Flexibility Driven Scheduling and Mapping for Distributed Real-Time Systems

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

Abstract

In this paper we present an approach to mapping and scheduling of distributed hard real-time systems, aiming at improving the flexibility of the design process. We consider an incremental design process that starts from an already existing system running a set of applications, with preemptive priority based scheduling at the process level, and time triggered static scheduling at the communication level. We are interested to implement new functionality so that the already running applications are disturbed as little as possible and there is a good chance that, later, new functionality can easily be added to the resulted system. The mapping and scheduling problems are considered in the context of a realistic communication model based on a TDMA protocol. Extensive experiments as well as a real life example demonstrate the relevance of this problem and the efficiency of our solutions.

1. Introduction

In this paper we concentrate on scheduling and mapping of hard real-time systems which are implemented on distributed architectures. Process scheduling is based on a static priority preemptive approach while the bus communication is statically scheduled.

Preemptive scheduling of independent processes with static priorities running on single processor architectures has its roots in [7]. The approach has later been extended to accommodate more general computational models and has also been applied to distributed systems [17]. The reader is referred to [1, 2, 13] for surveys on this topic. In many of the previous scheduling approaches researchers have assumed that processes are scheduled independently. However, this is not the case in reality, where process sets can exhibit both data and control dependencies. One way of dealing with data dependencies between processes with static priority based scheduling has been indirectly addressed by the extensions proposed for the schedulability analysis of distributed systems through the use of *release jitter* [17].

Currently, more and more real-time systems are used in physically distributed environments and have to be implemented on distributed architectures in order to meet reliability, functional, and performance constraints. We have to mention here some results obtained in extending real-time schedulability analysis so that network communication aspects can be handled. In [16], for example, the CAN protocol is investigated while the work reported in [4] considers systems based on the ATM protocol. Analysis for a simple TDMA protocol is provided in [17], which integrates processor and communication schedulability and provides a "holistic" schedulability analysis in the context of distributed real-time systems. The problem of how to allocate priorities to a set of distributed tasks is discussed in [5], based on the schedulability analysis from [17].

Another characteristic of research efforts concerning the codesign of real-time systems is that researchers concentrate on the design, from scratch, of a new system optimized for a particular application. For many application areas, however, such a situation is extremely uncommon and only rarely appears in design practice. It is much more likely that one has to start from an already existing system running a certain application and the design problem is to implement new functionality (including also upgrades to the existing one) on this system [3]. In such a context it is very important to make as few as possible modifications to the already running applications. The main reason for this is to avoid unnecessarily large design and testing times. Performing modifications on the (potentially large) existing applications increases design time and, even more, testing time (instead of only testing the newly implemented functionality, the old application, or at least a part of it, has also to be retested). However, this is not the only aspect to be considered. Such an incremental design process, in which a design is periodically upgraded with new features, is going through several iterations. Therefore, after new functionality has been implemented, the resulting system has to be structured such that additional functionality, later to be mapped, can easily be accommodated.

We consider mapping and scheduling for hard real-time systems in the context of a realistic communication model. Because our focus is on hard real-time safety critical systems, communication is based on a time division multiple access (TDMA) protocol, the time-triggered protocol (TTP), as recommended for applications in areas like, for example, automotive electronics [6].

In this paper, we have considered the design of distributed embedded systems in the context of an incremental design process as outlined above. This implies that we perform mapping and scheduling of new functionality so that certain design constraints are satisfied and:

a) the existing applications are disturbed as little as possible;

b) there is a good chance that new functionality can, later, be easily mapped on the resulted system.

In [11] we have discussed an incremental design strategy addressed to systems where *both* processes and messages are statically scheduled. However, considering preemptive priority based scheduling at the process level, with time triggered static scheduling at the communication level, can be the right solution under certain circumstances [8]. A communication protocol like TTP provides a global time base, improves fault-tolerance and predictability. At the same time, certain particularities of the application or of the underlying real-time operating system can impose a priority based scheduling policy at the process level.

In this paper we extend our approach to hard real-time systems where process scheduling is based on a static priority preemptive approach while the bus communication is statically scheduled. We accurately take into consideration overheads due to communication and consider, during the mapping and scheduling process, the particular requirements of the communication protocol. We propose a new heuristic, together with the corresponding design criteria, which finds the set of old applications which have to be remapped at the same time with the new one such that the disturbance on the running system (expressed as the total cost implied by the modifications) is minimized. Once this set of applications has been determined, mapping and scheduling is performed according to the requirements a) and b) stated above. Supporting such a design process is of critical importance for current and future industrial practice, as the time interval between successive generations of a product is continuously decreasing, while the complexity due to increased sophistication of new functionality is growing rapidly.

The paper is divided into 6 sections. The next section presents the architectures considered for system implementation, the computational model assumed together with the schedulability analysis employed, and the process allocation problem. Section 3 introduces the detailed problem formulation and the quality metrics we have defined. The mapping and scheduling strategy is presented in section 4, and the approaches are evaluated in section 5. The last section presents our conclusions.

2. Preliminaries

2.1 System Architecture

We consider architectures consisting of nodes connected by a broadcast communication channel. Every node consists of a TTP controller, a CPU, a RAM, a ROM and an I/O interface to sensors and actuators.

Communication between nodes is based on the TTP [6]. TTP was designed for distributed real-time applications that require predictability and reliability (e.g., drive-by-wire). The communication channel is a broadcast channel, so a message sent by a node is received by all the other nodes. The bus access scheme is time-division multiple-access (TDMA) (Figure 1). Each node N_i can transmit only during a predetermined time interval, the so called TDMA slot S_i. In such a slot a node can send several messages packaged in a frame. A sequence of slots corresponding to all the nodes in the architecture is called a TDMA round. A node can have only one slot in a TDMA round. Several TDMA rounds can be combined together in a cycle that is repeated periodically. The sequence and length of the slots are the same for all TDMA rounds. However, the length and contents of the frames may differ.

Every node has a TTP controller that implements the protocol services and runs independently of the node's CPU. The TDMA access scheme is imposed by a so called message descriptor list (MEDL) that is located in every TTP controller. MEDL serves as a schedule table for the TTP controller which has to know when to send or receive a frame to or from the communication channel. The TTP controller provides each CPU with a timer interrupt based on a local clock synchronized with the local clocks of the other nodes.

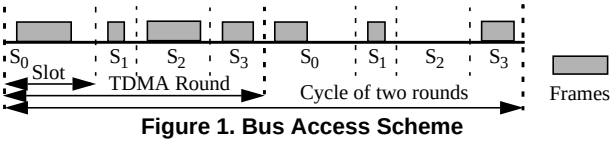

Figure 1. Bus Access Scheme

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. For the exact details of the software architecture and how it is taken into account by the schedulability analysis the reader is directed to [10].

2.2 Computational Model and Schedulability Analysis

We model an application as a set of processes. A process P_i has a period T_i, a deadline D_i, and a uniquely assigned priority. Each process P_i can potentially be mapped on several nodes. Let N_{P_i} be the set of nodes to which P_i can potentially be mapped. For each $N_i \in N_{P_i}$, we know the worst case execution time $C_{P_i}^{N_i}$ of process P_i, when executed on N_i. We consider a preemptive execution environment, which means that higher priority processes can interrupt the execution of lower priority processes. In this paper we use a deadline monotonic priority assignment scheme, but any other more advanced priority schemes like the ones in [5, 14] could as well be used. A lower priority process can block a higher priority process (e.g., it is in its critical section), and the blocking time is computed according to the priority ceiling protocol [13]. Processes exchange messages, and for each message m_i we know its size S_{m_i}. A message is sent once in every n_m invocations of the sending process, with a period $T_m = n_m T_i$ inherited from the sender process P_i, and has a unique destination process.

For a given mapping of processes to processors, Tindell et al. integrate in [17] processor and communication scheduling and provide a "holistic" schedulability analysis in the context of distributed real-time systems with communication based on a simple TDMA protocol. The basic idea in [17] is that the release jitter of a destination process depends on the communication delay between sending and receiving a message. The release jitter of a process is the worst case delay between the arrival of the process and its release (when it is placed in the run-queue for the processor). The communication delay is the worst case time spent between sending a message and the message arriving at the destination process.

Although there are many similarities with the general TDMA protocol, the analysis in the case of TTP is different in several aspects and also differs to a large degree depending on the policy chosen for message scheduling. In [10] we have proposed four approaches for scheduling of messages using TTP that differ in the way the messages are allocated to the communication channel (either statically or dynamically) and whether they are split or not into packets for transmission. For each of these approaches, we have also developed the corresponding schedulability analysis [10].

The first approach, called Static Single Message Allocation (SM), is to statically (off-line) schedule each of the messages into a slot of the TDMA cycle, corresponding to the node sending the message. We also consider that the slots can hold each at maximum one single message. The second approach, called Static Multiple Message Allocation (MM), is an extension of the first one. In this approach we allow more than one message to be statically assigned to a slot and all the

messages transmitted in the same slot are packaged together in a frame. While the previous two approaches have statically allocated one or more messages to their corresponding slots, the third approach, called Dynamic Message Allocation (DM), considers that the messages are dynamically allocated to frames, as they are produced. Finally, the last approach, called Dynamic Packets Allocation (DP), is an extension of the previous one, and allows the messages to be split into packets before they are transmitted on the communication channel. By splitting messages into packets we can obtain a higher utilization of the bus and reduce the release jitter.

Comparing these four approaches, in [10] we conclude that while the DP approach is generally the most performant since the dynamic scheduling of messages is able to reduce release jitter because no space is waisted in the slots if the packet size is properly selected, by using the MM approach we can obtain almost the same result if the messages are carefully allocated to slots. Moreover, in the case of larger process sets MM outperforms DP, as DP suffers from large overhead due to its dynamic nature. Also, DM performs worse than DP and MM because it does not split the messages into packets, and this results in a mismatch between the size of the messages dynamically queued and the slot size, leading to unused slot space that increases the jitter. SM performs the worst as it does not permit much room for improvement, leading to large amounts of unused slot space.

Therefore, for the purpose of this paper, we consider that the messages are scheduled using the MM approach, and for the details of the corresponding schedulability analysis the reader is referred to [10]. The discussion can easily be extended to any of the other three message passing approaches presented before.

2.3 Application Mapping and Scheduling

In order to implement an application, represented as a set of processes, the designer has to map the processes to the system nodes and generate the schedule table for the messages (MEDL) such that all deadlines are satisfied. But producing a mapping and scheduling so that the system is schedulable is not enough if we are to support an incremental design process as discussed in the introduction. In this case, starting from a schedulable system, we have to improve the mapping of processes and scheduling of messages so that not only the design constraints are satisfied, but there is also a good chance that, later, new functionality can easily be mapped on the resulted system.

To illustrate the role of mapping and scheduling in the context of an incremental design process, let us consider the example in Figure 2, where we have two processors with the same speed connected by a TTP bus. With black we represent the set of already running applications ψ while the current application $\Gamma_{current}$ to be mapped and scheduled is represented in grey and consists of two processes and three messages. To simplify the discussion, for this particular example we consider that the system is not schedulable if the *utilization factor* of any node is greater than one.We say that the processor can be "filled up" with processes until it reaches an utilization factor of one (the square depicting the processor is full). The utilization factor U_i of a process P_i is the ratio between the worst case execution time C_{P_i} of that process and its period T_i: $U_i = C_{P_i} / T_i$. The utilization factor of a node is the sum of the utilization factors of all processes mapped on that node. The processes and messages that are to be mapped on the processors are depicted as blocks. The height of a process block is equal with its utilization factor, while the length of a message block gives the size of the message. White space on a processor represents available utilization, while white space on the bus represents available slack in the schedule table.

Now, let us suppose that in the future another application, Γ_{future}, has to be mapped on the system. Γ_{future} consists of two processes and two messages represented as hashed blocks.

We can observe that the new application can be scheduled only in the third case, presented in Figure 2c. If $\Gamma_{current}$ has been implemented as in Figure 2b, we are not able to schedule process P_2 and message m_2 of Γ_{future}. The way our current application is mapped and scheduled will influence the likelihood of successfully mapping additional functionality on the system without being forced to modify the implementation of already running applications.

3. Problem Formulation

We model an application $\Gamma_{current}$ as a set of processes as outlined in section 2.2. Thus, for each process P_i we know the set N_{P_i} of potential nodes on which it could be mapped and its worst case execution time on each of these nodes. The underlying architecture is as presented in section 2.1. We consider fixed priority preemptive scheduling for processes and a time-triggered message passing policy, as imposed by the TTP protocol (section 2.1).

Our goal is to map and schedule an application $\Gamma_{current}$ on a system that already implements a set ψ of applications, considering the following requirements:

Requirement a: constraints on $\Gamma_{current}$ are satisfied and minimal modifications are performed to the applications in ψ.

Requirement b: new applications Γ_{future} can be mapped on the resulting system.

If it is not possible to map and schedule $\Gamma_{current}$ without modifying the already running applications, we have to change the mapping and scheduling of some applications in ψ. However, even with serious modifications performed on ψ, it is still possible that certain constraints are not satisfied. In this case the hardware architecture has to be changed by, for example, adding a new processor. In this paper we will not discuss this last case, but will concentrate on the situation where a possible mapping and scheduling which satisfies requirement a) can be found, and this solution has to be further improved by considering requirement b).

In order to achieve our goals we need certain information to be available concerning the set of applications ψ as well as the possible future applications Γ_{future}. We consider that $\Gamma_{current}$ can interact with the previously mapped applications ψ by reading messages generated on the bus by processes in ψ.

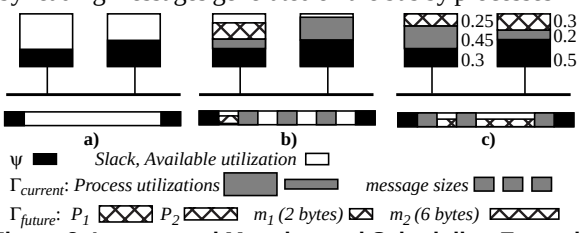

Figure 2. Incremental Mapping and Scheduling Example

3.1 Characterizing Existing Applications

To perform the mapping and scheduling of $\Gamma_{current}$, the minimum information needed on the existing applications ψ consists of the worst case execution time, period and deadline for each process on its node. As for messages, their length as well as their place in the particular TDMA frame have to be known.

If the initial attempt to schedule and map $\Gamma_{current}$ does not succeed, we have to modify the mapping of processes and schedule of messages belonging to ψ, in the hope to find a valid solution for $\Gamma_{current}$. One of the goals in this paper is to find that minimal modification to the existing system which leads to a correct implementation of $\Gamma_{current}$. In our context, such a minimal modification means remapping and rescheduling a subset of old applications $\Omega \subseteq \psi$ so that the total cost of reimplementing Ω is minimized. We represent a set of applications as a directed acyclic graph *G(V, E)*, where each node $\Gamma_i \in V$ represents an application. An edge $e_{ij} \in E$ from Γ_i to Γ_j indicates that any modification to Γ_i would trigger the need to also remap and schedule Γ_j. Such a relation can be imposed by certain interactions between applications. In Figure 3 we present the graph corresponding to a set of ten applications. Applications Γ_6, Γ_8, Γ_9 and Γ_{10}, depicted in black, are frozen: no modifications are allowed to them. The rest of the applications have the remapping cost R_i depicted on their left. Γ_7 can be remapped with a cost of 20. If Γ_4 is to be reimplemented, this also requires the modification of Γ_7, with a total cost of 90. In the case of Γ_5, although not frozen, no remapping is possible as it would trigger the need to remap Γ_6 which is frozen. Given a subset of applications $\Omega \subseteq \psi$, the total cost of modifying the applications in Ω is $R(\Omega)=\sum_{\Gamma_i \in \Omega} R_i$.

To each application $\Gamma_i \in V$ the designer has associated a cost R_i of reimplementing Γ_i. Such a cost can typically be expressed in hours needed to perform retesting of Γ_i and other tasks connected to the remapping of the application (moving processes between nodes).

3.2 Characterizing Future Applications

What do we suppose to know about the family Γ_{future} of applications which do not exist yet? Given a certain limited application area (e.g. automotive electronics), it is not unreasonable to assume that, based on the designers' previous experience, the nature of expected future functions to be implemented, profiling of previous applications, available incomplete designs for future versions of the product, etc., it is possible to characterize the family of applications which would be added to the current implementation. This is an assumption which is basic for the concept of incremental design.

Thus, we consider that, concerning the future applications, we know the set $S_U=\{U_{min},...U_i,...U_{max}\}$ of possible *processor utilization factors* for processes, and the set $S_b=\{b_{min},...b_i,...b_{max}\}$ of possible message sizes. The processor utilization factor U_i provides a measure of the computational load due to the a process P_i, and is expressed as $U_i=C_{P_i}/T_i$. The utilization factors in S_U are considered relative to the slowest node in the system. All the other nodes are characterized by a speed-up factor relative to this slowest node, which is used to calculate the actual utilization factor due to a process P_i if mapped on a node N_j. The utilization factor for an entire process set is given by $U=\sum_{i=1}^{n} U_i$.

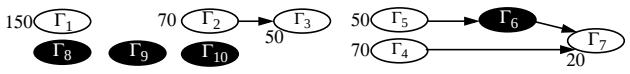

Figure 3. Characterizing the Set of Existing Applications

We also assume that over these sets we know the distributions of probability $f_{S_U}(U)$ for $U \in S_U$ and $f_{S_b}(b)$ for $b \in S_b$. For example, we might have possible utilization factors $S_U=\{0.02, 0.05, 0.1, 0.2\}$ for the future application. If almost half of the processes are assumed to have an utilization factor of 0.1, and there is a lower probability of having processes with utilization factors of 0.2 and 0.02, then our distribution function $f_{S_U}(U)$ could look like this: $f_{S_U}(0.02)=0.15$, $f_{S_U}(0.05)=0.25$, $f_{S_U}(0.1)=0.45$, $f_{S_U}(0.2)=0.15$.

Another information is related to the period of future applications. In particular, the smallest expected period T_{min} is assumed to be given, together with the expected necessary bus bandwidth b_{need} inside such a period T_{min}. As will be shown later, this information is used in order to provide a fair distribution of slacks on the bus.

For the sake of simplifying the discussion, we will not address here the memory constraints during process mapping and the implications of memory space in the incremental design process.

3.3 Quality Metrics

A designer will be able to map and schedule a Γ_{future} application on top of a system implementing ψ and $\Gamma_{current}$ only if there are sufficient resources available. In our case, the resources are the processor time and the bandwidth on the bus. In the context where processes are scheduled according to a fixed priority preemptive policy and messages are scheduled statically, having free resources translates into having enough processor capacity, and having space left for messages in the bus slots. We measure the processor capacity using the *available utilization*, while the available resources on the bus are called *slack*.

It is to be noted that the total quantity of computation and communication power available on our system after we have mapped and scheduled $\Gamma_{current}$ on top of ψ is the same regardless of the mapping and scheduling policies used. What depends on the mapping and scheduling strategy is the distribution of the available utilization on each processor, the size of the individual slacks on the bus, and the distribution of slacks along the time line. It is the distribution of available utilization and the size and distribution of the slacks that characterizes the quality of a certain design alternative. In this section we introduce the design criteria which reflect the degree to which one design alternative meets the requirement b) presented in section 3. For each criterion we provide metrics which quantify the degree to which the criterion is met. Relative to processes we have introduced one criterion which reflects how well the resulted available utilization on the nodes fits the requirements of a future application. For messages, there are two criteria. The first one reflects how well the resulted slack sizes fit a future application, and the second criterion expresses how well the slack is distributed over time.

3.3.1 Processes Related Criterion

The distribution of available utilization on the nodes, resulted after implementation of $\Gamma_{current}$ on top of ψ, should be

such that it best accommodates a given family of applications Γ_{future}, characterized by the set S_U and the probability distribution f_{S_U} as outlined before.

Let us consider the example in Figure 2, where we have two processors and the applications ψ and $\Gamma_{current}$ are already mapped. Suppose that application Γ_{future} consists of the two processes P_1 and P_2. If we schedule $\Gamma_{current}$ like in Figure 2b it is impossible to fit Γ_{future} because there is not enough available utilization on any of the processors that can accommodate process P_2. A situation as the one depicted in Figure 2c is desirable, where the resulted available utilization is such that the future application can be accommodated.

In order to measure the degree to which the available utilization in a given design alternative fits the future applications, we provide a metric, C_1^P, which captures to what extent the largest future application (considering the sum of available process utilization) could be mapped on top of the current design. This potentially largest application is determined knowing the total size of the available utilization, and the characteristics of the application: S_U and f_{S_U}. For example, if our *total* available utilization on *all* the processors is of 1.81 then we have to distribute this utilization according to the probabilities in f_{S_U}. Considering the numerical example for processes given in section 3.2, the largest application will result as having a total of 20 processes: 3 processes of utilization 0.02, 5 of 0.05, 9 processes (almost half, $f_{S_U}(0.1)$=0.45) of utilization 0.1, and 3 of 0.2. If the number of processes for a particular dimension is not an integer, then we use the ceiling. After we have determined the largest Γ_{future} we apply a *bin-packing* algorithm [9] using the *best-fit* policy in which we consider processes as the objects to be packed, and the available utilization as containers. The total utilization of unpacked processes relative to the total utilization of the process set gives the C_1^P metric. In the case presented in Figure 2b U_1=0.3 and U_2=0.25, and P_2 represents 45% of the largest possible future application. In this case C_1^P=45. However, in Figure 2c were we were able to completely map the future application C_1^P=0.

3.3.2 Criteria Related to Messages

The first criterion for messages is similar to the one defined for processes. Thus, the slack sizes in the message schedule table MEDL (see section 2.1) resulted after implementation of $\Gamma_{current}$ on top of ψ should be such that they best accommodate a given family of applications Γ_{future}, characterized by the set S_b and the probability distribution f_{S_b} for messages.

Let us consider the example in Figure 2, where we have two processors and the applications ψ and $\Gamma_{current}$ are already mapped. Application Γ_{future} has two messages m_1 and m_2. It can be observed that the best configuration, taking in consideration only slack sizes, is to have a contiguous slack. However, in reality, it is almost impossible to map and schedule the current application such that a contiguous slack is obtained. Not only is it impossible, but it is also undesirable from the point of view of the second design criterion, discussed below. On the other side, as we can see from Figure 2b, if we schedule $\Gamma_{current}$ so that it fragments too much the slack, it is impossible to fit Γ_{future} because there is no slack that can accommodate message m_2. A situation as the one depicted in Figure 2c is desirable, where the resulted slack sizes can accommodate the characteristics of the Γ_{future} application.

In order to measure the degree to which the slack sizes in a given design alternative fit the future applications, we provide the metric C_1^m. C_1^m captures how much of the communications of the largest future application which theoretically could be mapped on the system if the slacks on the bus would be summed, can be mapped on the current design alternative. The messages accounting for the largest amount of communication are determined, as shown above for processes, knowing the total size of the available slack, and the characteristics of the application: S_b and f_b.

C_1^m is calculated similarly to the metric C_1^P but, instead of packing the processes as objects, we try to pack the messages into the available slack on the bus. C_1^m is then the total size of unpacked messages, relative to the total size of messages in the largest future application. For Figure 2b, where m_2 could not be scheduled, C_1^m is 75% because m_2 of 6 bytes represents 75% of the total message sizes of 8 bytes. For the design alternative in Figure 2c C_1^m is 0% because all the messages have been scheduled.

We have just discussed a metric for how well the sizes of the slacks fit a possible future application. A similar metric is needed to characterize the distribution of slacks over time.

During implementation of $\Gamma_{current}$ we aim for a slack distribution such that the future application with the smallest expected period T_{min} and with the expected necessary bandwidth b_{need} inside the period T_{min}, can be accommodated. The minimum over the slacks inside each T_{min} period, which is available *periodically* to the messages of Γ_{future}, is the C_2^m metric.

In Figure 4 we present a message schedule table. We consider a situation with T_{min}=120 ms and b_{need}=40 ms. The length of the schedule table is 360 ms, and the already scheduled messages of ψ and $\Gamma_{current}$ are depicted in black. Let us consider the situation in Figure 4a. In the first period T_{min}, *Period 0*, there are 40 ms of slack available on the bus, in the second period 80 ms, and in the third period no slack is available. Thus, the total slack a future application with a period T_{min} can use on the bus in each period is C_2^m = min(40, 80, 0)=0 ms. In this case, the messages cannot be scheduled. However, if we move m_1 to the left in the schedule table, we are able to create, in Figure 4b, 40 ms of slack in each period, resulting a C_2^m = 40 ms = b_{need}.

3.4 Cost Function and Exact Problem Formulation

In order to capture how well a certain design alternative meets the requirement b) stated at the beginning section 3, the metrics discussed before are combined in a cost function, as follows: $C = w_1^P (C_1^P)^2 + w_1^m (C_1^m)^2 + w_2^m max(0, b_{need} - C_2^m)$, where the metric values are weighted by the constants w_i. Our mapping and scheduling strategy will try to minimize this function.

The first two terms measure how well a future application fits to the available utilization on the processors and slack sizes on the bus, respectively. In order to obtain a balanced solution, that favours a good fitting both on the processors

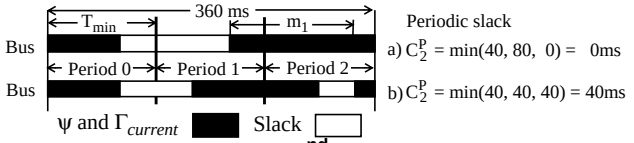

Figure 4. Example for the 2nd Message Design Criterion

and on the bus, we have used the squares of the metrics.

A design alternative that does not meet the second design criterion for messages is not considered a valid solution. Thus, using the last term, we strongly penalize the cost function if b_{need} is not satisfied, by using high values for the w_2 weight.

At this point, we can give an exact formulation to our problem. Given an existing set of applications ψ which are already mapped and scheduled, and an application $\Gamma_{current}$ to be mapped on top of ψ, we are interested to find a mapping and scheduling of $\Gamma_{current}$ which satisfies all deadlines such that the existing applications are disturbed as little as possible. At the same time, the solution should minimize the cost function C, considering a family of future applications characterized by the sets S_U and S_b, the functions f_{SU} and f_{Sb} as well as the parameters T_{min} and b_{need}.

4. Mapping and Scheduling Strategy

As shown in Figure 5, our mapping and scheduling strategy (MH) has two steps. In the first step we try to obtain a valid solution for $\Gamma_{current} \cup \Omega$ so that the total modification cost $R(\Omega)$ is minimized ($\Omega \subseteq \psi$ is the subset of existing applications that have to be modified to accommodate $\Gamma_{current}$). Starting from such a solution, a second step iteratively improves on the design in order to minimize the cost function C. We iteratively improve the design using a transformational approach. A new design is obtained from the current one by performing a transformation called *move*. We consider the following moves: moving a process to a different node, and moving a message to a different slack on the bus. Only those moves are valid moves that result in a schedulable system. The intelligence of the Mapping Heuristic lies in how the potential moves are selected. For each iteration a set of potential moves is generated by the PotentialMove functions. SelectMove functions then evaluate these moves with regard to the respective metrics and selects the best one to be performed.

4.1 The Initial Mapping and Scheduling

The first step of MH consists of an iteration that tries subsets $\Omega \subseteq \psi$ with the intention to find that subset $\Omega=\Omega_{min}$ which produces a valid solution for $\Gamma_{current} \cup \Omega$ such that $R(\Omega)$ is minimized.

Given a subset Ω, the InitialMappingScheduling function (IMS) constructs a mapping and schedule for $\Gamma_{current} \cup \Omega$ that meets the deadlines (both for processes in $\Gamma_{current}$ and those in Ω), without worrying about the design criteria in section 3.3. For IMS we used as a starting point the mapping algorithm introduced in [15], based on a simulated annealing strategy. We have modified the mapping algorithm in [15] to consider during mapping a set of previous applications that have already been mapped, and to schedule the messages according to the TDMA protocol, using the MM approach [12]. The schedulability test that checks a particular mapping alternative is performed according to our schedulability analysis presented in [10].

If IMS succeeds in finding a mapping and a schedule which meet the deadlines, this is not yet a valid solution. In order to produce a valid solution we iteratively try to satisfy the second design criterion for messages. In terms of our metrics, that means a mapping and scheduling such that $C_2^m \geq b_{need}$. Potential moves can be the shifting of messages inside their worst case (largest) [ASAP, ALAP] interval in order to improve the periodic slack. In PotentialMoveC_2^m , we also consider movement of processes, trying to place the sender and receiver of a message on the same processor and, thus, reducing the bus load. SelectMoveC_2^m evaluates these moves with regard to the second design criterion and selects the best one to be performed. Consider Figure 4a. In *Period 2* on node N_1 there is no available slack. However, if we move message m_1 with 40 ms to the left into *Period 1*, as depicted in Figure 4b, we create a slack in *Period 2*, thus the periodic slack on the bus will be min(40, 40, 40)=40, instead of 0.

4.2 Incremental Mapping and Scheduling Strategy

If Step 1 of the MH algorithm (Figure 5) has succeeded, a mapping and scheduling of $\Gamma_{current} \cup \Omega$ has been produced which corresponds to a valid solution. In addition, Ω is such that the total modification cost is as small as possible (minimization of the modification cost is detailed in section 4.3). Starting from this valid solution, the second step of the MH strategy tries to improve on the design in order to minimize the cost function C. In a similar way as during Step 1, we iteratively improve the design by successive moves, without invalidating the second criterion achieved in the first loop.

The loop ends when there is no improvement achieved on the first two terms of the cost function, or a limit imposed on the number of iterations has been reached. For each iteration, those moves are performed which have the highest chance to improve the cost function. The moves are generated in the PotentialMove functions, and are evaluated and selected based on the respective metrics in the SelectMove functions. We now briefly discuss the PotentialMoveC_1^P and PotentialMoveC_1^m functions (PotentialMoveC_2^m has been discussed in the previous section).

PotentialMoveC_1^P Let U_f be the total utilization factor of the largest future application Γ_{fmax}, and U_0 the utilization of that part which cannot be mapped in the current design alternative. This function is responsible for selecting moves

```
MappingSchedulingStrategy (MH)
Ω=∅ -- Step 1: try to find a valid schedule for Γcurrent that minimizes R(Ω)
repeat
   succeeded=IMS(ψ \ Ω, Γcurrent∪Ω) -- initial mapping and scheduling
   ASAP(Gcurrent∪Ω); ALAP(Gcurrent∪Ω)
   -- compute worst case ASAP-ALAP intervals for messages
   if succeeded then
      repeat -- try to satisfy the second message related design criterion
      -- find moves with highest potential to maximize C2m
      move_set=PotentialMoveC2m (Gcurrent∪Ω)
      -- select and perform move which improves most C2m
      move = SelectMoveC2m (move_set); Perform(move)
      succeeded = C2m ≥ bneed
      until succeeded or limit reached
   end if
   if succeeded and R(Ω) smallest so far then
      Ωvalid=Ω; solutionvalid=solutioncurrent
   end if
   Ω=NextSubset(Ω) -- try another subset
until termination condition
if not succeeded then modify architecture; go to step 1; end if
-- Step 2: try to improve the cost function C
solutioncurrent=solutionvalid; Ωmin=Ωvalid
repeat -- find moves with highest potential to minimize C
   move_set=PotentialMoveC1P (Gcurrent∪Ωmin)
      ∪ PotentialMoveC1m (Gcurrent∪Ωmin)
   -- select move which improves C
   -- and does not invalidate the second message related design criterion
   move = SelectMoveC1(move_set); Perform(move)
until C1 has not changed or limit reached
end MappingSchedulingStrategy
```

Figure 5. Mapping and Scheduling Strategy (MH)

of processes from one node to another so that $c_1^P = \frac{U_0}{U_f}100$ is reduced. Moving a process P_i with the utilization factor U_i from a node N_j where it is currently mapped to a node N_k will increase the available utilization on node N_j to $U_{Nj}+U_i$ and decrease the available utilization on N_k to $U_{Nk}-U_i$. To find out U_0 in this new case would mean executing the bin-packing with the processes of the future application as objects and the new available utilization configuration as containers. This can take significant execution time, however, since it has to be done for each potential move.

In section 3.3 we have explained how we can determine the processes that make up the largest future application, Γ_{fmax}, based on the total available utilization and the characterization of future applications. Let us assume that Γ_{fmax} consists of the set $P_{fmax}=\{P_{f1}, P_{f2},...P_{fn}\}$ of processes, and that $P_0=\{P_{fi}, P_{fi+1},...,P_{fm}\}$ are the ones that cannot be mapped in the current design alternative. The total utilization requested by the unmapped processes is $U_0=U_{fi} + U_{fi+1} + ... + U_{fm}$. For the potential move of P_i from N_j to N_k we have to recalculate C_1^P which means determining U_0.

In order to reduce the execution time needed by the bin-packing algorithm, we do not consider all the processes of Γ_{fmax} as objects to be packed. We consider for repacking only those processes belonging to Γ_{fmax} that had to be removed from N_k to make room for P_i, together with those that were already left outside. Our heuristic considers that to make room for P_i on node N_k we remove those processes $P_i^{Nk} \subset \Gamma_{fmax}$ mapped on N_k which have the smallest utilization factor, since they are the ones that should be easiest to fit on other nodes. The metric used by SelectMove to rank this move is the sum of the utilization factors of processes which are left out after trying to repack the $P_0 \cup P_i^{Nk}$ set.

Out of the best moves according the previous metric, we encourage those that have the smallest impact on the schedulability analysis, since we would like to keep the system schedulable. This means moving processes that have low priority (do not have a large impact on other processes) and have a response time that is considerably smaller than their deadline (D_i - R_i is large).

PotentialMoveC_1^m In order to avoid excessive fragmentation of the slack on the bus we will consider moving a message to a position that "snaps" to another existing message. A message is selected for potential move if it has the smallest "snapping distance", i.e. in order to attach it to other message it has to travel the smallest distance inside the schedule table. We also consider moves that try to increase the individual slacks sizes. Therefore, we first eliminate slack that is unusable: it is too small to hold the smallest message of the future application. Then, the slacks are sorted in ascending order and the smallest one is considered for improvement. Such improvement of a slack is performed through moving a nearby message, but avoiding to create as a result an even smaller individual slack.

4.3 Minimizing the Modification Cost

In the first step of our mapping strategy, described in Figure 5, we iterate on subsets Ω searching for a valid solution which also minimizes the total modification cost $R(\Omega)$. As a first attempt, the algorithm searches for a valid implementation of $\Gamma_{current}$ without disturbing the existing applications ($\Omega=\varnothing$). If no valid solution is found successive subsets Ω produced by the function NextSubset are considered, until a terminating condition is met. The performance of the algorithm, in terms of runtime and quality of the solutions produced, is strongly influenced by the implementation of the function NextSubset and the termination condition. They determine how the design space is explored while testing different subsets Ω of applications.

4.3.1 Exhaustive Search (ES)

In order to find Ω_{min}, the simplest solution is to try successively all the possible subsets $\Omega \subseteq \psi$. These subsets are generated in the ascending order of the total modification cost, starting from $\varnothing$. The termination condition is fulfilled when the first valid solution is generated. Since the subsets are generated in ascending order, according to their cost, the subset Ω that first produces a valid solution is also the subset with the minimum modification cost.

The generation of subsets is performed according to the graph G that characterizes the existing applications (see section 3.1). Finding the next subset Ω, starting from the current one, is achieved by a branch and bound algorithm that in the worst case grows exponentially in time with the number of applications. For the example in Figure 2, the call to NextSubset($\varnothing$) will generate $\{\Gamma_7\}$ which has the smallest non-zero modification cost. The next generated subsets, in order, together with their corresponding total modification cost are: $R(\{\Gamma_3\})$=50, $R(\{\Gamma_3, \Gamma_7\})$=70, $R(\{\Gamma_4, \Gamma_7\})$=90 (the inclusion of Γ_4 triggers the inclusion of Γ_7), $R(\{\Gamma_2, \Gamma_3\})$=120, $R(\{\Gamma_3, \Gamma_4, \Gamma_7\})$=140, $R(\{\Gamma_1\})$=150, and so on. The total number of possible subsets according to the graph G is 16.

This approach, while finding the optimal subset Ω, requires a large amount of computation time and can be used only with a small number of applications.

4.3.2 Ad-hoc Subset Selection (AS)

If the number of applications is large, a possible ad-hoc solution could be based on a greedy strategy which, starting from $\Omega=\varnothing$, progressively enlarges the subset until a valid solution is produced. The algorithm looks at all the non-frozen applications and picks that one which, together with its dependencies, has the smallest modification cost. If the new subset does not produce a valid solution, it is enlarged by including, in the same fashion, the next application with its dependencies. This greedy expansion of the subset is continued until the set is large enough to lead to a valid solution or no application is left. For the example in Figure 2 the call to NextSubset($\varnothing$) will produce $R(\{\Gamma_7\})$=20, and will be successively enlarged to $R(\{\Gamma_7, \Gamma_3\})$=70, $R(\{\Gamma_7, \Gamma_3, \Gamma_2\})$=140 ($\Gamma_4$ could have been picked as well in this step because it has the same modification cost of 70 as Γ_2 and its dependence Γ_7 is already in the subset), $R(\{\Gamma_7, \Gamma_3, \Gamma_2, \Gamma_4\})$=210, and so on.

While this approach finds very quickly a valid solution, if one exists, it is possible that the total modification cost is much higher than the optimal one.

4.3.3 Subset Selection Heuristic (SH)

An intelligent heuristic should be able to identify the reasons due to which a valid solution has not been found and use this information when selecting applications to be included in Ω. There can be two possible causes for not finding a valid solution: an initial mapping which meets the deadlines has not

been produced, or the second criterion is not satisfied.

Let us investigate the first reason. If an application Γ_i is schedulable, this means that all its processes meet their deadlines. If IMS determines that the application is not schedulable this means that at least one of the processes P_i missed its deadline: $R_i > D_i$. Besides the intrinsic properties of the application that can lead to this situation, process P_i can miss its deadline also because of the interference of higher priority processes that are mapped on the same node with P_i, processes that can also belong to other applications. In this situation we say that there is a *conflict* with processes belonging to other applications. We are interested to find out which applications are responsible for conflicts encountered by our $\Gamma_{current}$, and not only that, but also which ones are *flexible* enough to move away in order to avoid these conflicts (D_i - R_i is large).

IMS determines a metric Δ_i that characterizes the degree of conflict and the flexibility of application Γ_i in relation to $\Gamma_{current}$. A set of applications Ω will be characterized, in relation to $\Gamma_{current}$, by $\Delta(\Omega)=\sum_{\Gamma_i \in \Omega} \Delta_i$. The metric $\Delta(\Omega)$ will be used by our subset selection heuristic if IMS has failed to produce a solution which satisfies the deadlines. An application with a larger Δ_i is more likely to lead to a valid schedule if included in Ω.

Basically, Δ_i is the total amount of *interference* caused by higher priority processes of Γ_i to processes in $\Gamma_{current}$. For a process P_i, the interference I_{ji} from a higher priority process P_j mapped on the same node, is the time that P_j delays the execution of P_i, and is given by $I_{ji}=\left\lceil \frac{J_j + R_i}{T_j} \right\rceil C_j$ where J_j is the release jitter of process P_j and a detailed description of how it is calculated in the context of the MM approach for message scheduling over TTP is given in [10]. Figure 6 presents in more detail how Δ_i is calculated.

If the initial mapping was successful, the first step of MH could fail during the attempt to satisfy the second design criterion for messages. In this case, the metric Δ_i is computed in a different way. It will capture the potential of an application Γ_i to improve the metric C_2^m if remapped together with $\Gamma_{current}$. Thus, for the improvement of C_2^m we consider a total number of moves from all the non-frozen applications (determined using PotentialMoveC_2^m (y), see section 4.1). For each move that has as subject $m_j \in \Gamma_i$, we increment the metric Δ_i with the predicted improvement on C_2^m .

MH starts by trying an implementation of $\Gamma_{current}$ with $\Omega=\varnothing$. If this attempt fails, because of one of the two reasons mentioned above, the corresponding metrics Δ_i are computed for all $\Gamma_i \in \psi$. Our heuristic SH will then start by finding the ad-hoc solution Ω_{AH} produced by the AS algorithm (this will succeed if there exists any solution) with a corresponding cost $R_{AH}=R(\Omega_{AH})$ and a $\Delta_{AH}=\Delta(\Omega_{AH})$. SH now continues by trying to find a solution with a more favourable Ω (a smaller total cost R). Therefore, the thresholds $R_{max}=R_{AH}$ and $\Delta_{min}=\Delta_{AH}/n$ (for our experiments we considered n=2) are set. For generating new subsets Ω, the function NextSubset now follows a similar approach like ES but in a reverse direction, towards smaller subsets, and it will consider only subsets with a smaller total cost than R_{max} and a larger Δ than Δ_{min} (a small Δ means a reduced potential to eliminate the cause of the initial failure). Each time a valid solution is found, the current values of R_{max} and Δ_{min} are updated in order to further restrict the search space. The heuristic stops when no subset can be found with $\Delta>\Delta_{min}$, or a certain imposed limit has been reached (e.g. on the total number of attempts to find new sets).

```
DeltaMetrics(Γcurrent, Ω)
for each non frozen Γi ∈ Ω Δi = 0 end for
for each Pi ∈ Γcurrent
    if Ri > Di
        for each non frozen Γk ∈ Ω
            -- hp(Pi) is the set of processes with higher priority than Pi
            for each Pj ∈ Γk ∩ hp(Pi): Δk = Δk + Cj⌈(Jj+Ri)/Tj⌉ end for
        end for
    end if
end for
return Δ
end DeltaMetrics
```

Figure 6. Determinig the Δ metrics

5. Experimental Results

For the evaluation of our mapping strategies we first used process sets of 40, 80, 160, 240 and 320 processes representing the $\Gamma_{current}$ application generated for experimental purpose. 30 applications were generated for each set dimension, thus a total of 150 applications were used for experimental evaluation. We considered an architecture consisting of 10 nodes of different speeds. For the communication channel we considered a transmission speed of 256 kbps and a length below 20 meters. The maximum length of the data field in a bus slot was 8 bytes. All experiments were run on a SUN Ultra 10.

The first result concerns the quality of the designs obtained with our mapping strategy MH using the search heuristic SH compared to the case when the ad-hoc approach AS and the exhaustive search ES are used for subset selection. For each of the five application dimensions generated we have considered a set of existing applications ψ consisting of 160, 240, 320, 400 and 480 processes, respectively. The sets contained 4, 6, 8, 10 and 12 applications, each application with an associated modification cost assigned manually in the range 10 to 100. The dependencies between applications were such that the total number of subsets resulted for each set ψ were 8, 32, 128, 256, and 1024. We have considered that the future applications Γ_{future} consist of a process set of 80 processes, randomly generated according to the following specifications: S_U={0.02, 0.05, 0.1, 0.15, 0.2}, $f_{S_U}(S_U)$={10, 25, 45, 15, 5%}, S_b={2, 4, 6, 8 bytes}, $f_{S_b}(S_b)$={20, 50, 20, 10%}, T_{min}=250 ms and b_{need}=20 ms.

MH has been used to produce a valid solution for each of the 150 process sets representing $\Gamma_{current}$ on top of the existing applications ψ using the ES, AS and SH approaches to subset selection. For each of the resulted valid solutions, there corresponds a minimum modification cost $R(\Omega_{min})$. Figure 7a compares the three approaches to subset selection based on the modification cost needed in order to obtain a valid solution. The exhaustive approach ES is able to obtain valid solutions at the optimum (smallest) modification cost, (e.g. less than 400, in average, for systems with 12 applications consisting of a total of 480 processes), while the ad-hoc approach AS needs in average 3.11 times more costly modifications in order to obtain valid solutions (e.g. more than 1100 for 480 processes in Figure 7a). However, in order to find the optimal remapping the ES approach needs large computation times. For example, it can take more than 35 minutes, in average, in order to find the smallest cost subset to be remapped that

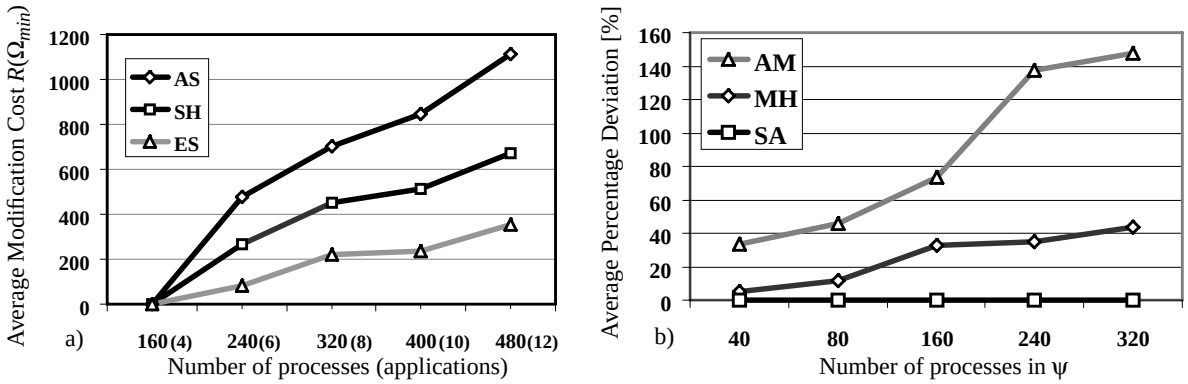

Figure 7. a) Average Modification Costs for AS, SH, ES and b) Percentage Deviations for AH, MH, SA

leads to a valid solution in the case we have 12 applications. From Figure 7a we can see that the proposed heuristic SH performs quite well, needing only 1.84 times larger costs, in average, in order to obtain a valid schedule, and this is achieved at a computation cost comparable with the fast ad-hoc approach AS. For the results in Figure 7a we have eliminated those situations in which a valid solution has not been produced by MH (which means that there is no solution regardless of the modification cost).

Next, we were interested to investigate the quality of the mapping heuristic MH compared to a so called *ad-hoc mapping approach* (AM). To concentrate on this, we have considered that *no modifications* are allowed to the applications in ψ. The AM approach is a simple, straight-forward solution to produce designs which, to a certain degree, support an incremental process. AM tries to evenly balance the available utilization remaining after mapping the current application. The quality of the designs obtained with MH and AM were compared with a near-optimal mapping and schedule obtained with a Simulated Annealing strategy (SA) strategy [12], that minimizes the cost function C (section 3.4). One of the drawbacks of the SA strategy is that in order to find near-optimal solutions it needs very large computation times. Such a strategy, although useful for the final stages of the system synthesis, cannot be used inside a design space exploration cycle.

MH, SA and AM have been used to map each of the 150 process sets representing $\Gamma_{current}$ on the existing applications ψ. For each of the resulted designs, the objective function *C* has been computed. Very long and expensive runs have been performed with the SA algorithm for each process set and the best ever solution produced has been considered as the near-optimum for that process set. We have compared the cost function obtained for the 150 process sets considering each of the three mapping algorithms. Figure 7b presents the average percentage deviation of the cost function obtained with the MH and AH from the value of the cost function obtained with the near-optimal scheme. We have excluded from the results in Figure 7b, 28 solutions obtained with AH for which the second design criterion for messages has not been met, and thus the objective function has been strongly penalized. The SA approach performs best in terms of quality at the expense of a large execution time, which can be up to 40 minutes for large sets of 320 processes. MH performs very well, and is able to obtain good quality solutions in a very short time, e.g., 6.5 seconds for 320 processes. AH is very fast, but since it does not address explicitly the design criteria presented in Section 3 it has the worst quality of solutions, according to the cost function.

The most important aspect of the experiments is determining to which extent the mapping strategies proposed in the paper really facilitate the implementation of future applications. To find this out, we have mapped process sets of 40, 80, 160 and 240 processes representing the $\Gamma_{current}$ application on top of the previously generated existing applications ψ. After mapping and scheduling each of these applications we have tried to add a new application Γ_{future} to the resulted system. Γ_{future} consists of a process set of 80 processes, randomly generated according to the same specifications presented before. The experiments have been performed two times, using first MH*[1] and then AM for mapping $\Gamma_{current}$. In both cases we were interested if it is possible to find a valid implementation for Γ_{future} on top of $\Gamma_{current}$ using the initial mapping algorithm IMS. Figure 8a shows the number of successful implementations in the two cases. In the case $\Gamma_{current}$ has been mapped with MH*, this means using the design criteria and metrics proposed in the paper, we were able to find a valid schedule for 56% of the total mapping attempts with IMS using Γ_{future}. However, using AH to map $\Gamma_{current}$ has led to a situation where IMS is able to find valid schedules in only 31% of the cases. Another observation from Figure 8 is that when the available utilization is large, as in the case $\Gamma_{current}$ has only 40 processes, it is easy for both MH* and AM to find a mapping that allows adding future applications. However, as $\Gamma_{current}$ grows to 80, only MH* is able to find a mapping of $\Gamma_{current}$ that supports an incremental design process, accommodating more than 60% of the future applications, while using AM only less than 25% are accommodated. If the remaining utilization is very small, after we map a $\Gamma_{current}$ of 240, it becomes practically impossible to map new applications without modifying the current system.

However, in the case the mapping heuristic is *allowed to modify* the existing system as discussed in this paper then we are able to increase the number of successfully mapped Γ_{future} applications to 73% from the total instead of only 56%. The percentage of accommodated Γ_{future} applications, for different dimensions of $\Gamma_{current}$, if modifications are allowed on the existing system, is shown by the diagram MH in Figure 8b. After mapping a $\Gamma_{current}$ with 80 processes using MH we

1. MH* is the same mapping heuristic as in Figure 5, but in which we do not allow modifications to the existing applications.

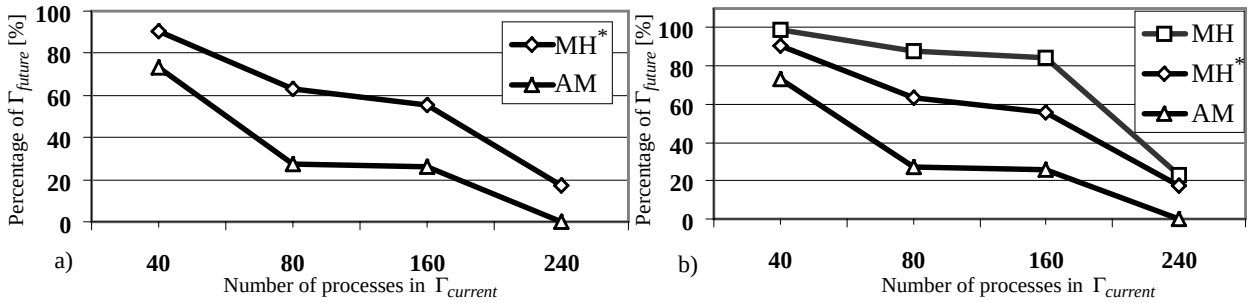

Figure 8. Percentage of Γ_{future} Apps. Successfully Mapped: a) No modifications b) Modifications Allowed

are able to accommodate 88% of the future applications, compared to only 61% in the case we do not allow modifications to the existing system (MH*). Such an increase is, of course, expected. The important aspect, however, is that it is obtained not by randomly selecting old applications to be remapped, but by performing this selection such that the total modification cost is minimized.

Finally, we considered an example implementing a vehicle cruise controller (CC) modelled as a process set. The CC has 32 processes and it was to be mapped on an architecture consisting of 4 nodes, namely: Anti Blocking System, Transmission Control Module, Engine Control Module and Electronic Throttle Module. The system ψ consists of 80 processes generated randomly. The CC is the $\Gamma_{current}$ application to be mapped. We have also generated 30 future applications of 40 processes each with the characteristics of the CC, which are typical for automotive applications. By mapping the CC using MH* we were able to later map 18 of the future applications, while using AH only 6 of the future applications could be mapped. MH* and AH do not consider modifications to the existing system. When modifications are allowed, using the MH approach, we are able to map 26 of the 30 future applications.

6. Conclusions

We have presented an approach to the incremental design of distributed hard real-time systems. Such a design process satisfies two main requirements when adding new functionality: the already running functionality is disturbed as little as possible, and there is a good chance that, later, new functionality can easily be mapped on the resulted system. Our approach was considered in the context of a fixed priority scheduling policy for processes and a static cyclic scheduling policy for messages. Scheduling of messages has been done using a realistic communication model based on a TDMA scheme.

We have introduced several design criteria with their corresponding metrics, that drive our mapping strategies to solutions supporting an incremental design process. For constructing an initial valid solution, we have shown that it is needed to take into account the features of the communication protocol.

Three algorithms have been proposed to produce a minimal subset of applications which have to be remapped and scheduled in order to implement the new functionality. ES is based on a, potentially slow, branch and bound strategy which finds an optimal solution. AS is very fast but produces solutions that could be of too high cost, while SH is able to quickly produce good quality results. The approaches have been validated through several experiments.

References

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

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

[3] Dobrin R., Özdemir, Y., Fohler, G. Task Attribute Assignment of Fixed Priority Scheduled Tasks to Reenact Off-Line Schedules, In Proc. of RTCSA 2000 Korea , December 2000.

[4] Ermedahl, H., Hansson, H., Sjödin, M. 1997. Response-Time Guarantees in ATM Networks. Proceedings of the 18th IEEE Real-Time Systems Symposium, 274-284.

[5] Gutiérrez García, J.J., González Harbour, M. 1995. Optimized Priority Assignment for Tasks and Messages in Distributed Hard Real-Time Systems, Proc. 3d Workshop on Parallel and Distributed Real-Time Systems, 124-132.

[6] Kopetz, H., Grünsteidl, G. 1994. TTP-A Protocol for Fault-Tolerant Real-Time Systems. IEEE Computer, 27(1), 14-23.

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

[8] Lonn, H., Axelsson, J. 1999. A Comparison of Fixed-Priority and Static Cyclic Scheduling for Distributed Automotive Control Applications. Proceedings of the 11th Euromicro Conference on Real-Time Systems, 142-149.

[9] Martello, S., Toth, P. 1990. Kanpsack Problems: Algorithms and Computer Implementations. Wiley.

[10] Pop P., Eles P., Peng Z. 1999. Schedulability-Driven Communication Synthesis for Time Triggered Embedded Systems. Proc. of the 6th International Conference on Real-Time Computing Systems and Applications, 287-294.

[11] Pop P., Eles P., Pop T., Peng Z. 2001. Minimizing System Modification in an Incremental Design Approach. Proceedings of the 9th Int. Symp. on Hardware/Soft. Codesign, 183-188.

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

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

[14] Sha, L., Rajkumar, R., Lehoczky, J. 1990. Priority Inheritance Protocols: An Approach to Real-Time Synchronization. IEEE Transactions on Computers, 39(9), 1175-1185.

[15] Tindell, K., Burns, A., Wellings, A.J. 1992. Allocating Real-Time Tasks (An NP-Hard Problem made Easy). Real-Time Systems, 4(2), 145-165.

[16] Tindell, K., Burns, A., Wellings, A.J. 1995. Calculating Controller Area Network (CAN) Message Response Times. Control Eng. Practice, 3(8), 1163-1169.

[17] Tindell, K., Clark, J. 1994. Holistic Schedulability Analysis for Distributed Hard Real-Time Systems, Microprocessing and Microprogramming, 40, 117-134.