Analysis and Optimization of Distributed Real-Time Embedded Systems

PAUL POP, PETRU ELES, ZEBO PENG, and TRAIAN POP
Linköping University

An increasing number of real-time applications are today implemented using distributed heterogeneous architectures composed of interconnected networks of processors. The systems are heterogeneous not only in terms of hardware and software components, but also in terms of communication protocols and scheduling policies. In this context, the task of designing such systems is becoming increasingly difficult. The success of new adequate design methods depends on the availability of efficient analysis as well as optimization techniques. In this article, we present both analysis and optimization approaches for such heterogeneous distributed real-time embedded systems. More specifically, we discuss the schedulability analysis of hard real-time systems, highlighting particular aspects related to the heterogeneous and distributed nature of the applications. We also introduce several design optimization problems characteristic of this class of systems: mapping of functionality, the optimization of access to communication channel, and the assignment of scheduling policies to processes. Optimization heuristics aiming at producing a schedulable system with a given amount of resources are presented.

Categories and Subject Descriptors: C.0 [**Computer Systems Organization**]: General—*System architectures*; D.4.1 [**Operating Systems**]: Process Management—*Scheduling*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*; D.4.7 [**Operating Systems**]: Organization and Design—*Real-time systems and embedded systems*; J.6 [**Computer Applications**]: Computer-Aided Engineering—*Computer-aided design (CAD)*

General Terms: Algorithms, Design, Performance, Theory

1. INTRODUCTION

Embedded real-time systems have to correctly implement the required functionality. In addition, they have to fulfill a wide range of competing constraints: development cost, unit cost, reliability, security, size, performance, power consumption, flexibility, time-to-market, maintainability, safety, etc. critical to the correct functioning of such systems are their timing constraints: "the correctness of the system behavior depends not only on the logical results of the computations, but also on the physical instant at which these results are produced" [Kopetz 1997].

Authors' address: P. Pop, P. Eles, Z. Peng, and T. Pop, Department of Computer and Information Science, Linköping University, 581 83 Linköping, Sweden; email: paupo@ida.liu.se.

Real-time systems have been classified as *hard* and *soft* [Kopetz 1997]. Basically, in a hard real-time system, a result that fails to meet a timing constraint is considered incorrect, and can potentially have catastrophic consequences. For example, a brake-by-wire system in a car failing to react within a given time interval can result in a fatal accident. On the other hand, a multimedia system, which is a soft-real time system, can under certain circumstances tolerate a given amount of delays, perhaps resulting in a patchier picture, but without serious consequences besides some possible inconvenience to the user.

Many real-time applications following physical, modularity, or safety constraints are implemented using *distributed architectures*. Such systems are composed of several different types of hardware components interconnected in a network. For these systems, the communication between functions implemented on different nodes has an important impact on overall system properties such as performance, cost, maintainability, etc.

1.1 Automotive Electronics

Although the discussion in this article is valid for several application areas, it is useful to understand the evolution of distributed embedded real-time systems and their design challenges in order to exemplify the developments in a particular area.

Automotive manufacturers, for example, were reluctant until recently to use computer-controlled functions in vehicles. Today this attitude has changed for several reasons. First, there is a constant market demand for increased vehicle performance, more functionality, less fuel consumption, and fewer exhausts, all at lower costs. In addition, from the manufacturers' point of view, there is a need for shorter time-to-market and reduced development and manufacturing costs. These demands, combined with advancements in semiconductor technology which are delivering ever increasing performance at lower and lower costs, have led to a rapid increase in the number of electronically-controlled functions in vehicles [Kopetz 1999].

It is estimated that in 2006, the electronics inside a car will amount to 25% of the total cost of the vehicle (35% for high-end models), a quarter of which will be due to semiconductors [Hansen 2002; Jost 2001]. High-end vehicles currently have up to 100 microprocessors implementing and controlling various parts of their functionality.

At the same time, with the increased complexity of embedded automotive electronics systems, the type of functions they implement has also evolved. Thanks to the semiconductor revolution in the late 1950s, electronic devices became small enough to install on board vehicles. In the 1960s the first analog fuel injection system appeared, and in the 70s, analog devices for controlling the transmission, carburetor, and spark advance timing were developed. The oil crisis of the 70s led to the demand for engine control devices that improved the efficiency of the engine, thus reducing fuel consumption. In this context, the first microprocessor-based injection control system appeared in 1976 in the USA. During the 80s, more sophisticated systems began to appear, like electronically-controlled braking systems, dashboards, information and navigation systems, air conditioning systems, etc. In the 90s, developments and improvementes concentrated on aspects of safety and convenience. Today, it is not uncommon to have highly critical functions like steering or braking implemented through electronic functionality only, without any mechanical backup, as is the case in drive-by-wire and brake-by-wire systems [Chiodo 1996; X-by-Wire Consortium 1998; Navet et al. 2005].

1.2 Timing Analysis and Design Optimization

An increasing number of real-time applications are implemented today using distributed *heterogeneous* architectures composed of interconnected networks of processors. The systems are heterogeneous not only in terms of hardware components, but also in terms of communication protocols and scheduling policies. Each network has its own communication protocol, each processor in the architecture can have its own scheduling policy, and several scheduling policies can share a processor [Lönn and Axelsson 1999; Richter et al. 2003; Pop et al. 2002, 2003].

The task of designing such systems is becoming both increasingly important and difficult at the same time. The success of adequate design methods depends on the availability of both analysis and optimization techniques. This article presents a holistic analysis for heterogeneous distributed hard real-time embedded systems which:

—can handle distributed applications, data, and control dependencies;
—accurately take into account the details of communication protocols; and
—handle heterogeneous scheduling policies.

Once this holistic analysis has been shown, we address some design problems characteristic of the systems under consideration:

—mapping of functionality to the components of the architecture;
—optimization of access to the communication channel; and
—assignment of a scheduling policy to the processes in the application.

The article is organized in nine sections. Sections 2 and 3 present the heterogeneous real-time embedded systems we address, and the type of application models we consider, respectively. Section 4 presents the system- level design of embedded systems and highlights the particular design tasks we address. Section 5 introduces our proposed holistic scheduling approach. The rest of the article focuses in more detail on certain design optimization issues. Section 6 identifies mapping, bus access optimization, and scheduling policy assignment as interesting design optimization problems characteristic of heterogeneous distributed embedded real-time systems. We show in Section 7 how the analysis presented in Section 5 can be used to drive the optimization strategies that deal with the aforementioned problems. The last two sections present some experimental results and conclusions.

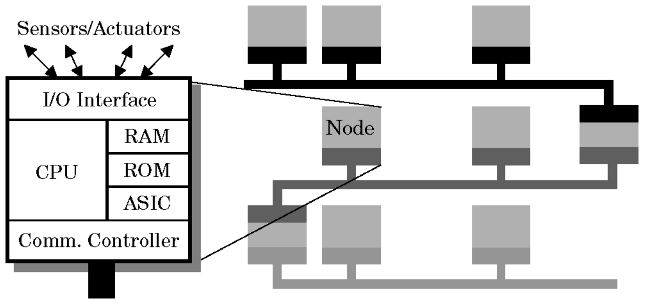


Fig. 1. Distributed real-time systems.

2. DISTRIBUTED REAL-TIME EMBEDDED SYSTEMS

2.1 Hardware Architecture

Currently, distributed real-time systems are implemented using architectures where each node is dedicated to the implementation of a single function or class of functions. The complete system can be, in general, composed of several networks interconnected with each other (see Figure 1). Each network has its own communication protocol, and internetwork communication is via a *gateway*, which is a node connected to both networks. The architecture can contain several such networks having different types of topologies.

A network is composed of several different types of hardware components, called *nodes*. Typically, every node, also called an *electronic control unit* (ECU), has a communication controller, CPU, RAM, ROM, and an I/O interface to sensors and actuators. Nodes can also have ASICs in order to accelerate parts of their functionality.

The microcontrollers used in a node and the type of network protocol employed are influenced by the nature of the functionality and the imposed real-time, fault-tolerance, and power constraints. In the automotive electronics field, the functionality is typically divided into two classes, depending on the level of criticalness:

—*Body electronics* refers to the functionality that controls simple devices such as the lights, mirrors, windows, and dashboard. The constraints of body electronic functions are determined by the reaction time of the human operator in the range of 100 to 200 ms. A typical body electronics system within a vehicle consists of a network of 10 to 20 nodes that are interconnected by a low bandwidth communication network such as LIN [Lin Consortium 2005]. A node is usually implemented using a single-chip 8 bit microcontroller (e.g., Motorola 68HC05 or Motorola 68HC11) with some hundred bytes of RAM and kilobytes of ROM, I/O points to connect sensors and control actuators, and a simple network interface. Moreover, the memory size is growing by more than 20% each year [Kopetz 1999].

—*System electronics* are concerned with the control of vehicle functions that are related to the movement of the vehicle. Examples of system electronics applications are engine control, braking, suspension, and vehicle dynamics control. The timing constraints of system electronic functions are in the range of a couple of ms to 20 ms, requiring 16 bit or 32 bit microcontrollers (e.g., Motorola 68332) with about 16 kilobytes of RAM and 256 kilobytes of ROM. These microcontrollers have built-in communication controllers (e.g., the 68HC11 and 68HC12 automotive families of microcontrollers have on-chip CAN controllers), I/Os to sensors and actuators, and are interconnected by high bandwidth networks [Kopetz 1999].

2.2 Communication Protocols

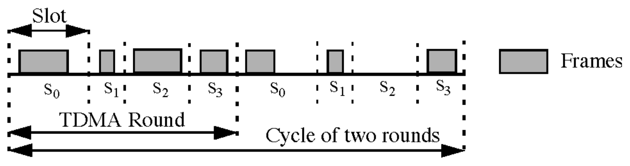
As communications become critical components, new protocols are needed that can cope with the required high bandwidth and predictability.

There are several communication protocols for real-time networks. Among those that have been proposed for vehicle multiplexing, only the Controller Area Network (CAN) [Bosch 1991], the Local Interconnection Network (LIN) [Lin Consortium 2005], and SAE's J1850 [SAE 1994] are currently in use on a large scale basis. Moreover, only a few of them are suitable for safety-critical applications where predictability is mandatory [Rushby 2001]. A survey and comparison of communication protocols for safety-critical embedded systems is available in Rushby [2001]. Communication activities can be triggered either dynamically in response to an event, or statically at predetermined moments in time.

— On the one hand, there are protocols that schedule messages statically based on the progression of time, for example, the SAFEbus [Hoyme and Driscoll 1992] and SPIDER [Miner 2000] protocols for the avionics industry, and the TTCAN [International Organization for Standardization 2002] and Time-Triggered Protocol (TTP) [Kopetz and Bauer 2003] intended for the automotive industry.
— On the other hand, there are several communication protocols where message scheduling is performed dynamically, such as Controller Area Network (CAN), used in a large number of application areas including automotive electronics, LonWorks [Echelon 2005], and Profibus [Profibus 2005] for real-time systems in general, etc. Out of these, CAN is the most well known and widespread event-driven communication protocol in the area of distributed embedded real-time systems.

2.2.1 *Time-Triggered Protocol.* The TTP integrates all the services necessary for fault-tolerant real-time systems. The bus access scheme is time-division multiple-access (TDMA), meaning that each node N_i connected to the bus can transmit only during a predetermined time interval, the TDMA slot S_i. In such a slot, a node can send several messages packed in a frame. A sequence of slots corresponding to all the nodes in the architecture is called a TDMA round. A node can have only one slot in a TDMA round. Several TDMA rounds can be combined together in a cycle that is repeated periodically. The sequence and length of the slots are the same for all the TDMA rounds. However, the length and contents of the frames may differ.

The TDMA access scheme is imposed by a message descriptor list (MEDL) that is located in every TTP controller. The MEDL serves as a schedule table



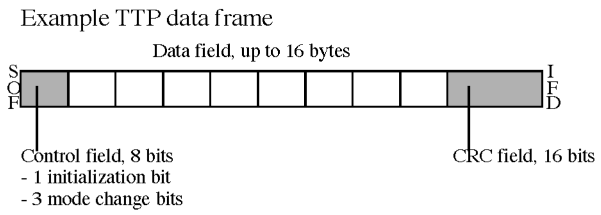


Fig. 2. Time-Triggered Protocol.

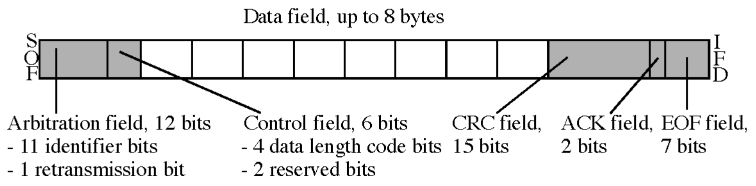


Fig. 3. Controller Area Network data frame (CAN 2.0A).

for the TTP controller, which has to know when to send/receive a frame to/from the communication channel.

There are two types of frames in the TTP: the initialization frames, or I-frames, which are needed for the initialization of a node, and the normal frames, or N-frames, which are the data frames containing, in their data field, the application messages. A TTP data frame (Figure 2) consists of the following fields: a start-of-frame bit (SOF), control field, a data field of up to 16 bytes containing one or more messages, and a cyclic redundancy check (CRC) field. Frames are delimited by the interframe delimiter (IDF, 3 bits).

2.2.2 *Controller Area Network.* The CAN bus is a priority bus that employs a collision avoidance mechanism whereby the node that transmits the frame with the highest priority wins the contention. Frame priorities are unique and are encoded in the frame identifiers, which are the first bits to be transmitted on the bus.

In the case of CAN 2.0A [Bosch 1991], there are four frame types: data frame, remote frame, error frame, and overload frame. A data frame is depicted in Figure 3. It contains seven fields: a SOF, an arbitration field that encodes the 11 bits frame identifier, a control field, a data field of up to 8 bytes, a CRC field, an acknowledgement (ACK) field, and an end-of-frame field (EOF).

2.3 Scheduling Policies

There are two main approaches to scheduling real-time processes running on a network node:

(1) *Time-Triggered* (*TT*)
In the time-triggered approach, activities are initiated at predetermined points in time. In a distributed time-triggered system it is assumed that the clocks of all nodes are synchronized to provide a global notion of time. Time-triggered systems are typically implemented using *nonpreemptive static cyclic scheduling* (SCS), where the process activation or message communication is done based on a schedule table that is built offline. This schedule table contains activation times for each process such that the timing constraints of processes are satisfied.

(2) *Event-Triggered* (*ET*)
In the event-triggered approach, activities happen when a significant change of state occurs. Event-triggered systems are typically implemented using *preemptive priority-based* scheduling, where as a response to an event, the appropriate process is invoked to service it. Two of the most widely used priority-based policies are *fixed priority* scheduling (FPS) and *earliest deadline first* (EDF). In FPS, each process has a fixed (static) priority which is computed offline. The decision of which ready process to activate is taken online according to priority. In the case of EDF, that process will be activated which has the nearest deadline.

There has been a long debate in the real-time and embedded systems communities concerning the advantages and disadvantages of different scheduling approaches [Audsley et al. 1993; Buttazzo 2005; Kopetz 1997, Xu and Parnas 1993, 2000]. SCS has the advantage of *predictability* and testability [Kopetz 1997]. However, such static approaches lack the *flexibility* offered by event-driven ones such as FPS and EDF. EDF is optimal on single processor systems, and in general, leads to high and thus efficient resource utilization [Buttazzo 2005]. In addition, advances in the area of priority-based preemptive scheduling show that predictable applications with hard real-time guarantees can also be handled with strategies such as FPS and EDF [Audsley et al. 1993; Sha et al. 2004].

2.4 Heterogeneous Distributed Real-Time Embedded Systems

An interesting comparison of scheduling approaches from a more industrial— in particular, automotive—perspective can be found in Lönn and Axelsson [1999]. Their conclusion is that we have to choose the right scheduling approach depending on the particularities of the scheduled processes. This means not only that there is no single "best" approach to be used, but also that inside a certain application several scheduling approaches can be used *together*.

Efficient implementation of new, highly sophisticated automotive applications entails the use of time-triggered process sets together with event-triggered ones implemented on top of complex distributed architectures.

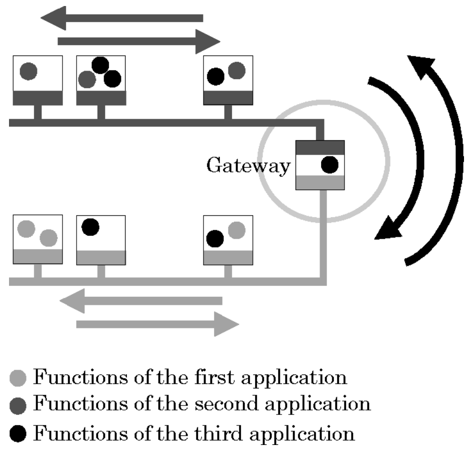

Gateway
● Functions of the first application
● Functions of the second application
● Functions of the third application

Fig. 4. Distributed safety-critical applications.

2.4.1 *Heterogeneous Multiclusters.* Considering the automotive industry, the way functionality has been distributed on an architecture has evolved over time. Initially, distributed real-time systems were implemented using architectures where each node was dedicated to the implementation of a single function or class of functions. This allowed system integrators to purchase nodes implementing required functions from different vendors, and to meld them into their system [EAST-EEA 2002]. There are, however, several problems with this very rigid, one-to-one technique for mapping functionality to nodes:

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

This has created a huge pressure to reduce the number of nodes by integrating several functions into one node and, at the same time, to distribute certain functionality over several nodes (see Figure 4). Moreover, although an application is typically distributed over a single cluster, we are beginning to see applications that are distributed across several clusters. For example, in Figure 4, the third application (represented as black dots) is distributed over two clusters.

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

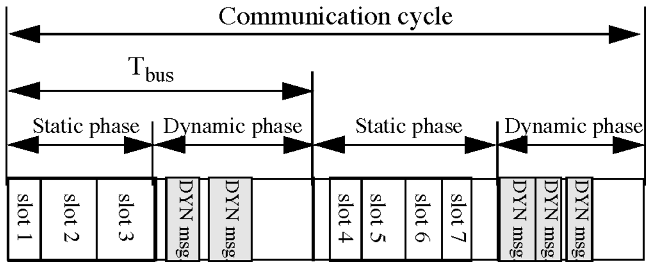


Fig. 5. Mixed communication cycle.

critical information through the network and gateway nodes [Navet et al. 2005].

We call a *time-triggered cluster* (TTC) a cluster where processes are scheduled using SCS and messages are transmitted using a static communication protocol such as TTP. Conversely, and *event-triggered cluster* (ETC) is a cluster where processes are scheduled with FPS or EDF, and the bus implements dynamic communication protocol such as CAN. In this article we consider heterogeneous multicluster systems where an application can be mapped over both types of clusters.

2.4.2 *Heterogeneous Communication Protocols.* In the case of heterogeneous multiclusters, as presented in the previous section, the TT and ET domains interact via the gateway node. However, TT and ET domains can share the same resources. This is particularly true at the level of the communication protocol.

Static, time-triggered protocols have the advantages of simplicity and predictability, while event-triggered protocols are more flexible. Moreover, protocols like TTP offer the fault-tolerant services necessary for implementing safety-critical applications. However, it has been shown by Tindell et al. [1995] that predictability can also be achieved with dynamic protocols such as CAN.

Researchers have proposed hybrid communication protocols such as the Universal Communication Model (UCM) [Demmeler and Giusto 2001], where the TT and ET domains share the same bus. Hybrid types of communication protocols such as Byteflight [Berwanger et al. 2000] introduced by BMW for automotive applications, and the FlexRay protocol [FlexRay Group 2005] allow the sharing of the bus by event-driven and time-driven messages. A hybrid communication protocol like FlexRay offers some of the advantages of both worlds.

In our article we consider that every node in the architecture has a communication controller that implements the protocol services. The controller runs independently of the node's CPU. We model the bus access scheme using the universal communication model. The bus access is organized as consecutive cycles, each with the duration T_{bus}. We consider that the communication cycle is partitioned into static (ST) and dynamic (DYN) phases (Figure 5).

—ST phases consist of time slots, and during a slot only the node associated to that particular slot is allowed to transmit SCS messages. The transmission times of SCS messages are stored in a schedule table, as in the TTP protocol.

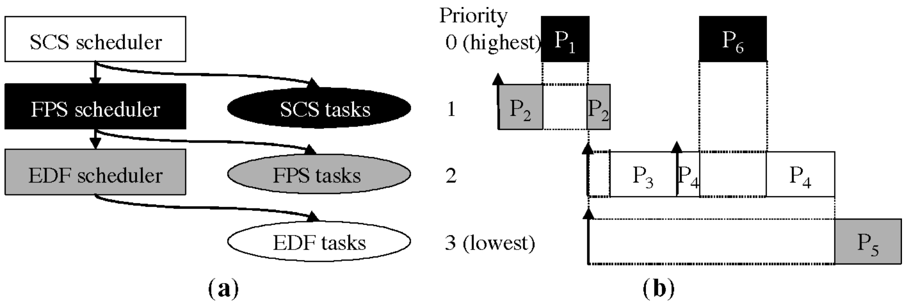


Fig. 6. Heterogeneous scheduling policies.

—During a DYN phase, all nodes are allowed to send messages and the conflicts between nodes trying to send simultaneously are solved by an arbitration mechanism that allows the transmission of the message with the highest priority, as in the CAN protocol. Hence, the ET messages are organized in a prioritized ready queue.

In this article we consider the UCM for the bus communication with heterogeneous protocols. However, the approaches presented can be generalized to other hybrid communication protocols, such as FlexRay, which will very likely become the *de facto* standard for in-vehicle communications.

2.4.3 *Heterogeneous Scheduling Policies.* For the systems we are studying, we have designed a software architecture which runs on the CPU of each node, and which supports resource sharing by the TT and ET domains. The main component of the software architecture is a real-time kernel containing three schedulers (for SCS, FPS, and EDF) organized *hierarchically* (Figure 6a).

(1) The top-level scheduler is a SCS scheduler which is responsible for the activation of SCS processes and the transmission of SCS messages based on a schedule table, and for the activation of the FPS scheduler. Thus, SCS processes and messages are time-triggered (TT), that is, activated at predetermined points in time, and nonpreemptable.
(2) The FPS scheduler activates FPS processes and transmits FPS messages based on their priorities, and activates the EDF scheduler. Processes scheduled using FPS are event-triggered (ET), that is, initiated whenever a particular event is noted, and are preemptable. Messages produced by FPS processes are ET and nonpreemptable.
(3) The EDF scheduler activates EDF processes and sends EDF messages based on their deadlines. EDF processes are ET and preemptable. Messages produced by EDF processes are ET and nonpreemptable.[1]

When several processes are ready on a node, the process with the highest priority is activated and preempts the other processes. Let us consider the example in Figure 6b, where we have six processes sharing the same node. Processes P_1 and

[1]The integration of EDF messages within a priority-based arbitration mechanism, such as CAN, has been detailed in [Livani and Kaiser 1998].

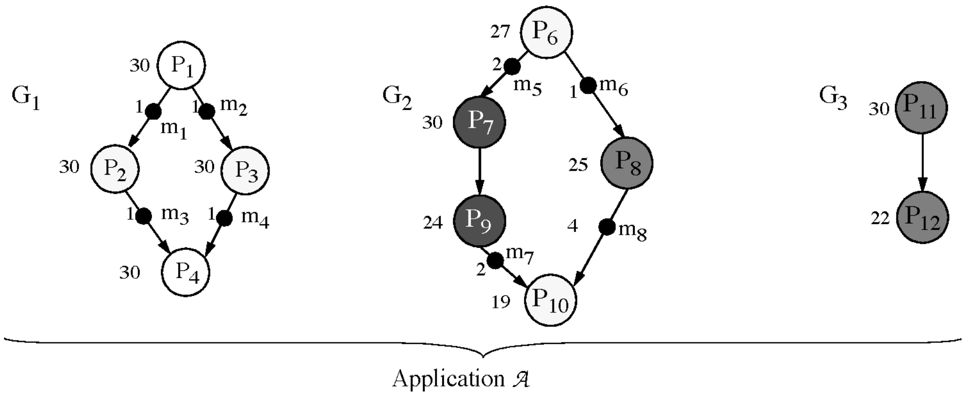


Fig. 7. Application model.

P_6 are scheduled using SCS P_2 and P_5 are scheduled using FPS, and processes P_3 and P_4 are scheduled with EDF. The priorities of the FPS and EDF processes are indicated in the figure. The arrival time of these processes is depicted with an upwards-pointing arrow. Under these assumptions, Figure 6b presents the worst-case response time of each process. The SCS processes, P_1 and P_6, will never compete for a resource because their synchronization is performed based on the schedule table. Moreover, since SCS processes are nonpreemptable and their start time is offline fixed in the schedule table, they also have the highest priority (denoted with priority level "0" in the figure). FPS and EDF processes can only be executed in the *slack* of the SCS schedule table.

FPS and EDF processes are scheduled based on their priorities. Thus, a higher priority process such as P_2 will preempt a lower priority process such as P_3. In order to integrate EDF processes with FPS, we use the approach in González Harbour and Palencia [2003] by assuming that FPS priorities are not unique, and that a group of processes having the same FPS priority on a processor is to be scheduled with EDF. Thus, whenever the FPS scheduler notices ready processes that share the same priority level, it will invoke the EDF scheduler which will schedule those processes based on their deadlines. Such a situation is present in Figure 6b for processes P_3 and P_4. There can be several such EDF priority levels within a process set on a processor. Higher priority EDF processes can interrupt lower priority FPS (as is the case with P_3 and P_4, which preempt P_5) and EDF processes. Lower priority EDF processes will be interrupted by both higher priority FPS and EDF processes, as well as SCS processes.

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

3. APPLICATION MODEL

In this article, we model an application A as a set of process graphs $G_i \in \mathcal{A}$ (see Figure 7). Nodes in the graph represent processes and arcs represent dependencies between the connected processes. A *process* is a sequence of computations (corresponding to several basic blocks in a programming language) which starts when all its inputs are available. When it finishes executing, the process produces its output values. Processes can be preemptable or nonpreemptable. Nonpreemptable processes are those that cannot be interrupted during their execution, and are scheduled using SCS. Preemptable processes can be interrupted during their execution, and are scheduled with FPS or EDF. For example, a higher priority process has to be activated to service an event; in this case, the lower priority process will be temporarily preempted until the higher priority process finishes its execution.

A process graph is polar, which means that there are two nodes, called the source and the sink, that conventionally represent the first and last process. If needed, these nodes are introduced as dummy processes so that all other nodes in the graph are successors of the source and predecessors of the sink, respectively. In addition, the graphs are acyclic: functional loops are unrolled based on known loop bounds.

The communication time between processes mapped on the same processor is considered to be part of the process worst-case execution time and is not modeled explicitly. Communication between processes mapped to different processors is performed by passing messages over the buses and, if needed, through the gateway. Such message passing is modeled as a communication process inserted on the arc connecting the sender and the receiver process (the black dots in Figure 7).

Each process $P_i \in$ is mapped on a node $M(P_i)$ (the mapping is represented by shading in Figure 7), and has a worst-case execution time C_i on that node (depicted to the left of each node). The designer can manually provide such worst-case times, or tools can be used in order to determine the worst-case execution time of a piece of code on a given processor [Puschner and Burns 2000].

For each message, we know its size (in bytes, indicated to its left) and its period, which is identical to that of the sender process. Processes and messages activated based on events also have uniquely assigned priorities, $priority_{P_i}$ for processes and $priority_{m_i}$ for messages.

All processes and messages belonging to a process graph G_i have the same period, $T_i = T_{G_i}$, which is the period of the process graph. A deadline, D_{G_i}, is imposed on each process graph, G_i. Deadlines can also be placed locally on processes. Release times as well as individual deadlines of some processes can be easily modelled by inserting dummy nodes between certain processes and the source or sink node, respectively. These dummy nodes represent processes with a certain execution time, but which are not allocated to any node.

3.1 Conditional Process Graph

One drawback of dataflow process graphs is that they are not suitable for capturing the control aspects of an application. For example, it can happen that the execution of some processes can also depend on conditions computed by previously executed processes. Several researchers have proposed extensions to the dataflow process graph model in order to capture these control dependencies [Eles et al. 1998; Thiele et al. 1999, Klaus and Huss 2001].

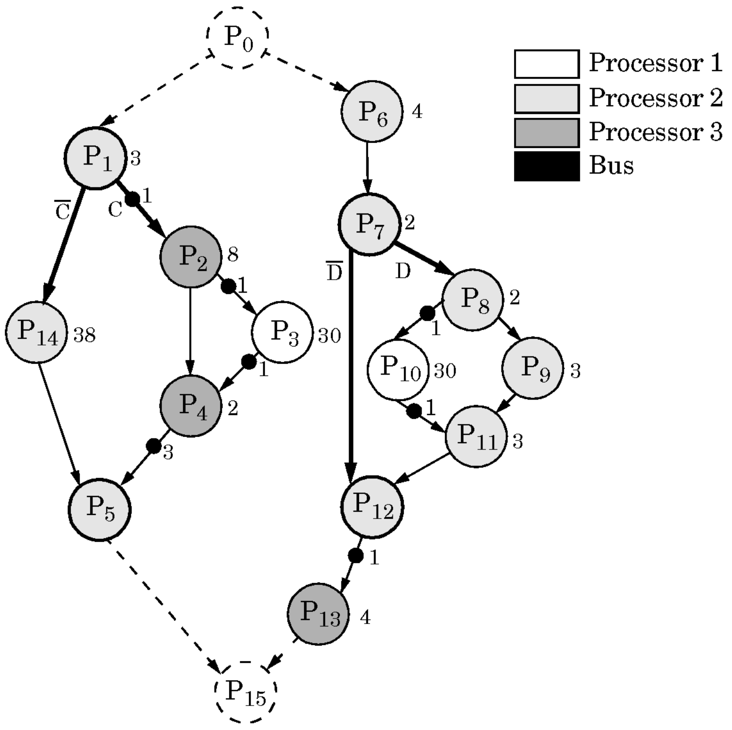


Fig. 8. A conditional process graph example.

We have proposed [Eles et al. 1998, 2000] the *conditional* process graph (CPG) as an abstract model for representing the behavior of an application, which not only captures both dataflow and the flow of control, but is also suitable for handling timing aspects.

Such a CPG is depicted in Figure 8: P_0 and P_{15} are the source and sink nodes, respectively. The nodes denoted as $P_1, P_2, \ldots, P_{14}$ are "ordinary" processes specified by the designer. Conditional *edges* (represented by thick lines in Figure 8) have associated a condition value. In Figure 8, processes $P_1$and P_7 have conditional edges at their output. We call a node with conditional edges at its output a *disjunction* node (and the corresponding process a disjunction process). A disjunction process has one associated condition, the value of which it computes. Alternative paths starting from a disjunction node, which correspond to complementary values of the condition, are disjoint, and they meet in a so-called node (with the corresponding process called a *conjunction process*).[2] In Figure 8, circles representing conjunction and disjunction nodes are depicted with thick borders. The alternative paths starting from disjunction node P_1, which computes condition C, meet in conjunction node P_5. We assume that conditions are independent and alternatives starting from different processes cannot depend on the same condition.

[2]If no process is specified on an alternative path, it is modeled by a conditional edge from the disjunction to the corresponding conjunction node (a communication process may be inserted on this edge at mapping).

The conditional process graph has the following execution semantic:

—A conjunction process can be activated after messages coming on one of the alternative paths have arrived (as opposed to "ordinary" processes, which are activated only when *all* their inputs have arrived).

—A Boolean expression X_{P_i}, called a *guard*, can be associated to each node P_i in the graph. It represents the necessary conditions for the respective process to be activated. X_{P_i} is not only necessary, but also sufficient, for process P_i to be activated during a given system execution.

—Transmission on conditional edges takes place only if the associated condition value is true and not, as on simple edges, for each activation of the input process P_i.

At a given activation of the system, only a certain subset of the total amount of processes is executed, and this subset differs from one activation to the other. For example, if condition C, calculated by process P_1, is true and D, computed by P_7, is false, the $\{P_1–P_7, P_{12}, P_{13}\}$ process set is activated, while if C is false and D is true, processes in the set $\{P_1, P_{14}, P_5–P_{13}\}$ are activated instead. Because the values of the conditions are unpredictable, the decision as to which process to activate and at which time has to be made without knowing which values the conditions will later get. On the other hand, at a certain moment during execution, once the values of some of the conditions are already known, they have to be used as the basis for making the best possible decisions on when and which process to activate in order to reduce the schedule length [Eles et al. 2000].

In Eles et al. [2000] and Pop et al. [2000] we have presented scheduling and schedulability analysis for conditional process graphs. In order to keep the presentation focused, in the remainder of the article we will not further explore specific issues related to the conditional nature of application graphs.

4. SYSTEM-LEVEL DESIGN

The aim of a design methodology is to coordinate design tasks such that the time-to-market is minimized, design constraints are satisfied, and various parameters are optimized.

A system-level design flow is illustrated in Figure 9. According to the figure, system-level design tasks take as input the specification of the system and its requirements (system model), and produce a model of the system implementation which is later synthesized into hardware and software.

One of the most important components of any system design methodology is the definition of a system *platform*. Such a platform consists of a hardware infrastructure, together with software components that will be used for several product versions and will be shared with other product lines in hopes of reducing costs and the time-to-market [Keutzer et al. 2000]. In this article we consider a heterogeneous platform, as presented in Section 2.4.

In this work we concentrate on three system-level design tasks in the context of heterogeneous systems: mapping of functionality, bus access optimization, and scheduling policy assignment.

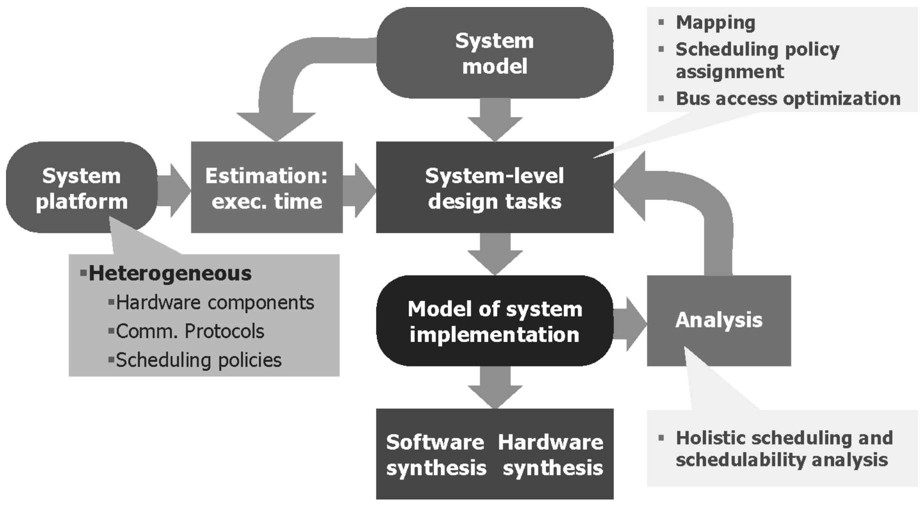


Fig. 9. System-Level design flow example.

An essential component of such a design flow is an adequate analysis technique (the "analysis" box in Figure 9) to drive the design space exploration. In order to (automatically) make informed design decisions, we developed accurate analysis techniques that:

—can handle distributed applications, data, and control dependencies;
—accurately take into account the details of the communication protocols; and
—handle heterogeneous scheduling policies.

In the next section we present the holistic scheduling and schedulability analysis approach we have proposed for heterogeneous distributed embedded real-time systems.

5. HOLISTIC SCHEDULING AND SCHEDULABILITY ANALYSIS

There is a large body of research [Kopetz 1997; Audsley et al. 1995, Balarin et al. 1998] related to scheduling and schedulability analysis, with results having been incorporated into analysis tools such as TimeWiz [TimeSys 2005], RapidRMA [Tri-Pacific Software 2005], RTA-OSEK Planner [ETAS 2005], Aires [Gu et al. 2003], Volcano Network Architect [Mentor Graphics 2005], and MAST [González Harbour 2001]. The tools determine if the timing constraints of the functionality are met and support the designer in exploring several design scenarios in order to facilitate the design of optimized implementations.

However, none of the existing approaches offer a holistic analysis for heterogeneous distributed systems, with the exception of SymTA/S [Richter et al. 2003], which is based on a compositional approach for global analysis. The holistic approach we propose can handle applications distributed across different types of networks (e.g., CAN, FlexRay, TTP) consisting of nodes that may use different scheduling policies (e.g., static cyclic scheduling, fixed priority preemptive scheduling, earliest deadline first).

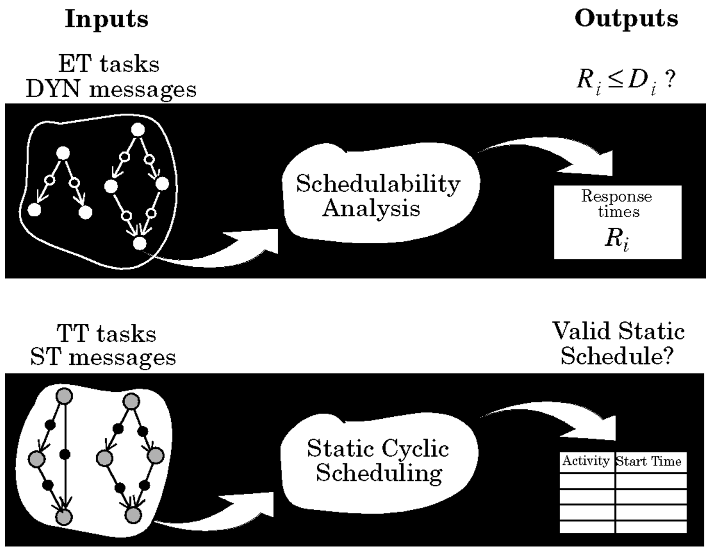


Fig. 10. Timing analysis for isolated TT and ET domains.

5.1 Isolated Domains

If the TT and ET domains are isolated (i.e., there is no resource sharing or communication between them) as depicted in Figure 10, the timing analysis can be performed separately for each domain. This is the case, for example, if we have a two-cluster system with a TTC and an ETC (see Section 2.4.1) where there is no time-critical communication exchanged between the two clusters.

The basic idea of the analysis in Figure 10 is that SCS processes can be considered schedulable if it is possible to build a schedule table such that the timing requirements are satisfied [Eles et al. 2000]. For this purpose, we use the list scheduling-based approach presented in Eles et al. [2000]. List scheduling heuristics are based on priority lists from which processes are extracted in order to be scheduled at certain moments. We use the *modified partial critical path* priority function presented in Eles et al. [2000]. For FPS and EDF processes and messages, the answer to whether or not they are schedulable is given by a *schedulability analysis*. To this end, we use a *response time analysis* where the schedulability test consists of a comparison between the worst-case response time, R_i, of a process, P_i, and its deadline, D_i.

Preemptive scheduling of independent processes with static priorities running on single-processor architectures has its roots in the work of Liu and Layland [1973]. The approach was later extended to accommodate more general computational models and has also been applied to distributed systems [Tindell and Clark 1994]. The reader is referred to Audsley et al. [1995], Balarin et al. [1998], Stankovic and Ramamritham [1993], and Sha et al. [2004] for surveys on this topic. Static cyclic scheduling of a set of data dependent soft-ware processes on a multiprocessor architecture has also been intensively researched [Kopetz 1997, Xu and Parnas 2000].

In our application model in Section 3, we capture data dependencies and precedence relationships between two processes, P_i and P_j, using an edge, e_{ij}. This means that P_j cannot start before P_i has finished executing. Xu and Parnas [1990] were the first to propose an approach that can provide efficient solutions to applications that exhibit such dependencies within the context of static cyclic scheduling. Advances in the area of fixed priority preemptive scheduling later showed that such applications can also be handled with other scheduling strategies [Audsley et al. 1993].

One way of dealing with data dependencies between processes in the context of static priority-based scheduling was indirectly addressed by the extensions proposed for the schedulability analysis of distributed systems through the use of the *release jitter* [Tindell and Clark 1994]. Release jitter is the worst-case delay between the arrival of a process and its release (when it is placed in the ready-queue for the processor) and can include the communication delay due to the transmission of a message on the communication channel.

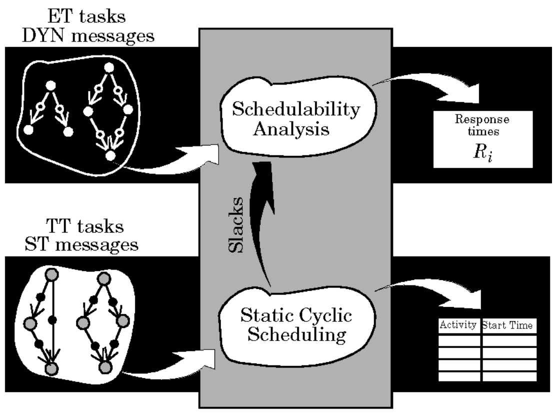
In Tindell and Clark [1994] and Yen and Wolf [1997] time *offset* relationships and *phases*, respectively, were used to model data dependencies. Offset and phase are similar concepts that express the existence of a fixed interval in time between the arrivals of sets of processes. The authors showed that by introducing such concepts into the computational model, the uncertainty of the analysis is significantly reduced when bounding the time behavior of a system with data-dependent processes. The concept of *dynamic* offsets was later introduced and used to model data dependencies [Palencia and González Harbour 1998].

5.2 TT and ET Resource Sharing

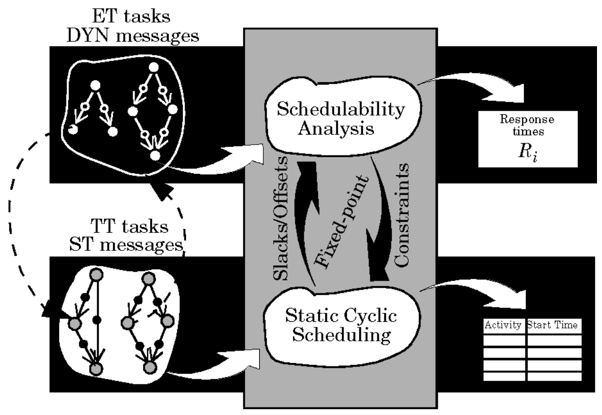
Figure 11a illustrates our strategy for scheduling and schedulability analysis of heterogeneous systems where TT and ET activities share the same resource (but there is no communication between processes in the two domains). This is, for example, the case in a hybrid-bus system (such as UCM or FlexRay), and where processes are scheduled using both time- and event-triggered scheduling policies but the TT and ET processes do not exchange time-critical communication.

TT activities are statically scheduled and, as an output, a static cyclic schedule will be produced. Similarly, the worst-case response times of ET activities are determined using the schedulability analysis presented in Pop et al. [2005b]. As a result, the system is considered schedulable if the static schedule is valid and if the ET activities are guaranteed to meet their deadlines.

In the case when TT and ET activities share the same resource, the calculation of the worst-case response times for ET activities has to consider the preemption from the TT domain. Thus, the ET processes can only execute in the slack of the TT schedule table (see the arrow labeled "Slacks" in Figure 11a). This means that the scheduling algorithm will have to generate a SCS schedule which not only guarantees that TT activities meet their deadlines, but also that the interference introduced from such a schedule will not increase to an unacceptable extent the response times of ET activities. In conclusion, an efficient scheduling algorithm requires a close interaction between the static scheduling of TT activities and the schedulability analysis of ET activities.



(a) TT and ET resource sharing



(b) TT and ET resource sharing and communication

Fig. 11. Scheduling and schedulability analysis for heterogeneous systems.

To solve the problem of finding a schedulable system we have to consider several aspects:

—When performing the schedulability analysis for the FPS and EDF processes and messages, we have to take into consideration the interference from the statically scheduled activities in the system. We have extended the analysis in Palencia and González Harbour [1998] to handle this aspect [Pop et al. 2002].

—Among the possible correct schedules for SCS activities, it is important to build one which favours as much as possible the degrees of schedulability of FPS and EDF activities. We have proposed such a static scheduling algorithm in Pop et al. [2002].

5.3 TT and ET Communication

In the case when TT and ET processes communicate, this interdomain communication creates a circular dependency: the static schedules determined for the TT activities influence through the offsets the response times of the ET processes (the "Slacks/Offsets" arrow in Figure 11b), which in their turn influence the schedule table construction for the TT activities (the "Constraints" arrow in the figure). Our holistic scheduling algorithm loops until the worst-case response times of the ET tasks can no longer be tightened (the "fixed point" loop). In Figure 11b we illustrate the case when we have both sharing and communication.

In our response time analysis we consider the influences between the two clusters by making the following observations:

—The offsets of ET processes have to be set by a scheduling algorithm such that the precedence relationships are preserved. This means that if process P_B depends on process P_A, the following condition must hold: $O_B \geq O_A + R_A$ (where O_i denotes the offset of process P_i). Note that for the TT processes which receive messages from ET processes, this translates to setting the start times of the processes such that a process is not activated before the worst-case arrival time of the message from an ET process. In general, TT offsets are set such that all the necessary messages are present at the process invocation.

—The worst-case response times for the interdomain messages have to be calculated according to the schedulability analysis we have proposed in Pop et al. [2003].

The holistic scheduling algorithm for the general case of resource-sharing communicating TT and ET domains (see Sections 5.2 and 5.3) has been implemented as a Holistic Scheduling algorithm, and is presented in detail in Pop et al. [2005a].

6. DESIGN OPTIMIZATION

Considering the types of systems[3] and applications described in Sections 2.4 and 3, respectively, and using the analysis outlined in the previous section, several design optimization problems can be addressed.

The research presented in this article concentrates on the following system-level design tasks:

—Mapping: Section 6.1 presents the classical problem of the mapping of functionality, which has to take into account the details of the bus protocol.

—Communication synthesis: Section 6.2 introduces the bus access optimization problem.

—Scheduling policy assignment: Section 6.3 briefly outlines the problem of deciding the scheduling policy for each process of the application.

The goal of these optimization problems is to produce an implementation which meets all the timing constraints (i.e., the application is schedulable).

[3]We consider single-bus systems in this section, but the approach can be extended to other types of topologies.

In order to drive our optimization algorithms towards schedulable solutions, we characterize a given design alternative using the degree of schedulability of the application. The *degree of schedulability* is calculated as:

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

where n is the number of processes in the application, R_i is the worst-case response time of a process P_i, and D_i its deadline. The worst-case response times are calculated by the HolisticScheduling algorithm mentioned in the previous section.

If the application is not schedulable, the term c_1 will be positive and, in this case, the cost function is equal to c_1. However, if the process set is schedulable, $c_1 = 0$, and we use c_2 as a cost function as it is able to differentiate between two alternatives, both leading to a schedulable process set. For a given set of optimization parameters leading to a schedulable process set, a smaller c_2 means that we have improved the worst-case response times of the processes, so the application can potentially be implemented on a cheaper hardware architecture (with slower processors and/or buses).

6.1 Mapping

The designer might have already decided on the mapping for a part of the processes. For example, certain processes, due to constraints such as having to be close to sensors/actuators, have to be physically located in a particular hardware unit. They represent the set $\mathcal{P}^M \subseteq \mathcal{P}$ of already mapped processes. Consequently, we denote with $\mathcal{P}^* = \mathcal{P}/\mathcal{P}^M$ the processes for which the mapping has not yet been decided.

For a distributed heterogeneous system, the communication infrastructure has an important impact on the design and, in particular, on the mapping decisions. Let us consider the example in Figure 12 where we have an application consisting of four processes, P_1 to P_4, and an architecture with three nodes, N_1 to N_3. Thus, the bus, using an UCM protocol, will have three static slots, S_1 to S_3, one for each node. The sequence of slots on the bus is S_2, followed by S_1, and then S_3. We have decided to place a single dynamic phase within a bus cycle (labeled "DYN" and depicted in gray) preceding the three static slots (see Section 2.4.2 for details about the bus protocol). We assume that P_1, P_3, and P_4 are mapped on node N_1, and we are interested in mapping process P_2. Process P_2 is allowed to be mapped on node N_2 or on node N_3, and its execution times are depicted in the table labeled "mapping." Note that an "x" in the table means that the process is not allowed to be mapped on that node. Moreover, the scheduling policy is fixed for each process (see the table caption "SPA"— scheduling policy assignment— in Figure 12) such that all processes are scheduled with SCS and communicate through the static slots. Similarly, an "x" in the SPA table means that the process cannot be scheduled with the corresponding scheduling policy.

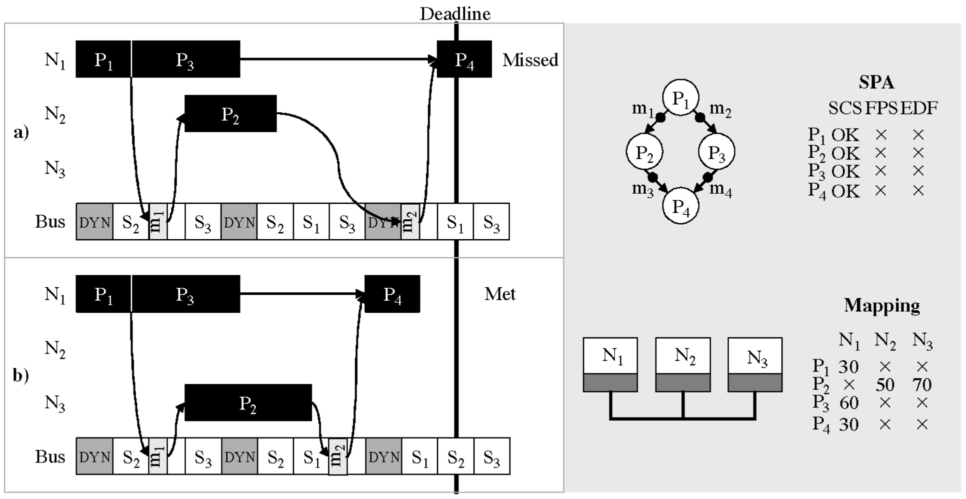


Fig. 12. Mapping example.

In order to meet the deadline, one would map P_2 on the node it executes the fastest, that is, node N_2 (see Figure 12a). However, this will lead to a deadline miss due to the bus slot configuration which introduces communication delays. The application will meet the deadline only if P_2 is, counterintuitively, mapped on the slower node, that is, node N_3, as depicted in Figure 12b.

6.2 Bus Access Optimization

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 timing requirements. The 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 static phases, the parameters to be fixed are the order, number, and length of slots assigned to the different nodes. For example, consider the situation in Figure 13, where process P_1 is mapped on node N_1 and sends a message m to process P_2, which is mapped on node N_2. In case (a) process P_1 misses the start of the ST $Slot_1$ and therefore, message m will be sent during the next bus cycle, causing the receiver process P_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 P_2 will meet its deadline. The resulting situation can be further improved, as can be seen in Figure 13(c) where process P_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).

6.3 Scheduling Policy Assignment

Very often, the scheduling policy assignment (SPA) and mapping decisions are made based on the experience and preferences of the designer, considering

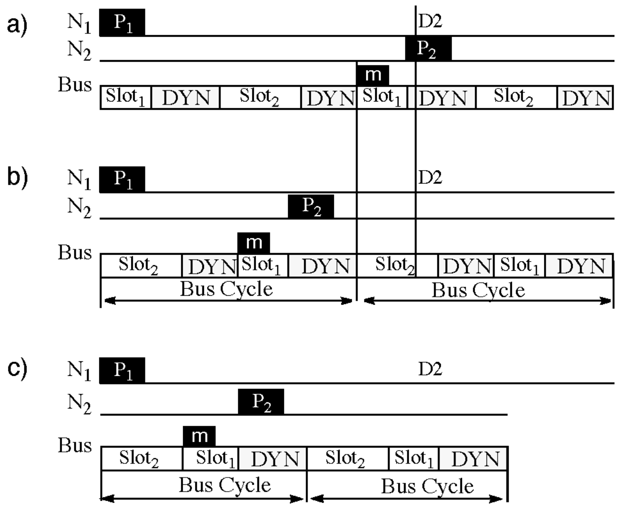


Fig. 13. Optimization of bus access cyle.

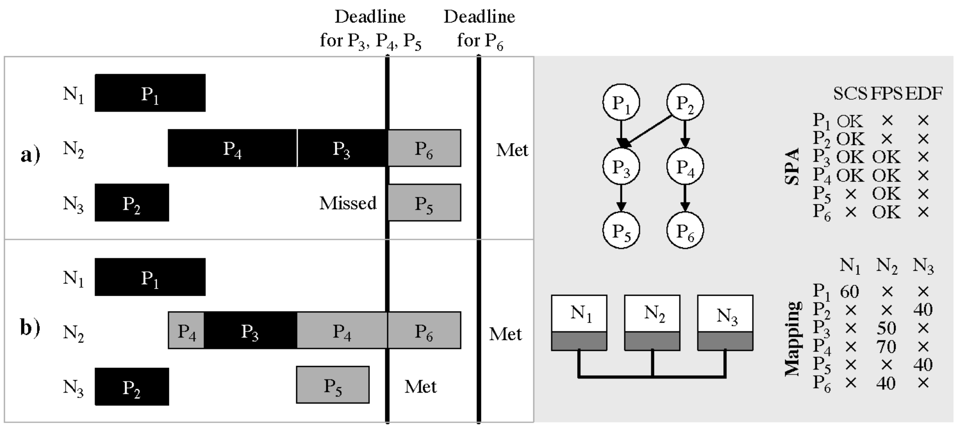
aspects such as the functionality implemented by the process, the hardness of the constraints, sensitivity to jitter, etc. Moreover, due to legacy constraints, the mapping and scheduling policy of certain processes might be fixed.

Thus, we denote with $\mathcal{P}_{SCS} \subseteq \mathcal{P}$ the subset of processes for which the designer has assigned SCS, $\mathcal{P}_{EPS} \subseteq \mathcal{P}$ contains processes to which FPS is assigned, while $\mathcal{P}_{EDF} \subseteq \mathcal{P}$ contains those processes for which the designer has decided to use the EDF scheduling policy. There are, however, processes which do not exhibit certain particular features or requirements which would obviously lead to their scheduling as SCS, FPS, or EDF activities. The subset $\mathcal{P}^+ = \mathcal{P} \backslash (\mathcal{P}_{SCS} \cup \mathcal{P}_{EPS} \cup \mathcal{P}_{EDF})$ of processes could be assigned any scheduling policy. Decisions concerning the SPA to this set of activities can lead to various tradeoffs concerning, for example, the schedulability properties of the system, the size of the schedule tables, the utilization of resources, etc.

Let us illustrate some of the issues related to SPA in such a context. In the example presented in Figure 14 we have an application[4] with six processes, P_1 to P_6, and three nodes, N_1, N_2, and N_3. The worst-case execution times on each node are given in the table labeled "Mapping" (the mapping of processes is thus fixed for this example). The scheduling policy assignment is captured by the table labeled "SPA." Thus, processes P_1 and P_2 are scheduled using SCS, while processes P_5 and P_6 are scheduled with FPS. We have to decide which scheduling policy to use for processes P_3 and P_4, which can be scheduled with any of the SCS or FPS scheduling policies.

We can observe that the scheduling of P_3 and P_4 has a strong impact on their successors, P_5 and P_6, respectively. Thus, we would like to schedule P_4 such that not only P_3 can start on time, but P_4 also starts soon enough to allow P_6 to meet its deadline. As we can see from Figure 14a, this is impossible to achieve by scheduling P_3 and P_4 with SCS. Although P_3 meets its deadline, it finishes

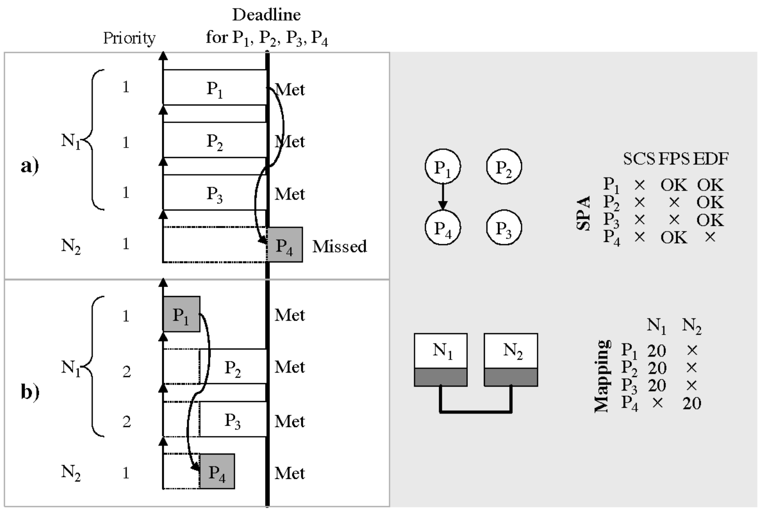
[4]Communications are ignored for the examples in this subsection.



	SCS	FPS	EDF
P_1	OK	×	×
P_2	OK	×	×
P_3	OK	OK	×
P_4	OK	OK	×
P_5	×	OK	×
P_6	×	OK	×

	N_1	N_2	N_3
P_1	60	×	×
P_2	×	×	40
P_3	×	50	×
P_4	×	70	×
P_5	×	×	40
P_6	×	40	×

Fig. 14. Scheduling policy assignment example #1.



	SCS	FPS	EDF
P_1	×	OK	OK
P_2	×	×	OK
P_3	×	×	OK
P_4	×	OK	×

	N_1	N_2
P_1	20	×
P_2	20	×
P_3	20	×
P_4	×	20

Fig. 15. Scheduling policy assignment example #2.

too late for P_5 to finish on deadline. However, if we schedule P_4 with FPS, for example, as in Figure 14b, both deadlines are met. In this case, P_3 finishes on time to allow P_5 to meet its deadline. Moreover, although P_4 is preempted by P_3, it still finishes on time, meets its deadline, and allows $P_6$to meet its deadline as well. Note that using EDF for P_4 (if it would share the same priority level with P_6, for example) will also meet the deadline.

For a given set of preemptable processes, the example in Figure 15 illustrates the optimization of the assignment of FPS and EDF policies. In Figure 15 we have an application composed of four processes running on two nodes. Processes P_1, P_2, and P_3 are mapped on node N_1, while process P_4 runs on N_2. Process P_4 is data, dependent on process P_1. All processes in the system have the same worst-case execution times (20 ms), deadlines (60 ms), and periods (80 ms).

Processes P_2 and P_3 are scheduled with EDF at priority level "1," P_4 with FPS, and we have to decide the scheduling policy for P_1 between EDF and FPS.

If P_1 is scheduled according to EDF, thus sharing the same priority level of "1" with the processes on node N_1, then process P_4 misses its deadline (Figure 15a). Note that in the time line for node N_1 in Figure 15, we depict several worst-case scenarios: each EDF process on node N_1 is depicted considering the worst-case interference from the other EDF processes on N_1. However, the situation changes if on node N_1 we use FPS for P_1 (i.e., changing the priority levels of P_2 and P_3 from "1" to "2"). Figure 15b shows the response times when process P_1 has the highest priority on N_1 (P_1 retains priority "1") and the other processes are running under EDF at a lower priority level (P_2 and P_3 share the lower priority "2"). Because in this situation there is no interference from processes P_2 and P_3, the worst-case response time for process P_1decreases considerably, allowing process P_4 to finish before its deadline so that the system becomes schedulable.

6.4 Problem Formulation

As an input to our optimization problem, we have an application $\mathcal{A}$, given as a set of process graphs (Section 3) and a system architecture consisting of a set $\mathcal{N}$ of nodes (Section 2). As introduced previously, $\mathcal{P}_{SCS}$, $\mathcal{P}_{EPS}$, and $\mathcal{P}_{EDF}$ are the sets of processes for which the designer has already assigned SCS, FPS, or EDF scheduling policies, respectively. Also, $\mathcal{P}^M$ is the set of already mapped processes.

As part of our problem, we are interested in:

—finding a scheduling policy assignment S for processes in $\mathcal{P}^+ = \mathcal{P} \backslash (\mathcal{P}_{SCS} \cup \mathcal{P}_{EPS} \cup \mathcal{P}_{EDF})$;
—deciding on a mapping for processes in $\mathcal{P}^* = \mathcal{P} \backslash \mathcal{P}^M$;
—determining a bus configuration $\mathcal{B}$; and
—determining the schedule table for the SCS processes and the priorities of FPS and EDF processes

such that the imposed deadlines are guaranteed to be satisfied.

In this article, we will consider the assignment of scheduling policies simultaneously with the mapping of processes to processors.

7. DESIGN OPTIMIZATION STRATEGY

The design problem formulated in the previous section is at least NP-complete (both the scheduling and the mapping problems, considered separately, are already NP-complete [Garey and Johnson 2003]). Therefore, our strategy, outlined in Figure 16, is to elaborate a heuristic and to divide the problem into several more manageable subproblems. Our optimization strategy has three steps:

(1) In the first step (lines 1–3) we decide on an initial bus access configuration $\mathcal{B}^0$ (function InitialBusAccess), and an initial policy assignment $\mathcal{S}^0$ and mapping $\mathcal{M}^0$ (function InitialMSPA). The initial bus access configuration (line 1) is determined for the ST slots by assigning, in order, nodes to the slots

```
OptimizationStrategy($\mathcal{A}$)
1  Step 1: $\mathcal{B}^0$ = InitialBusAccess($\mathcal{A}$)
2        ($\mathcal{M}^0$, $\mathcal{S}^0$) = InitialMSPA($\mathcal{A}$, $\mathcal{B}^0$)
3        if HolisticScheduling($\mathcal{A}$, $\mathcal{M}^0$, $\mathcal{B}^0$, $\mathcal{S}^0$) returns schedulable then stop end if
4  Step 2: ($\mathcal{M}$, $\mathcal{S}$, $\mathcal{B}$) = MSPAHeuristic($\mathcal{A}$, $\mathcal{M}^0$, $\mathcal{B}^0$, $\mathcal{S}^0$)
5        if HolisticScheduling($\mathcal{A}$, $\mathcal{M}$, $\mathcal{S}$, $\mathcal{B}$) returns schedulable then stop end if
6  Step 3: $\mathcal{B}$ = BusAccessOptimization($\mathcal{A}$, $\mathcal{M}$, $\mathcal{B}$, $\mathcal{S}$)
7        HolisticScheduling($\mathcal{A}$, $\mathcal{M}$, $\mathcal{B}$, $\mathcal{S}$)
end OptimizationStrategy
```

Fig. 16. The general optimization strategy.

$(S_i = N_i)$ and fixing the slot length to the minimal allowed value, which is equal to the length of the largest message in the application. Then, we add to the end of the ST slots an equal length single DYN phase. The initial scheduling policy assignment and mapping algorithm (line 2 in Figure 16) maps processes so that the amount of communication is minimized. The initial scheduling policy of processes in P^+ is set to FPS. Once an initial mapping, scheduling policy assignment, and bus configuration are obtained, the application is scheduled using the HolisticScheduling algorithm (line 3) mentioned in Section 5.

(2) If the application is schedulable, the optimization strategy stops. Otherwise, it continues with the second step by using an iterative improvement mapping and policy assignment heuristic, MSPAHeuristic (line 4), presented in the next Section, to improve the partitioning and mapping obtained in the first step.

(3) If the application is still not schedulable, we use in the third step the algorithm BusAccessOptimization presented Section 7.2, which optimizes the access to the communication infrastructure (line 6). If the application is still unschedulable, we conclude that no satisfactory implementation could be found with the available amount of resources.

7.1 Mapping and Scheduling Policy Assignment Heuristic

In Step 2 of our optimization strategy (Figure 16), the following design transformations are performed iteratively with the goal of producing a schedulable system implementation:

—change the scheduling policy of a process;
—change the mapping of a process; and
—change the priority level of an FPS of an EDF process.

Our optimization algorithm is presented in Figure 17 and implements a greedy approach in which every process in the system is iteratively mapped on each node (line 4) and assigned to each scheduling policy (line 8), under the constraints imposed by the designer. The next step involves adjustments to the bus access cycle (line 10), which are needed in case the bus cycle configuration cannot handle the minimum requirements of the current internode communication. Such adjustments are mainly based on enlargement of the static slots

```
MSPAHeuristic($\mathcal{A}$, $\mathcal{M}$, $\mathcal{B}$, $\mathcal{S}$)
1     for each activity $P_i$ in the system do
2       for each processor $N_j \in \mathcal{N}$ in the system do
3         if $P_i$ in $\mathcal{P}^*$ then -- can be remapped
4           $\mathcal{M}(P_i) = N_j$
5         end if
6         for policy = SCS, FPS do
7           if $P_i$ in $\mathcal{P}^+$ then -- the scheduling policy can be changed
8             $\mathcal{S}(P_i)$ = policy
9           end if
10          adjust bus cycle($\mathcal{A}$, $\mathcal{M}$, $\mathcal{B}$, $\mathcal{S}$)
11          recompute FPS priority levels
12          for all FPS tasks $\tau_{ab}$ sharing identical priority levels do $\mathcal{S}(\tau_{ab})$ = EDF end for
13          HolisticScheduling($\mathcal{A}$, $\mathcal{M}$, $\mathcal{B}$, $\mathcal{S}$)
14          if $\delta_{\mathcal{A}}$ < best_$\delta_{\mathcal{A}}$ then
15            best_policy$_i$ = $\mathcal{S}(P_i)$; best_processor$_i$ = $\mathcal{M}(P_i)$
16            best_$\delta_{\mathcal{A}}$ = $\delta_{\mathcal{A}}$
17          end if
18          if $\delta_{\mathcal{A}} < 0$ then
19            return best ($\mathcal{M}$, $\mathcal{B}$, $\mathcal{S}$)
20          end if
21        end for
22      end for
23    end for
end MSPAHeuristic
```

Fig. 17. Policy assignment and mapping heuristic.

or dynamic phases in the bus cycle, and are required in case the bus has to support larger messages than before. New messages may appear on the bus due to, for example, remapping of processes; consequently, there may be new TT messages that are larger than the current static slot for the sender node (or similarly, the bus will face the situation where new ET messages are larger than the largest DYN phase in the bus cycle). For more details on the subject of bus access optimization and adjustment, the reader is referred to the next section.

Before the system is analyzed for its timing properties, our heuristic also tries to optimize the priority assignment of processes running under FPS (line 11). The state-of-the-art approach for such a process is the HOPA algorithm for assigning priority levels to processes in multiprocessor systems [Gutiérrez García and González Harbour 1995]. However, due to the fact that HOPA is computationally expensive when run inside such a design optimization loop, we use a scaled-down greedy algorithm in which we drastically reduce the number of iterations needed for determining an optimized priority assignment.

Finally, the resulting system configuration is analyzed (line 13) using the scheduling and schedulability analysis algorithm outlined in Section 5. The resulting cost function (δ_A, see Section 6) will decide whether this configuration

```
BusAccessOptimization($\mathcal{A}$, $\mathcal{M}$, $\mathcal{B}$, $\mathcal{S}$)
1  for i = 1 to NrNodes
2    for j = i to NrNodes
3      swap $Slot_i$ with $Slot_j$
4      for all slot lengths $\lambda$ > min_len($Slot_i$)
5        len($Slot_i$) = $\lambda$
6        for all DYN phase lengths $\pi$
7          len($Ph_i$) = $\pi$
8          if $\delta_{\mathcal{A}} \le 0$ then stop endif
9          keep solution with lowest $\delta_{\mathcal{A}}$
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
end BusAccessOptimization
```

Fig. 18. Bus access optimization.

is better than the current best (lines 14–17). Moreover, if all activities meet their deadlines ($\delta_A < 0$), the optimization heuristic stops the exploration process and returns the current best-so-far configuration (lines 18–20).

7.2 Bus Access Optimization Heuristic

It may be the case that even after the mapping and partitioning step, some ET activities are still not schedulable. In the third step (line 6, Figure 16), our algorithm tries to remedy this problem by changing the parameters of the bus cycle, such as 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 adapted to the particular process configuration. The heuristic is illustrated in Figure 18. The algorithm iteratively looks for the right place and size of $Slot_i$ used for transmission of ST messages from $Node_i$ (outermost loops). The position of $Slot_i$ is swapped with all the positions of higher-order slots (line 3). Also, all alternative lengths (lines 4–5) of $Slot_i$ larger than its minimal allowed length (which is equal to the length of the largest ST message generated by a process 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 $\delta_{\mathcal{A}}$, and the solution with the lowest cost is selected. If $\delta_{\mathcal{A}} \le 0$, 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) also includes the value 0. Therefore, in the final bus cycle it is not necessary for each static slot to be followed by a dynamic phase. Dynamic phases introduced as the result of the previous steps can be eliminated by setting the length to $\pi = 0$. It should also be mentioned that enlarging a slot/phase can increase the schedulability by allowing several ST/DYN messages to be transmitted quickly,

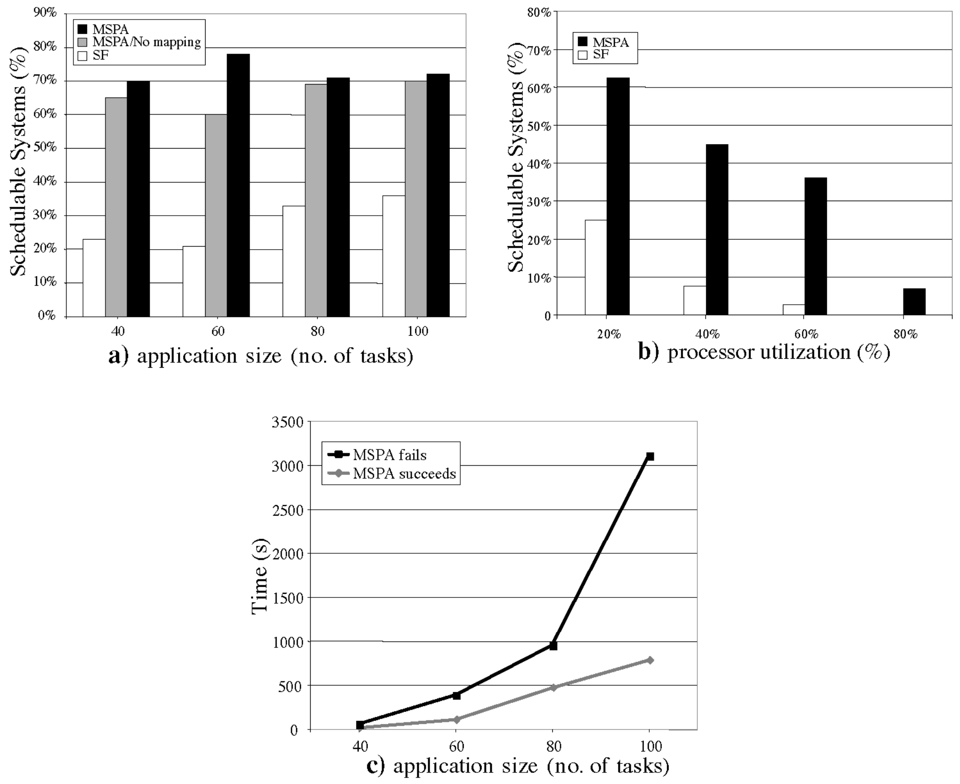


Fig. 19. Performance of the design optimisation heuristic.

one immediately 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 18 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.

8. EXPERIMENTAL RESULTS

For the evaluation of our design optimization heuristic we used synthetic applications as well as a real-life example consisting of a vehicle cruise controller. Thus, we randomly generated applications of 40, 60, 80, and 100 processes on systems with 4 processors. A total of 56 applications were generated for each dimension, thus, a total of 224 applications were used for experimental evaluation. An equal number of applications with processor utilizations of 20%, 40%, 60%, and 80% were generated for each application dimension. The bus implemented the UCM (see Section 2.4.2), and the cycle period was set to 1/100 of the longest period of the task graphs in the system. All experiments were run on an AMD AthlonXP 2400+ processor with 512 MB RAM.

We were first interested in determining the quality of our design optimization approach for hierarchically scheduled systems, the MSPA Heuristic (MSPA, see Figure 17). We compared the percentage of schedulable implementations found by MSPA with the number of schedulable solutions obtained by the InitialMSPA algorithm described in Section 7 (see Figure 16, line 2), which derives a straightforward system implementation denoted by SF. The results are depicted in Figure 19a. We can see that our MSPAHeuristic (black bars) performs very well, and finds a number of schedulable systems that are considerably and consistently higher than the number of schedulable systems obtained with the SF approach (white bars). On average, MSPA finds 44.5% more schedulable solutions than SF.

Secondly, we were interested in determining the impact of the scheduling policy assignment (SPA) decisions on the number of schedulable applications obtained. Thus, for the same applications we considered that the process mapping is fixed by the SF approach, and only the SPA is optimized. Figure 19a presents this approach, labeled "MSPA/No mapping," corresponding to the gray bars. We can see that most of the improvement over the SF approach is obtained by carefully optimizing the SPA in our MSPA Heuristic.

We also applied the bus access optimization step on top of the MSPAHeuristic. The BusAccessOptimization is able to further improve the results obtained by MSPA (note that bus access adjustment is also performed in MSPA, line 10 in Figure 17; at this point, we refer only to the bus access optimization on top of MSPA). Thus, we were able to obtain 5%, 6%, 5%, and 7% more schedulable systems for the application dimensions presented in Figure 19a, respectively.

We were also interested in finding out the impact of the processor utilization of an application on the quality of the implementations produced by our optimization heuristic. Figure 19b presents the percentage of schedulable solutions found by MSPA and SF as we ranged the utilization from 20% to 80%. We can see that SF degrades very quickly with increased utilization, with under 10% schedulable solutions for applications with 40% utilization, and with no schedulable solution for applications with 80% utilization, while MSPA is able to find a significant number of schedulable solutions even for high processor utilizations.

In Figure 19c we show the average runtimes obtained by applying our MSPA heuristic on the examples presented in Figure 19a. The upper curve illustrates the average runtime of the heuristic for those applications which were not found schedulable by our heuristic. This curve can be considered an upper bound for the computation time of our algorithm. For the examples that were found schedulable, our heuristic stopped the exploration process earlier, thus leading to smaller computation times, as shown in the lower curve of Figure 19c. We can see that, considering the complex optimization and analysis steps performed, our design optimization heuristic produces good quality results in a reasonable amount of time (for example, the heuristic will finish, on average, in less than 500 seconds for applications with 80 processes that are found schedulable).

8.1 The Vehicle Cruise Controller

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

The specification assumes that the CC will operate in an environment consisting of two clusters. There are four nodes which functionally interact with the CC system: the antilock braking system (ABS), the transmission control module (TCM), the engine control module (ECM), and the electronic throttle module (ETM).

It was decided to map the functionality (processes) of the CC over these four nodes. The ECM and ETM nodes had an 8 bit Motorola M68HC11 family CPU with 128 Kbytes of memory, while the ABS and TCM were equipped with a 16 bit Motorola M68HC12 CPU and 256 Kbytes of memory. The 16 bit CPUs were twice as fast as the 8 bit. The bus implemented UCM communication.

The process graph that models the CC had 32 processes, and is described in Pop et al. [2004]. The CC was mapped on an architecture consisting of three nodes: electronic throttle module (ETM), antilock breaking system (ABS), and transmission control module (TCM). We considered a deadline of 250 ms.

In this setting, SF failed to produce a schedulable implementation. Our design optimization heuristic MSPA was considered first so that the mapping was fixed by SF, and we only allowed reassigning of scheduling policies. After running for 29.5, the best scheduling policy allocation found by MSPA still resulted in an unschedulable system, but with a "degree of schedulability" three times higher than that obtained by SF. When mapping optimization was allowed, MSPA managed to find a schedulable system after 28.49.

9. CONCLUSIONS

Heterogeneous distributed real-time systems are used in several application areas to implement increasingly complex applications that have tight timing constraints. The heterogeneity is manifested not only at the hardware and communication protocol levels, but also at the level of the scheduling policies used. In order to reduce costs and use the available resources more efficiently, applications are distributed across several networks.

We have introduced the current state-of-the-art analysis and optimization techniques available for such systems, and have addressed in more detail a special class of heterogeneous distributed real-time embedded systems consisting of several interconnected clusters where time-triggered and event-triggered domains can share the same resource.

We have presented an analysis for such systems and outlined several characteristic design problems related to the mapping of functionality, assignment of scheduling policies, and optimization of access to the communication infrastructure. Schedulability-driven optimization heuristics were proposed for solving the aforementioned design optimization problems.

The main message of this article is that efficient analysis and optimization methods are needed and can be developed for the implementation of applications distributed over interconnected heterogeneous networks.

REFERENCES

AUDSLEY, N., BURNS, A., DAVIS, R., TINDELL, K., AND WELLINGS, A. 1995. Fixed priority preemptive scheduling: An historical perspective. *J. Real-Time Syst. 8*, 2/3, 173–198.

AUDSLEY, N., TINDELL, K., AND BURNS, A. 1993. The end of line for static cyclic scheduling? In *Proceedings of the Euromicro Workshop on Real-Time Systems*, 36–41.

BALARIN, F., LAVAGNO, L., MURTHY, P., AND SANGIOVANNI-VINCENTELLI, A. 1998. Scheduling for embedded real-time systems. *IEEE Des. Test Comput. 15*, 1, 71–82.

BERWANGER, J., PELLER, M., AND GRIESSBACH, R. 2000. A new high performance data bus system for safety-related applications. www.byteflight.de.

BOSCH GMBH. 1991. CAN Specification, Version 2.0. http://www.can.bosch.com.

BUTTAZZO, G. 2005. Rate monotonic vs. EDF: Judgment day. In *Real-Time Syst. 29*, 1, 5–26.

CHIODO, M. 1996. Automotive electronics: A major application field for hardware-software codesign. In *Hardware/Software Co-Design.* Kluwer Academic, Hingham, Mass., 295–310.

DEMMELER, T. AND GIUSTO, P. 2001. A universal communication model for an automotive system integration platform. In *Proceedings of the Design, Automation and Test in Europe Conference*, 47–54.

EAST-EEA PROJECT. 2002. ITEA Full Project Proposal. www.itea-office.org.

ECHELON. 2005. LonWorks: The LonTalk protocol specification. www.echelon.com.

ELES, P., DOBOLI, A., POP, P., AND PENG, Z. 2000. Scheduling with bus access optimization for distributed embedded systems. *IEEE Trans. VLSI Syst. 8*, 5, 472–491.

ELES, P., KUCHCINSKI, K., PENG, Z., DOBOLI, A., AND POP, P. 1998. Scheduling of conditional process graphs for the synthesis of embedded systems. In *Proceedings of the Design Automation and Test in Europe Conference*, 132–139.

ETAS. 2005. RTA-OSEK planner. en.etasgroup.com/products/rta.

FLEXRAY GROUP. 2005. FlexRay Requirements Specification, Version 2.1 Rev. A. www.flexray.com.

GAREY, M. R. AND JOHNSON, D. S. 2003. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York.

GONZÁLEZ HARBOUR, M., GUTIÉRREZ GARCÍA, J. J., PALENCIA GUTIÉRREZ, J. C., AND DRAKE MOYANO, J. M. 2001. MAST: Modeling and analysis suite for real time applications. In *Proceedings of the 13th Euromicro Conference on Real-Time Systems*, 125–134.

GONZALÉZ HARBOUR, M. AND PALENCIA, J. C. 2003. Response time analysis for tasks scheduled under EDF within fixed priorities. In *Proceedings of the Real-Time Systems Symposium*, 200–209.

GU, Z., WANG, S. KODASE, S., AND SHIN, K. G. 2003. An end-to-end tool chain for multi-view modeling and analysis of avionics mission computing software. In *Proceedings of the 24th IEEE International Real-Time Systems Symposium*.

GUTIÉRREZ GARCÍA, J. J. AND GONZÁLEZ HARBOUR, M. 1995. Optimized priority assignment for tasks and messages in distributed hard real-time systems. In *Proceedings of the Workshop on Parallel and Distributed Real-Time Systems*, 124–132.

HANSEN, P. 2002. *The Hansen Report on Automotive Electronics*. www.hansenreport.com.

HOYME, K. AND DRISCOLL, K. 1992. SAFEbus. *IEEE Aerospace Electron. Syst. Magazine 8*, 3, 34–39.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. 2002. Road vehicles—Controller area network (CAN)—Part 4: Time-triggered communication. ISO/DIS 11898–4.

JOST, K. 2001. From fly-by-wire to drive-by-wire. *Automotive Eng. International.*

KEUTZER, K., MALIK, S., AND NEWTON, A. R. 2000. System-Level design: Orthogonalization of concerns and platform-based design. *IEEE Trans. Comput. Aided Des. Integrated Circuits Syst. 19*, 12, 1523–1543.

KLAUS, S. AND HUSS, S. A. 2001. Interrelation of specification method and scheduling results in embedded system design. In *Proceedings of the ECSI International Forum on Design Languages*.

KOPETZ, H. 1997. *Real-Time Systems—Design Principles for Distributed Embedded Applications*. Kluwer Academic, Hingham, Mass.

KOPETZ, H. 1999. Automotive electronics. In *Proceedings of the 11th Euromicro Conference on Real-Time Systems*, 132–140.

KOPETZ, H. AND BAUER, G. 2003. The time-triggered architecture. *Proceedings IEEE 91*, 1, 112–126.

LIN CONSORTIUM. 2005. *Local Interconnect Network Protocol Specification*. www.lin-subbus.org

LIVANI, M. A. AND KAISER, J. 1998. EDF consensus on CAN bus access for dynamic real-time applications. In *Proceedings of the IPPS/SPDP Workshops*. Lecture Notes in Computer Science, vol. 1586, 1088–1097.

LIU, C. L. AND LAYLAND, J. W. 1973. Scheduling algorithms for multiprogramming in a hard- real-time environment. *J. ACM 20*, 1, 46–61.

LÖNN, H. AND AXELSSON, J. 1999. A comparison of fixed-priority and static cyclic scheduling for distributed automotive control applications. In *Proceedings of the Euromicro Conference on Real-Time Systems*, 142–149.

MENTOR GRAPHICS. 2005. Volcano Network Architect. www.mentor.com/products/vnd/network_design_tools/vna.

MINER, P. S. 2000. Analysis of the SPIDER fault-tolerance protocols. In *Proceedings of the 5th NASA Langley Formal Methods Workshop*.

NAVET, N., SONG, Y., SIMONOT-LION, F., AND WILERT, C. 2005. Trends in automotive communication systems. In *Proceedings IEEE 93*, 6, 1204–1223.

PALENCIA, J. C. AND GONZÁLEZ HARBOUR, M. 1998. Schedulability analysis for tasks with static and dynamic offsets. In *Proceedings of the 19th IEEE Real-Time Systems Symposium*, 26–37.

PALENCIA, J. C. AND GONZÁLEZ HARBOUR, M. 2003. Offset-Based response time analysis of distributed systems scheduled under EDF. In *Proceedings of the Euromicro Conference on Real-Time Systems*, 3–12.

POP, P., ELES, P., AND PENG, Z. 2000. Schedulability analysis for systems with data and control dependencies. In *Proceedings of the 12th Euromicro Conference on Real-Time Systems*, 201–208.

POP, T., ELES, P., AND PENG, Z. 2002. Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems. In *Proceedings of the International Symposium on Hardware/Software Codesign*, 187–192.

POP, P., ELES, P., AND PENG, Z. 2003. Schedulability analysis and optimization for the synthesis of multi-cluster distributed embedded systems. *IEEE Comput. Digital Techniques J. 150*, 5, 303–312.

POP, P., ELES, P., AND PENG, Z. 2004. *Analysis and Synthesis of Distributed Real-Time Embedded Systems*. Kluwer Academic, Hingham, Mass.

POP, P., ELES, P., AND PENG, Z. 2005a. Schedulability-driven frame packing for multi-cluster distributed embedded systems. *ACM Trans. Embedded Comput. Syst. 4*, 1, 112–140.

POP, T., POP P., ELES, P., AND PENG, Z. 2005b. Optimization of hierarchically scheduled heterogeneous embedded systems. In *Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 67–71.

PROFIBUS INTERNATIONAL. 2005. PROFIBUS DP Specification. www.profibus.com.

PUSCHNER, P. AND BURNS, A. 2000. A review of worst-case execution-time analyses. *J. Real-Time Syst.*, *18*, 115–128.

RICHTER, K., JERSAK, M., AND ERNST, R. 2003. A formal approach to MpSoC performance verification. *IEEE Comput. 36*, 4, 60–67.

RUSHBY, J. 2001. Bus Architectures for Safety-Critical Embedded Systems. Lecture Notes in Computer Science, vol.2211, Springer Verlag, 306–323.

SAE VEHICLE NETWORK FOR MULTIPLEXING AND DATA COMMUNICATIONS STANDARDS COMMITTEE. 1994. *SAE J1850 Standard*.

SHA, L. ABDELZAHER, T., ARZEN, K. E., CERVIN, A., BAKER, T., BURNS, A., BUTTAZZO, G., CACCAMO, M., LEHOCZKY, J., AND MOK, A. K. 2004. Real time scheduling theory: A historical perspective. In *Real-Time Syst. 28*, 101–155.

TIMESYS. 2005. TimeWiz. www.timesys.com

TINDELL, K. AND CLARK, J. 1994. Holistic schedulability analysis for distributed hard real-time systems. *Microprocess. Microprogram. 40*, 117–134.

TINDELL, K., BURNS, A., AND WELLINGS, A. 1995. Calculating CAN message response times. *Control Eng. Practice 3*, 8, 1163–1169.

THIELE, L., STREHL, K., ZIEGENGEIN, D., ERNST, R., AND TEICH, J. 1999. FunState—An internal design representation for codesign. In *Proceedings of the International Conference on Computer-Aided Design*, 558–565.

TRI-PAC SOFTWARE. 2005. RapidRMA. www.tripac.com.

STANKOVIC, J. A. AND RAMAMRITHAM, K. 1993. *Advances in Real-Time Systems*. IEEE Computer Society Press, Los Alamitos, Calif.

X-BY-WIRE CONSORTIUM. 1998. X-By-Wire: Safety Related Fault Tolerant Systems in Vehicles. www.vmars.tuwien.ac.at/projects/xbywire.

XU, J. AND PARNAS, D. L. 1990. Scheduling processes with release times, deadlines, precedence and exclusion relations. *IEEE Trans. Softw. Eng. 16*, 3, 360–369.

XU, J. AND PARNAS, D. L. 1993. On satisfying timing constraints in hard-real-time systems. *IEEE Trans. Softw. Eng. 19*, 1, 70–84.

XU, J. AND PARNAS, D. L. 2000. Priority scheduling versus pre-run-time scheduling. *J. Real Time Syst.*, *18*, 1, 7–24.

YEN, T. Y. AND WOLF, W. 1997. *Hardware-Software Co-Synthesis of Distributed Embedded Systems*. Kluwer Academic, Hingham, Mass.

Received January 2006; accepted May 2006