma$, i.e., the response time r_{G_i} is larger than the deadline D_{G_i}. Our heuristic is to perform changes to the mapping of processes in Γ that would reduce the critical path of G_i, and thus the worst-case response time r_{G_i}.

In each iteration, the algorithm selects that unschedulable process graph G_i which has the maximum delay $\Delta_{G_i} = r_{G_i} - D_{G_i}$ between its response time and the deadline (line 2). Let us denote the maximum delay with Δ_{max}, and the corresponding graph with G_{max}. Next, we determine the critical path $\mathcal{P}_{CP}$ of the process graph G_{max}. For example, for the process graph in Figure 7 scheduled as in case (a), the critical path is composed of P_2, P_4 and P_5.

The intelligence of the heuristic lies in how it determines changes (i.e., design transformations) to the mapping of processes that potentially can lead to a shortening of the critical path (lines 7 and 9). The list of proposed changes *List* leading to a potential improvement are then evaluated (lines 11–18) to find out the change that produces the largest reduction of Δ_{max}, which is finally applied to the system configuration (line 20). Reducing Δ_{max} means, implicitly, reducing the response time of the process graph G_{max} investigated in the current iteration. The algorithm terminates if all graphs in the application are schedulable, or no improvement to Δ_{max} is found.

Since a call to MultiClusterScheduling that evaluates the changes is costly in terms of execution time, it is crucially to find out a short list of proposed changes that will potentially lead to the largest improvement. Looking at Equation 2, we can observe that the length of the critical path $\mathcal{P}_{CP}$ would be reduced if, for a process $P_i \in \mathcal{P}_{CP}$, we would:

```
PMHeuristic(Γ, M, B) -- Partitioning and Mapping Heuristic
1  while (∃ G_i ∈ Γ ∧ r_Gi > D_Gi) and (Δ_max improved in the previous iteration) do
2      Δ_max = maximum of r_Gi - D_Gi, ∀ G_i ∈ Γ ∧ r_Gi > D_Gi
3      G_max = graph corresponding to Δ_max
4      P_CP = FindCriticalPath(G_max)
5      for each P_i ∈ P_CP do -- find changes with a potential to improve r_Gmax
6          if M(P_i) ∈ N_T then
7              List = ProposedTTChanges(P_i)
8          else -- in this case M(P_i) ∈ N_E
9              List = ProposedETChanges(P_i)
10         end if
11         for each ProposedChange ∈ List do -- determine the improvement
12             Perform(ProposedChange); MultiClusterScheduling(Γ, M, B)
13             Δ_max = maximum of r_Gi - D_Gi, ∀ G_i ∈ Γ ∧ r_Gi > D_Gi
14             if Δ_max smallest so far then
15                 BestChange = ProposedChange
16             end if
17             Undo(ProposedChange)
18         end for
19         -- apply the move improving the most
20         If ∃ BestChange then Perform(BestChange) end if
21     end for
22 end while
23 return M
end PMHeuristic
```

Figure 8. The Partitioning and Mapping Heuristic

1. reduce the offset O_i (first term of Equation 2);
2. decrease the worst-case response time r_i (second term);
3. reduce the critical path from P_i to the sink node (third term).

To reduce (1) we have to reduce the delay of the communication from P_i's predecessors to P_i. Thus, we consider transformations that would change the mapping of process P_i and of predecessors of P_i such that the communication delay is minimized. However, only those predecessors are considered for remapping which actually delay the execution of P_i. Let us go back to Figure 7, and consider that PMH starts from an initial partitioning and mapping as depicted in Figure 7a. In this case, to reduce the offset O_4 of process P_4, we will consider mapping P_4 on node N_2 as depicted in Figure 7b, reducing thus the offset from 120 to 80.

The approach to reduce (2) depends on the type of process. Both for TT and ET processes we can decrease the worst-case execution time C_i by selecting a faster node. For example, in Figure 7, by moving P_4 from N_2 to N_1 we reduce its worst-case execution time from 40 to 30. However, for ET processes we can further reduce r_i by investigating the interference from other processes on P_i (Equation 4). Thus, we consider mapping processes with a priority higher than P_i on other nodes, reducing thus the interference.

Point (3) is concerned with the critical path from process P_i to the sink node. In this case, we are interested to reduce the delay of the communication from P_i to its successor process on the critical path. This is achieved by considering changes to the mapping of P_i or to the mapping of the successor process (e.g., by including them in the same cluster, same processor, etc.). For example, in Figure 7a, the critical path of P_4 is enlarged by the communication delay due to m_5 exchanged by P_4 on the TTC with P_5 on the ETC. To reduce the length of the critical path we will consider mapping P_4 to N_2, and thus the communication will take place on the same processor.

4.3 Bus Access Optimization (BAO)

The BusAccessOptimization function (line 6 in Figure 5) determines the configuration $\mathcal{B}$ consisting of the sequence and size of the slots in a TDMA round on the TTC (β) and the priorities of messages on the ETC (π). This optimization is performed as a last attempt to obtain a schedulable configuration. The optimization of the β and π parameters starts from the initial values set by the InitialBusAccess function.

The algorithm performs a greedy optimization whereby the ET priorities π are determined using the HOPA heuristic [5], where priorities in a distributed real-time system are determined based on the local deadlines, which are calculated for each activity considering the end-to-end (global) deadlines. Next, the TTP configuration β is determined. Thus, simultaneously with searching for the right node $N_i \in \mathcal{N}_T \cup \{N_G\}$ to be assigned to the first slot, the algorithm looks for the optimal slot length. Once a node was selected for the first slot and a slot length fixed, the algorithm continues with the next slots, trying to assign nodes (and to fix slot lengths) from those nodes which have not yet been assigned. When calculating the length of a certain slot we consider the feedback from the MultiClusterScheduling algorithm which recommends slot sizes to be tried out. Before starting the actual optimization process for the bus access scheme, a scheduling of the initial solution is performed which generates the recommended slot lengths. We refer the reader to [3] for details concerning the generation of the recommended slot lengths for the time-triggered protocol.

5. Experimental Results

For the evaluation of our algorithms we used applications of 50, 100, 150, 200, and 250 processes (all unpartitioned and unmapped), to be implemented on two-cluster architectures consisting of 2, 4, 6, 8, and 10 different nodes, respectively, half on the TTC and the other half on the ETC, interconnected by a gateway.

Thirty examples were randomly generated for each application dimension, thus a total of 150 applications were used for experimental evaluation. We generated both graphs with random structure and graphs based on more regular structures like trees and groups of chains. Execution times and message lengths were assigned randomly using both uniform and exponential distribution within the 10 to 100 ms, and 2 to 8 bytes ranges, respectively. The experiments were done on SUN Ultra 10 computers.

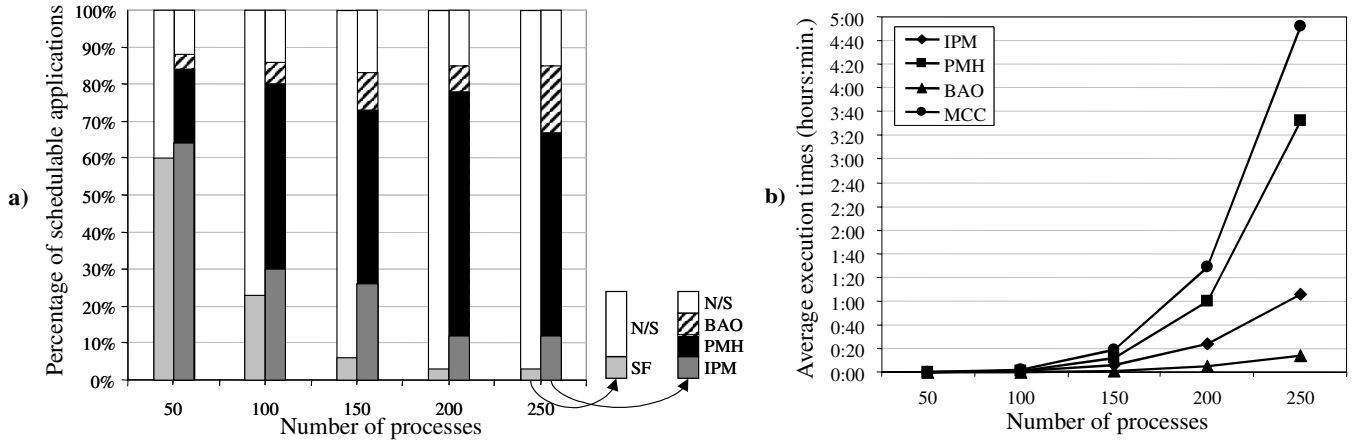

Figure 9. Comparison of the Heuristics

We were interested to evaluate the proposed approaches. Hence, we have implemented each application, on its corresponding architecture, using the MultiClusterConfiguration (MCC) strategy from Figure 5. Figure 9a presents the number of schedulable solutions found after each step of our optimization strategy (N/S stands for "not schedulable"). Together with the MCC steps, Figure 9a also presents a straightforward solution (SF). The SF approach performs a partitioning and mapping that tries to balance the utilization among nodes and buses. This is a configuration which, in principle, could be elaborated by a careful designer without the aid of optimization tools like the one proposed in the paper.

Out of the total number of applications, only 19% were schedulable with the implementation produced by SF. However, using our MCC strategy, we are able to obtain schedulable applications in 85% of the cases: 30% after step one (IPM), 76% after step two (PMH), and 85% after step three (BAO). It is easy to observe that, for all application dimensions, by performing the proposed optimization steps, large improvements over the straightforward configuration could be produced. Moreover, as the applications become larger, it is more difficult for SF to find schedulable solutions, while the optimization steps of MCC perform very well. For 150 processes, for example, MCC has been able to find schedulable implementations for 83% of the applications. The bottom bar, corresponding for 26%, is the percentage of schedulable applications found by IPM. On top of that, PMH, depicted by a black bar, adds another 47%. The top bar from the stack, represented using a hashed rectangle, represent the additional 10% of schedulable implementations found after performing the bus access optimization (BOA).

Figure 9b presents the execution times for each of the three steps of our multi-cluster configuration strategy, as well as for the complete algorithm (MCC). Note that the times presented in the figure for MCC include a complete optimization loop, that performs partitioning, mapping, bus access optimization and scheduling. The complete optimization process implemented by the MCC strategy takes under five hours for very large process graphs of 250 processes, while for applications consisting of 100 processes it takes on average 2.28 minutes.

Finally, we considered a real-life example implementing a vehicle cruise controller (CC). The process graph that models the CC has 32 processes, and is described in [10]. The CC was mapped on an architecture consisting of five nodes: Engine Control Module (ECM) and Electronic Throttle Module (ETM) on the TTC, Anti Blocking System (ABS) and Transmission Control Module (TCM) on the ETC, and the Central Electronic Module (CEM) as the gateway. We have considered a deadline of 150 ms.

In this setting, the SF approach failed to produce a schedulable implementation, leading to response time of 392 ms. After IPM (first step of MCC), we were able to reduce the response time of the CC to 154, which is still larger than the deadline. However, applying PMH (step two) we are able to obtain a schedulable implementation with a response time of 146 ms. Applying the BAO step is able to reduce it further to 144 ms. All these three steps taken together execute for under two minutes for the CC.

6. Conclusions

In this paper we have presented design optimization strategies for real-time applications distributed over multi-cluster systems. We have considered systems of time-triggered and event-triggered clusters, interconnected via gateways.

The proposed approaches solve the problems characteristic to such multi-cluster systems: partitioning, mapping the functionality of the application on the heterogeneous nodes of a cluster and adjusting the parameters of the communication protocols such that the timing constraints of the application are guaranteed.

Extensive experiments using synthetic applications, as well as a real-life example, show that by using our optimization approaches we are able to find schedulable implementations under limited resources, achieving an efficient utilization of the system.

References

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

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

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

[4] R. Ernst, "Codesign of Embedded Systems: Status and Trends", IEEE Design & Test of Computers, April-June, 1998.

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

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

[7] 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.

[8] T. Pop, P. Eles, Z. Peng, "Design Optimization of Mixed Time/Event-Triggered Distributed Embedded Systems", CODES+ISSS'03, 83-89.

[9] P. Pop, P. Eles, Z. Peng, "Schedulability Analysis and Optimization for the Synthesis of Multi-Cluster Distributed Embedded Systems", DATE'03, 184-189.

[10] P. Pop, "Analysis and Synthesis of Communication-Intensive Heterogeneous Real-Time Systems", Linköping Studies in Science and Technology, Ph.D. Dissertation No. 833.

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

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

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

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

[15] W. Wolf, "A Decade of Hardware/Software Codesign," Computer, 36/4, 38–43, 2003.