Design Optimization of Mixed Time/Event-Triggered Distributed Embedded Systems

Traian Pop, Petru Eles, Zebo Peng
Dept. of Computer and Information Science, Linköping University
{trapo, petel, zebpe}@ida.liu.se

Abstract

Distributed embedded systems implemented with mixed, event-triggered and time-triggered task sets, which communicate over bus protocols consisting of both static and dynamic phases, are emerging as the new standard in application areas such as automotive electronics. In a previous paper, we have developed a holistic timing analysis and scheduling approach for this category of systems. Based on this result, in the present paper, new design problems are solved, which we identified as characteristic for such hybrid systems: partitioning of the system functionality into time-triggered and event-triggered domains and the optimization of parameters corresponding to the communication protocol. We addressed both problems in the context of a heuristic which performs mapping and scheduling of the system functionality. We demonstrated the efficiency of the proposed technique with extensive experiments.

Categories and Subject Descriptors

B.8.2 [Performance and Reliability]: Performance Analysis and Design Aids;
C.1.4 [Processor Architectures]: Distributed Systems;
C.3 [Special-Purpose and Application-Based Systems]: Real-Time and Embedded Systems;
D.4.1 [Operating Systems]: Scheduling;
J.6 [Computer-aided Engineering]: Computer Aided Design (CAD);

General Terms

Algorithms, Performance, Design.

Keywords

Distributed embedded systems, real-time systems, time-triggered, event-triggered, mixed communication protocols, automotive applications.

1. Introduction

There are two basic approaches for handling tasks in real-time applications [7]. In the event-triggered approach (ET), activities are initiated whenever a particular event is noted. In the time-triggered (TT) approach, activities are initiated at predetermined points in time. There has been a long debate in the real-time and embedded systems communities concerning the advantages of each approach [1, 8, 12].

If we look at the communication infrastructure, message passing activities can be triggered either dynamically, in response to an event, as with the controller area network (CAN) bus [3], or statically, at predetermined moments in time, as in the case of time-division multiple access (TDMA) protocols and, in particular, the time-triggered protocol (TTP) [7].

CODES+ISSS'03, October 1-3, 2003, Newport Beach, California, USA.

In [8] the authors compare the ET and TT approaches from an industrial perspective (considering, in particular, automotive applications). Their conclusion is that one has to choose the right approach depending on the particularities of the actual tasks. This means not only that there is no single "best" approach to be used, but also that inside a certain application the two approaches can be used together, some tasks being TT and others ET.

In this context, it is not surprising that several activities have been started aiming at the development and standardization of bus protocols which support both static (ST) and dynamic (DYN) communication. Such a protocol has been suggested in [9] and [10]. Also, the first mixed protocol has been proposed by a consortium, to be used in automotive applications [6]. In [4], the authors describe the so called Universal Communication Model (UCM), a framework for modelling at a high level of abstraction the communication infrastructure in automotive applications.

New, highly sophisticated automotive applications consist of both TT and ET task sets implemented on top of complex distributed architectures based on mixed ST/DYN bus protocols. In [13] we have presented an approach to scheduling and schedulability analysis for such mixed time/event triggered systems. Such an analysis and scheduling procedure constitutes the fundament for any synthesis approach aiming at an efficient, highly optimized implementation of a distributed application which is also guaranteed to meet the timing constraints.

Starting from such a holistic scheduling and analysis, this paper is the first one to address specific design issues of hybrid ET/TT systems like those outlined above. The proposed approach solves the problems of partitioning a certain functionality into ET and TT, mapping the functionality on a distributed architecture and adjusting the parameters of the communication protocol such that the timing constraints of the final implementation are guaranteed.

The paper is organized in 7 sections. In the next section we present the architecture of the distributed systems and the application model that we are studying. Section 3 describes briefly the holistic scheduling and schedulability analysis we have developed in [13]. Some specific optimization issues are presented in Section 4. Section 5 states the design problem we intend to solve and outlines our solution, while Section 6 presents some experimental results. The last section presents our conclusions.

2. System Architecture and Application Model

2.1 Hardware Architecture and Bus Access

We consider architectures consisting of nodes connected by a unique broadcast communication channel. Each node consists of a communication controller, a CPU, memories (RAM, ROM), and an I/O interface to sensors and actuators (see Figure 1).

We model the bus access scheme using the Universal Communication Model [4]. The bus access is organized as consecutive cycles, each with the duration T_{bus}. We consider that the communication cycle is partitioned into static and dynamic phases (Figure 1). Static phases consist of time slots, and during a slot only one node is allowed to send ST messages; this is the node associated to that particular slot.

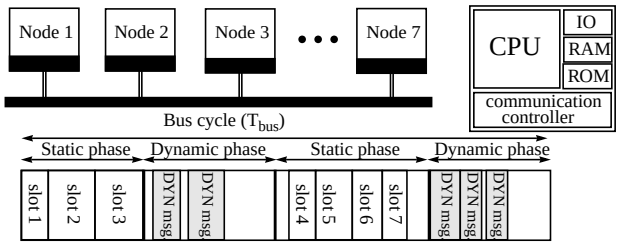

Figure 1. System Architecture

During a dynamic phase, all nodes are allowed to send DYN messages and the conflicts between nodes trying to send simultaneously are solved by an arbitration mechanism based on priorities assigned to messages. The bus access cycle has the same structure during each period T_{bus}. Every node has a communication controller that implements the static and dynamic protocol services. The controller runs independently of the node's CPU.

2.2 Software Architecture

For the systems we are studying, we have designed a software architecture which runs on the CPU of each node. The main component of the software architecture is a real-time kernel which supports both time-triggered and event-triggered activities. An activity is defined as either the execution of a task or as the transmission of a message on the bus. For the TT activities, the kernel relies on a static schedule table which contains all the information needed to take decisions on activation of TT tasks or transmission of ST messages. For the ET tasks, the kernel maintains a prioritized ready queue in which tasks are placed whenever their triggering event has occurred and they are ready for activation, or when they have been pre-empted.

The real-time kernel will always activate a TT task at the particular time fixed for that task in the schedule table. If at that moment, an ET task is running on that node, that task will be pre-empted and placed into the ready queue according to its priority. If no tasks are active, ET tasks are extracted from the ready queue and are (re)activated. ET tasks can pre-empt each other based on their priority.

The transmission of messages is handled in a similar way: for each node, the sending and receiving times of ST messages are stored in the schedule table; the DYN messages are organized in a prioritized ready queue. ST messages will be placed at predetermined time moments into a bus slot assigned to the sending node. DYN messages can be potentially sent during any dynamic phase and conflicts are solved by the communication controllers based on message priorities. Once the transmission of a DYN message has started, no other message will be sent on the bus until the current transmission finishes.

TT activities are triggered based on a local clock available in each processing node. The synchronization of local clocks throughout the system is provided by the communication protocol.

2.3 Application Model

We model an application as a set of task graphs. Nodes represent tasks and arcs represent communication (and implicitly dependency) between the connected tasks.

- A task can belong either to the TT or to the ET domain.
- Communication between tasks mapped to different nodes is performed by message passing over the bus. Such a message passing is modelled as a communication task inserted on the arc connecting the sender and the receiver tasks. The communication time between tasks mapped on the same node is considered to be part of the task execution time. Thus, such a communication activity is not modelled explicitly. For the rest of the paper, when referring to messages, we consider only the communication activity over the bus.
- A message can belong either to the static (ST) or the dynamic (DYN) domain.
- All tasks in a certain task graph belong to the same domain, either ET, or TT, which is called the domain of the task graph. However, the messages belonging to a certain task graph can belong to any domain (ST or DYN). Thus, in the most general case, tasks belonging to a TT graph, for example, can communicate through both ST and DYN messages.
- Each task τ_{ij} (belonging to the task graph Γ_i), has a period T_{ij}, and a deadline D_{ij} and, when mapped on node $Proc_k$, it has a worst case execution time $C_{ij}(Proc_k)$. Each ET task also has a given priority $Prio_{ij}$.
- All tasks τ_{ij} belonging to a task graph Γ_i have the same period T_i which is the period of the task graph.
- For each message we know its size (which can be directly converted into communication time on the particular communication bus). The period of a message is identical with that of the sender task. DYN messages also have given priorities.

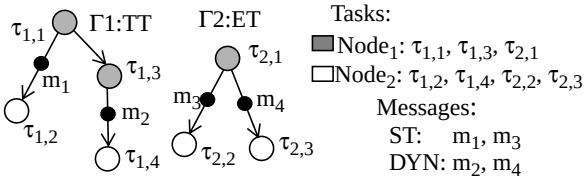

Figure 2. Application Model Example

Figure 2 shows an application modelled as two task graphs mapped on two nodes.

In order to keep the separation between the TT and ET domains, which are based on fundamentally different triggering policies, communication between tasks in the two domains is not included in the model. Technically, such a communication is implemented by the kernel, based on asynchronous non-blocking send and receive primitives (using proxy tasks if the sender and receiver are on different nodes). Such messages are typically non-critical and are not affected by hard real-time constraints.

3. Holistic Scheduling

In [13], we introduced a scheduling and schedulability analysis approach for applications as those presented in Section 2. The algorithm constructs a correct static schedule for the TT tasks and ST messages (a schedule which meets all time constraints related to these activities) and conducts the schedulability analysis in order to check that all ET tasks meet their deadlines. Two important aspects should be noticed:

1. When performing schedulability analysis for the ET tasks and DYN messages, one has to take into consideration the interference from the statically scheduled TT tasks and ST messages.
2. Among the possible correct schedules for TT tasks and ST messages, it is important to construct one which favours, as much as possible, the schedulability of ET tasks and DYN messages.

In [13], first we developed a schedulability analysis algorithm for a set of ET tasks and DYN messages, considering a fixed given static schedule of TT tasks and ST messages. Then, we introduced a method for building a valid static schedule for the TT tasks and ST messages, which favours the schedulability of ET tasks and DYN messages. This static schedule is computed over a period T_{SS} which is equal to the least common multiplier of the periods of TT task graphs.

In order to guide the static scheduling process towards "good" solutions, we use a metric which captures the "degree of schedulabil-

ity" of the ET task set. For this purpose we used a cost function similar with the one described in [15]:

$$\text{DSch} = \begin{cases} f_1 = \sum_{i=1}^{N} \sum_{j=1}^{N_i} max(0, R_{ij} - D_{ij}), \text{ if } f_1 > 0 \\ f_2 = \sum_{i=1}^{N} \sum_{j=1}^{N_i} (R_{ij} - D_{ij}), \text{ if } f_1 = 0 \end{cases}$$

- where N is the number of ET task graphs, N_i is the number of activities in the ET task graph Γ_i, and R_{ij} is the response time computed by the schedulability analysis for task τ_{ij}.

If the ET task set is not schedulable, there exists at least one task for which $R_{ij} > D_{ij}$. In this case, $f_1 > 0$ and the cost function is a metric of how far we are from achieving schedulability. If the set of ET tasks is schedulable, $f_2 \leq 0$ is used as a metric. A value $f_2 = 0$ means that the task set is "just" schedulable. A smaller value for f_2 means that the ET tasks are schedulable and a certain amount of processing capacity is still available.

4. Design Problems

Considering a hard real-time system like the one described in Section 2, several design problems emerge. There are, of course, the classical issues as selection of an architecture (e.g. number and kind of nodes), the mapping of tasks on the processing nodes, or the assignment of priorities to ET tasks and DYN messages [1, 5, 11]. However, due to the heterogeneous ET and TT nature of the application and the mixed synchronous/dynamic bus protocol, some new and very interesting problems can be identified:

- *Partitioning of the system functionality into time-triggered and event-triggered activities*. During the design process, a decision should be made on which tasks and messages will be implemented as TT/ET and ST/DYN activities, respectively. Typically, this decision is taken, based on the experience and preferences of the designer, considering aspects like the functionality implemented by the task, the hardness of the constraints, sensitivity to jitter, etc. There exists, however, a subset of tasks/messages which could be assigned to any of the domains. Decisions concerning the partitioning of this set of activities can lead to various trade-offs concerning, for example, the size of the schedule table or the schedulability properties of the system. For example, in Figure 3 we show a system with two nodes on which three tasks are mapped: τ_1 on $Node_1$, τ_2 and τ_3 on $Node_2$; τ_2 is data dependant on τ_1; worst case execution times (C_i) and deadlines (D_i) are shown in the figure. In order to keep the example simple, communication delays between τ_1 and τ_2 are ignored. When all three tasks belong to the TT domain, the system is unschedulable. In this case, either τ_2 (scheduling alternative depicted in Figure 3.a) or τ_3 (Figure 3.b) misses its deadline. If, however, τ_3 is moved into the ET domain (Figure 3.c), all tasks are schedulable (in this case, τ_2 will pre-empt the execution of τ_3).

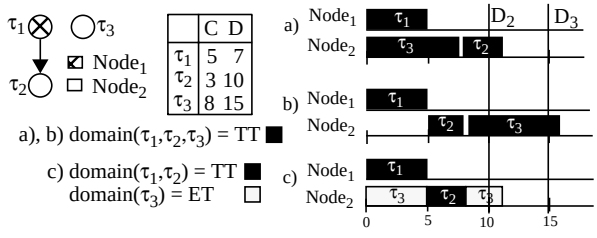

Figure 3. Partitioning into TT/ET domains

- *Determining the optimal structure of the bus access cycle*. The configuration of the bus access cycle has a strong impact on the global performance of the system. The parameters of this cycle have to be optimized such that they fit the particular application and the timing requirements. Parameters to be optimized are the number of static and dynamic phases during a communication cycle, as well as the length and order of these phases. Considering the static phases, parameters to be fixed are the order, number, and length of slots assigned to the different nodes. For example, consider the situation in Figure 4, where task τ_1 is mapped on node N_1 and sends a message m to task τ_2 which is mapped on node N_2. In case a), task τ_1 misses the start of the ST $Slot_1$ and, therefore, message m will be sent during the next bus cycle, causing the receiver task τ_2 to miss its deadline D_2. In case b), the order of ST slots inside the bus cycle is changed, the message m will be transmitted earlier and τ_2 will meet its deadline. The resulted situation can be further improved, as it can be seen in Figure 4.(c), where task τ_2 finishes even earlier, if the first DYN phase in the bus cycle can be eliminated without producing intolerable delays of the DYN messages (which have been ignored in this example).

The optimization problems identified above can be approached once we have solved the holistic scheduling problem outlined in Section 3 and presented in [13]. In the following section, we discuss a heuristic aiming at such a global optimization.

5. Design Heuristic

We consider a system specification and an architecture as described in Section 2. We also consider that some of the tasks are already mapped to nodes and their domain (TT or ET) is fixed. This can be the result of decisions already taken by the designer or/and because part of the functionality is inherited from previous generations of the product. However, we assume that there are tasks which are not mapped yet and some of the task graphs are not yet partitioned between the two domains. We denote with Ψ^P the set of all tasks which are not yet assigned to any of the ET or TT domains and with Ψ^M the set of all tasks which are not mapped to any node. Note that $\Psi^P \cap \Psi^M$ may be not-empty, which means that some tasks have neither a fixed domain, nor are they mapped on any node. The tasks in the set $\Psi^P \cup \Psi^M$ are those to which we refer in the rest of this paper when we discuss mapping and partitioning. None of the other tasks is affected, in terms of partitioning and mapping, by any of the design decisions. Our goal is threefold:

1. to partition the task set Ψ^P among the ET and TT domains;
2. to map the tasks in the set Ψ^M onto the nodes in the architecture;
3. to optimize the parameters of the communication protocol.

The above design tasks have to be performed with the overall goal that the timing constraints of the resulted system are satisfied. If this is achieved, we say that we have obtained a schedulable im-

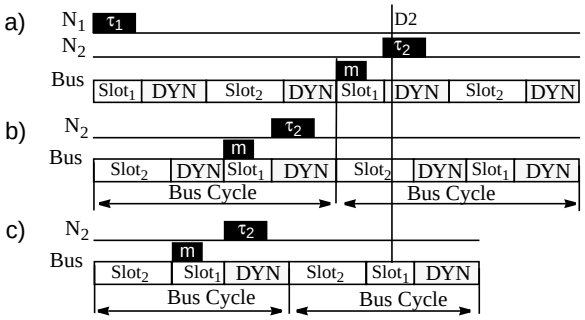

Figure 4. Optimization of Bus Access Cycle

```
01 Gen_Part, Gen_Map, Gen_Bus_Cycle
02 if TT not schedulable then
03    change partitioning (TT to ET)
04    change mapping
05    change bus cycle
06 endif
07 if TT not schedulable then stop endif
08 if ET not schedulable then
09    Mapping_and_Partitioning
10    if ET not schedulable then
11       Optimize_Bus_Access
12    endif
13 endif
```

Figure 5. Overview of the Design Heuristic

plementation of the system, which implies that the following two conditions are satisfied:

1) The tasks in the TT domain are schedulable, meaning that we were able to build a static schedule (Section 3) for all the tasks in the TT partition such that their deadlines are satisfied;

2) The tasks in the ET domain are schedulable. This means that after running the scheduling process described in Section 3, the function *DSch*, expressing the schedulability degree of the ET activities, will have a value $DSch \leq 0$.

Before starting to discuss the actual heuristics, some further observations have to be made. According to the application model presented in Section 2.3, all tasks in a task graph belong to the same domain. Thus, the task set Ψ^P contains complete task graphs and, by deciding on the partitioning of a certain task, the whole task graph is assigned to either the TT or ET domain.

A similar partitioning problem, as formulated above for tasks, could be also defined at the level of messages: considering a set of messages, for each message it has to be decided if it should be transmitted in an ST phase (statically scheduled) or in a DYN phase (dynamically scheduled). In order to keep the presentation reasonably simple and given the space limitations, in this paper we consider that all messages are preassigned as ST or DYN. For the same reason, we also consider that all tasks in the set Ψ^P have a pre-assigned priority which is used if the task is assigned to the ET domain.

The design problem outlined above is a combination of subproblems, each of exponential complexity. Therefore, we have elaborated a design space exploration strategy based on the application of several heuristics in three successive steps, as shown in Figure 5:

1. The first step (lines 01-06) starts by generating an initial mapping, partitioning and bus structure, using several basic criteria (line 01). If this initial solution is not schedulable, successive transformations are applied to the partitioning, mapping, and the bus cycle, with the aim of finding a solution such that the TT tasks are schedulable. This is performed by generating configurations (in terms of partitioning, mapping and bus cycle) which are more and more favourable to the TT partition.
   The first step is stopped once a solution with a schedulable TT partition has been reached. If at the end of the first step no such solution has been found, we conclude that, given the amount of available resources, no correct implementation of the system can be generated. This decision is justified by the fact that, if under the most favorable conditions no static schedule could be generated for the TT tasks, no further design transformations could lead to a globally schedulable solution, except for modifications of the underlying system architecture (e.g adding a new node, replacing a node with a faster one or a similar replacement of the bus). If the configuration generated after the first step is not globally schedulable, but a correct schedule of the TT tasks and ST messages has been found, the heuristic moves into the second step.
2. During the second step (line 09), a partitioning and mapping algorithm tries to produce a solution such that not only the TT static schedule is correct, but also the degree of schedulability of the ET partition is as good as possible. The cost function driving the design space exploration during this step is *DSch* (see Section 3). Simultaneously with each partitioning and mapping decision, also the bus cycle is modified in order to fit the new configuration.
3. If the second step did not succeed in producing a schedulable ET partition, the third step (line 11) tries to further improve the degree of schedulability by an aggressive optimization of the bus cycle.

In the following subsections we further elaborate on the optimization steps outlined above.

5.1 The first step: Building an initial configuration

The first step starts with generating, based on a very simple and fast heuristic, a mapping and partitioning of the tasks, as well as a bus cycle (line 01 in Figure 5):

- The partitioning is performed with the only constraint to evenly distribute the load between the TT and the ET domains.
- The mapping is based on a very fast heuristic aimed at minimising inter-processor communication while keeping a balanced processor load.
- The initial bus cycle is constructed in the following two steps:
  1. The ST slots are assigned in order to the nodes such that $Node_i$ transmits during $Slot_i$ (Figure 1). The length of $Slot_i$ is set to a value which is equal to the length of the largest ST message generated by a task mapped on $Node_i$. Considering an architecture of 4 nodes, a structure like the one in Figure 6.(a) is produced after this step.
  2. Dynamic phases are introduced in order to generate a mixed ST/DYN bus cycle. We start from the rough assumption that the total length of the dynamic phases over a period T_{SS} (T_{SS} is the length of the static schedule, see Section 3) is equal to the total length of the DYN messages transmitted over the same period, which is:

$$\sum_{m_i \in DYNdomain} \frac{T_{SS}}{T_i} \cdot L_i ,$$

  where T_i and L_i are the period and the length (expressed in time units) of the DYN message m_i. We set the length of a DYN phase to the length of the largest DYN message L_{DYN}^{max} The number n of dynamic phases in each cycle can be determined from the following equation:

$$\frac{T_{SS}}{L_{ST} + n \cdot L_{DYN}^{max}} \cdot n \cdot L_{DYN}^{max} = \sum_{m_i \in DYNdomain} \frac{T_{SS}}{T_i} \cdot L_i ,$$

  where L_{ST} is the total length of the static slots in a bus cycle and $L_{ST} + n\, L_{DYN}^{max}$ is the length of the bus cycle. Finally, the dynamic phases are evenly distributed inside the bus cycle. Figure 6.(b) illustrates such an initial bus configuration.

Once we have decided on the above configuration, we can run the holistic scheduling algorithm, which will lead to one of the following outcomes:

a) the system is found schedulable;

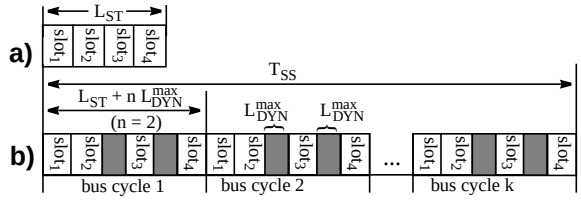

Figure 6. Initial Bus Configuration

b) the TT activities are schedulable but the ET ones are not (a valid static schedule has been built but the analysis has identified at least one ET activity for which $R_{ij} > D_{ij}$);

c) the ET activities are schedulable but the TT ones are not;

d) both ET and TT activities are not schedulable.

In the first case, the design goal has been achieved and, therefore, no further optimizations are performed. In the cases c) and d), we perform the following successive operations, aimed at achieving a schedulable TT domain (lines 03-05 in Figure 5):

1) Task graphs are moved, one by one, from the TT to the ET domain, until either the remaining TT activities are schedulable or there are no more task graphs to be moved (whole task graphs are moved and not individual tasks, because, as mentioned earlier, all tasks in a task graph belong to the same domain). The order in which task graphs are moved is based on a priority function that captures the mobility of tasks in the graph:

$$RelAvgMob(\Gamma_i) = \frac{1}{n_i} \cdot \sum_{j=1}^{n_i} \frac{D_{ij} - ASAP_{ij}}{C_{ij}}$$

where n_i is the number of tasks in the task graph Γ_i, and $ASAP_{ij}$ is the earliest possible start time for task τ_{ij}. Task graphs with a low average relative mobility are moved first, because, in principle, they are the most difficult to be scheduled statically.

2) If the TT domain still is unschedulable, TT tasks are remapped with the goal of avoiding unbalanced node utilization by TT tasks.

3) If no schedulable TT domain has been yet produced, transformations of the bus cycles are performed such that the delays produced by ST messages are reduced. In this step, a simpler and faster version of the heuristic presented in Section 5.3 is used.

If no schedulable TT domain has been produced by the above transformations, no correct implementation can be obtained with our heuristic given the available resources. If both the TT and ET domains are schedulable, we have achieved our design goal, while in the case of an unschedulable ET domain, the heuristic is continued with the second step.

5.2 The second step: Mapping and Partitioning

The mapping and partitioning step (line 09 in Figure 5) receives as an input a configuration in which the TT activities are schedulable and the ET ones are not. The algorithm is illustrated in Figure 7. It selects iteratively tasks $\tau_{ij} \in \Psi^P \cup \Psi^M$ (line 03) in order to be remapped and/or repartitioned. The order in which tasks are processed is defined by the following two rules, similar to those used in list scheduling:

1. τ_{ij} is selected only after all its predecessors in the task graph Γ_i have already been processed (these tasks are called ready).
2. Among the ready tasks, the selection is based on a priority function PF similar to the one proposed by us in [14] (line 03). This function is based on a critical path metric and it also takes into consideration the delay introduced by message passing considering the particular communication protocol, as well as the nature of the messages (ST or DYN).

Once a task τ_{ij} has been selected, its mapping and domain will be decided in a greedy fashion. If $\tau_{ij} \in \Psi^M$ (the task mapping is not fixed), it will be successively mapped to each node (lines 06-15) and for each alternative, the schedulability analysis (Section 3, [13]) returns the cost *DSch*, which captures the degree of schedulability of the produced configuration. If the domain, ET or TT, is also to be decided ($\tau_{ij} \in \Psi^P$), both alternatives are evaluated (line 08). This is performed using the function `partition` (lines 25-29). Finally, that node and domain are selected for τ_{ij} which produce the smallest value for *DSch*. If only the domain of τ_{ij} is to be decided, but the mapping is fixed, the best of the two alternatives is selected (line 19). It should be mentioned that a mapping or

```
01 while (ΨP∪ΨM ≠ ∅ and BestCost > 0) do
02    update priority function PF
03    select task τij∈ ΨP∪ΨM with highest PF
04    BestCost = ∞
05    if τij∈ ΨM then                          -- task τij is not mapped
06      for (p = 1 to NrNodes)do
07        map τij on Nodep and adjust bus access cycle
08        if τij∈ ΨP then Cost,d = partition(τij) -- τij is not partitioned
09        else Cost = DSch; d = domain of τij
10        endif
11        if BestCost > Cost then
12          BestCost = Cost; BestDomain = d;
13          BestNode = p; BestCycle = BusCycle;
14        endif
15      endfor
16    else
17      BestNode = Node on which τij is mapped
18      BestCycle = BusCycle;
19      BestCost, BestDomain = partition(τij);
20    endif
21    τij.node = BestNode; BusCycle=BestCycle;
22    set domain(Γi) to BestDomain
23    ΨP=ΨP \ {τij}; ΨM = ΨM \ {τij}
24 end while
25 function partition(τij)
26    τij.domain = ET; Cost1 = DSch; d1 = ET;
27    τij.domain = TT; Cost2 = DSch; d2 = ET;
28    return min(Cost1, Cost2) and associated di
29 end partition
```

Figure 7. Mapping and Partitioning algorithm

partitioning alternative is considered only if, with the resulted configuration, the TT domain is still schedulable (this aspect is not captured in Figure 7).

Whenever the mapping of a task is modified, the bus cycle has to be adjusted so that it can ensure the minimum requirements for transmitting the messages (for example, in line 07). Such an adjustment of the bus access cycle is illustrated in Figure 8, where 4 TT tasks are mapped on 3 nodes (N_1, N_2 and N_3). The number at the side of each message represents its length. Tasks mapped on different nodes communicate through ST messages and an ST slot should be able to accommodate the longest message transmitted by the associated node. The figure shows how the lengths of the slots associated with N_1 and N_2 are modified after a task has been remapped. In one case, task τ_2 is moved from N_2 to $N_1$and therefore, the message $m_{1,2}$ will disappear (τ_1 and τ_2 are both mapped on N_1), while message $m_{2,4}$ will be transmitted in $Slot_1$ instead of $Slot_2$. In the second case, τ_3 is moved from N_2 to N_1, which means that $m_{1,3}$ disappears, while $m_{3,4}$ is transmitted in $Slot_1$.

5.3 The third step: Optimization of the bus cycle

It may be the case that even after the mapping and partitioning step, some ET activities are still not schedulable. In the third step (lines 10-12, Figure 5), our algorithm tries to remedy this problem by changing the parameters of the bus cycle, like ST slot lengths and order, as well as the number, length and order of the ST and DYN phases. The goal is to generate a bus access scheme which is adapt-

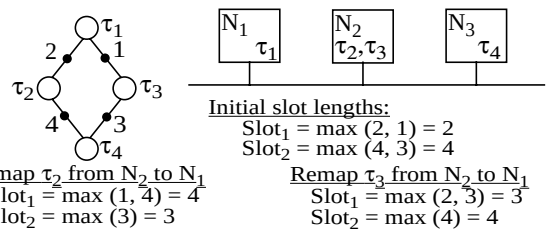

Figure 8. Adjustment of the Bus Access Cycle

ed to the particular task configuration. The heuristic is illustrated in Figure 9. The algorithm iteratively looks for the right place and size of $Slot_i$ used for transmission of ST messages from $Node_i$ (outermost two loops). $Slot_i$ is swapped with all the slots of higher order (line 03), and all alternative lengths (lines 04-05) of $Slot_i$ larger than its minimal allowed length (which is equal to the length of the largest ST message generated by a task mapped on $Node_i$) are considered. For any particular length and position of $Slot_i$, alternative lengths of the adjacent ET phase Ph_i are considered (innermost loop). For each alternative, the schedulability analysis evaluates cost *DSch*, and the solution with the lowest cost is selected. If *DSch* ≤ 0, the system is schedulable and the heuristic is stopped..

It is important to notice that the possible length π of an ET phase (line 6) includes also the value 0. Therefore, in the final bus cycle, it is not needed that each static slot is followed by a dynamic phase (see also Figure 1). Dynamic phases introduced as result of the previous steps can be eliminated by setting the length to π=0 (such a transformation is illustrated in Figure 4.c). It should be also mentioned that enlarging a slot/phase can increase the schedulability by allowing several ST/DYN messages to be transmitted quickly immediately one after another. At the same time, the following slots are delayed, which means that ST messages transmitted by nodes assigned to upcoming slots will arrive later. Therefore, the optimal schedulability will be obtained for slot and phase lengths which are not tending towards the maximum. The number of alternative slot and phase lengths to be considered by the heuristic in Figure 9 is limited by the following two factors:

1. The maximum length of a static slot or dynamic phase is fixed by the technology (e.g. 32 or 64 bits).
2. Only frames consisting of entire messages can be transmitted, which excludes several alternatives.

6. Experimental Results

In order to evaluate the proposed heuristic, we have generated a large set of applications with different characteristics. All experiments were run on an AMD Athlon 850 MHz PC. For our first experiments we considered an architecture consisting of 6 nodes. We have generated 4 sets of applications composed of 60, 75, 90, and 120 tasks respectively. Each set consists of 40 applications. The number of unmapped tasks was between 10 and the total number of tasks in the application. 10 task graphs are considered to be unassigned to any of the two domains (ET and TT). The average load on the processors is 60%. Figure 10 shows the percentage of schedulable applications obtained after the successive steps of our heuristic. By straight forward configuration we mean the mapping, partitioning and bus cycle generated at the start of step 1 (line 01 in Figure 5).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

```
01 for i = 1 to NrNodes
02     for j = i to NrNodes
03       swap Slot_i with Slot_j
04       for all slot lengths λ > min_len(Slot_i)
05         len(Slot_i) = λ
06         for all DYN phase lengths π do
07           len(Ph_i) = π
08           if DSch ≤ 0 then stop endif
09           keep solution with smallest DSch
10         end for
11       end for
12       swap back Slot_i and Slot_j
13     end for
14     bind best position and length of Slot_i
15     bind length of Ph_i
16 end for
```

Figure 9. Bus Access Optimization

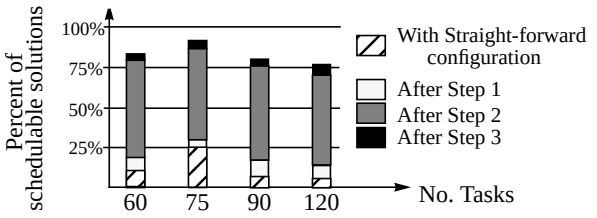

Figure 10. Percentage of Schedulable Applications

consisting of 60 tasks, for example, only 10% were schedulable with the straight-forward configuration and 90% continued the optimization process. 9% of the total number of tasks have been found schedulable with the configuration generated by step 1. As expected, the mapping, partitioning and bus cycle adjustment performed in step 2 are leading to a huge improvement, adding 61% of the total number of applications to the group of schedulable ones. An additional 4% of the total number of applications is found schedulable after performing the bus optimization in step 3. A similar trend can be followed in the experiments with 75, 90 and 120 tasks. It is easy to observe that by performing the proposed optimizations, huge improvements over the straight-forward configuration could be produced.

An interesting question is to what extent the partitioning of tasks into the ET and TT domains is contributing to the results illustrated in Figure 10. Or, ar these results mostly due to the optimized mapping? The same question can also be put relative to the bus cycle optimization. In order to answer these questions, we considered a second set of applications consisting of 60, 80 and 100 tasks grouped into 12, 15, 18 or 20 task graphs and mapped on 4 or 6 nodes. We have run our heuristic for each of these applications considering four cases. First, with a subset of tasks that have to be partitioned but no tasks to be mapped ($|\Psi^M| = 0$). Second, with the same subset of tasks open for mapping but not for partitioning ($|\Psi^P| = 0$). The third case does not allow any bus access optimization, so we switched off the optimizations in lines 5 and 11 in Figure 5 (however, we keep the bus cycle adjustment which is needed in Step 2, line 7 in Figure 7). The fourth case represents the reference, the complete heuristic. The results are presented in Figure 11, which shows the percentage of schedulable applications (relative to the total number of applications) that have been produced by each optimization step. For example, after step 2, 45% additional applications were schedulable if we only allow to perform re-mapping ($|\Psi^P| = 0$), as opposed to 74% in the case when both optimizations are performed. The same number is 40% if we only allow to perform re-partitioning ($|\Psi^M| = 0$). The percentage of unschedulable tasks after the three steps is 34% when $|\Psi^P| = 0$, 44% for $|\Psi^M| = 0$, and 24% when no bus optimization was performed, as compared to 7% in the case of the complete heuristic. The conclusions which we can draw are the following:

1. An efficient partitioning into the ET and TT domains is contributing essentially to the overall optimization, to an extent comparable to the mapping.

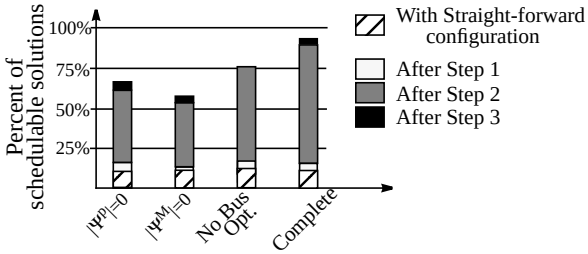

Figure 11. Partial Optimization

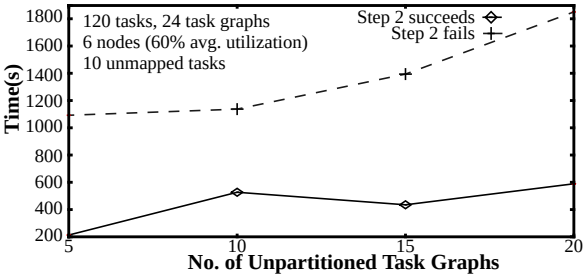

Figure 12. Runtime of step 2 function of $|\Psi^P|$

2. When applied together, the three techniques provide much better results than the ones obtained when any of the techniques is eliminated.

Concerning the runtime needed for the optimization process, we have analyzed each of the three steps separately. For the examples leading to the results in Figure 10, the average run time for step 1 was 11.7s (60 tasks), 40.1s (75 tasks), 73.2s (90 tasks), and 150s (120 tasks).

The execution time for step 2 is presented in more detail in Figure 12 and Figure 13. Figure 12 illustrates the time needed for step 2 as a function of the total number of task graphs to be partitioned (the characteristics of the applications and the number of nodes are shown in the figure). The upper curve illustrates the average execution times for those applications which are running through step 2 without reaching a system configuration which makes them schedulable. This curve can be considered as an upper bound for the execution time in step 2. The second curve in Figure 12 gives the average execution times of those applications that have been found schedulable during step 2.

In Figure 13, we show, in a similar way, the average execution times as a function of the number of unmapped tasks. The execution times needed for the third optimization step are given in Figure 14. As this step is concentrating only on the communication aspect, the average execution time is given as a function of the number of nodes.

Finally, we considered a real-life example from the automotive area, implementing a vehicle cruise controller and a control application related to the Anti Blocking System (ABS) on an architecture consisting of 5 nodes. The cruise controller consists of 42 tasks organized in 11 task graphs. One of these task graphs is fixed into the TT domain, and the other 10 are unpartitioned. 10 out of the 42 tasks are unmapped. The ABS system consists of 35 tasks already mapped over the 5 nodes and assigned to the ET domain. Running our optimization heuristic, step 1 was able to generate a correct static schedule for the TT domain, but without producing a globally schedulable system. Step 2 manages to improve the degree of schedulability of the system (function *DSch*) by two orders of magnitude without, however, producing a schedulable system. A correct

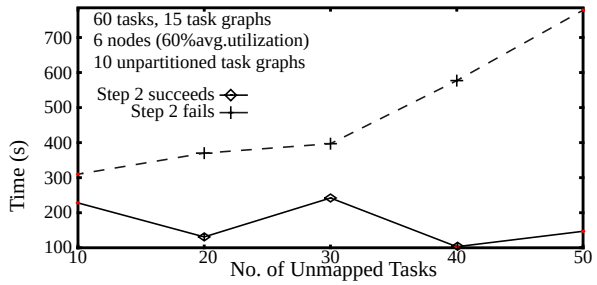

Figure 13. Runtime of step 2 function of $|\Psi^M|$

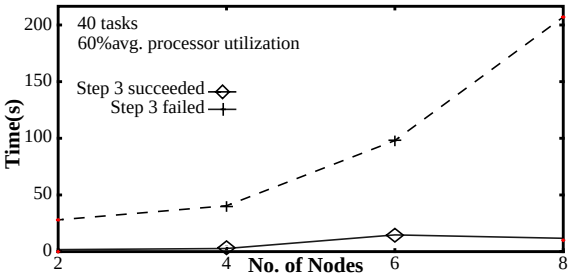

Figure 14. Runtime of step 3

implementation has been produced after the bus optimization in step 3. It is interesting to mention that for the final schedulable solution, out of the 10 unpartitioned task graphs, 2 were assigned to the ET and 8 to the TT partition. The run times for the three optimization steps were 5.3s, 708s and 164s respectively.

7. Conclusions

Distributed embedded systems based on mixed static/dynamic communication protocols are becoming the new standard for automotive applications. Such systems typically run applications consisting of both ET and TT tasks. We have identified a new class of system optimization issues typical for the heterogeneous systems considered in the paper: partitioning of the system functionality into TT and ET domains and the optimization of the bus access scheme. Both problems were considered in the context of a heuristic which performs the mapping and scheduling of the system functionality. We have shown that the quality of the system implementation can be significantly improved by the proposed optimization heuristics.

8. References

[1] N. Audsley, K. Tindell, A. et. al., “The End of Line for Static Cyclic Scheduling?”, 5th Euromicro Works. on Real-Time Systems, 1993.

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

[3] R. Bosch GmbH, “CAN Specification Version 2.0”, 1991.

[4] T. Demmeler, P. Giusto, “A Universal Communication Model for an Automotive System Integration Platform”, DATE, 2001.

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

[6] FlexRay homepage: http://www.flexray-group.com/.

[7] H. Kopetz, “Real-Time Systems - Design Principles for Distributed Embedded Applications”, Kluwer Academic Publisher, 1997.

[8] H. Lönn, J. Axelsson, “A Comparison of Fixed-Priority and Static Cyclic Scheduling for Distributed Automotive Control Applications”, Euromicro Conf. on Real-Time Systems, 1999.

[9] P. Pedreiras, L. Almeida, “Combining Event-Triggered and Time-Triggered Traffic in FTT-CAN: Analysis of the Asynchronous Messaging System”, WFCS, 2000.

[10] P. Raja, G. Noubir, “Static and Dynamic Polling Mechanisms for Fieldbus Networks”, ACM Operating Systems Review, 27(3), 1993.

[11] W. Wolf, “Hardware-Software Co-Design of Embedded Systems”, *Proceedings of the IEEE*, V82, N7, 1994.

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

[13] T. Pop, P. Eles, Z. Peng, “Schedulability Analysis for Distributed Heterogeneous Time/Event-Triggered Real-Time Systems”, EuroMicro Conf. on Real-Time Systems, 2003.

[14] P., A. Doboli, P. Pop, Z. Peng, “Scheduling with Bus Access Optimization for Distributed Embedded Systems”, IEEE Transactions on VLSI Systems, 8(5), 472-491, 2000.

[15] P. Pop, P. Eles, Z. Peng, “Bus Access Optimization for Distributed Embedded Systems based on Schedulability Analysis”, DATE, 2000.