Analysis and optimisation of heterogeneous real-time embedded systems

Pop, Paul; Eles, Petru; Peng, Zebo

Published in:
IEE Proceedings - Computers and digital Techniques

Link to article, DOI:
10.1049/ip-cdt:20045069

Publication date:
2005

Document Version
Publisher's PDF, also known as Version of record

Analysis and optimisation of heterogeneous real-time embedded systems

P. Pop, P. Eles and Z. Peng

Abstract: 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 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. In this context, the task of designing such systems is becoming increasingly important and difficult at the same time. The success of such new design methods depends on the availability of analysis and optimisation techniques. Analysis and optimisation techniques for heterogeneous real-time embedded systems are presented in the paper. The authors address in more detail a particular class of such systems called multi-clusters, composed of several networks interconnected via gateways. They present a schedulability analysis for safety-critical applications distributed on multi-cluster systems and briefly highlight characteristic design optimisation problems: the partitioning and mapping of functionality, and the packing of application messages to frames. Optimisation heuristics for frame packing aimed at producing a schedulable system are presented. Extensive experiments and a real-life example show the efficiency of the frame-packing approach.

1 Introduction

Embedded real-time systems have to be designed such that they implement correctly the required functionality. In addition, they have to fulfil a wide range of competing constraints: development cost, unit cost, reliability, security, size, performance, power consumption, flexibility, time-to-market, maintainability, correctness, safety, etc. Very important for the correct functioning of such systems are their timing constraints: 'the correctness of the system behaviour depends not only on the logical results of the computations, but also on the physical instant at which these results are produced' [1].

Real-time systems have been classified as *hard* real-time and *soft* real-time systems [1]. Basically, hard real-time systems are systems where failing to meet a timing constraint can potentially have catastrophic consequences. For example, a brake-by-wire system in a car failing to react within a given time interval can result in a fatal accident. On the other hand, a multimedia system, which is a soft real-time system, can, under certain circumstances, tolerate a certain amount of delays, resulting maybe in a patchier picture, without serious consequences besides some possible inconvenience to the user.

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 such systems, the communication between the functions implemented on different nodes has an important impact on the overall system properties such as performance, cost, maintainability, etc.

The analysis and optimisation approaches presented in this paper are aimed towards heterogeneous distributed hard real-time systems that implement safety-critical applications where timing constraints are of utmost importance to the correct behaviour of the application.

1.1 Automotive electronics

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

If we take the example of automotive manufacturers, they were reluctant, until recently, to use computer controlled functions onboard 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 less exhausts, all of these at lower costs. Then, from the manufacturers' side, there is a need for shorter time-to-market and reduced development and manufacturing costs. These, combined with the advancements of semiconductor technology, which is delivering ever increasing performance at lower and lower costs, has led to the rapid increase in the number of electronically controlled functions onboard a vehicle [2].

The amount of electronic content in an average car in 1977 had a cost of $110. In 2004, that cost was $1341, and it was expected that this figure would reach $1476 by the year 2005, continuing to increase because of the introduction of sophisticated electronics found until now only in high-end cars [3, 4]. It is estimated that in 2006 the electronics inside

IEE Proceedings online no. 20045069

doi: 10.1049/ip-cdt:20045069

Paper first received 5th July 2004 and in final revised form 8th February 2005

The authors are with the Computer and Information Science Department, Linköping University, SE-581 83, Linköping, Sweden

E-mail: paupo@ida.liu.se

a car will amount to 25% of the total cost of the vehicle (35% for the high-end models), a quarter of which will be due to semiconductors [3, 5]. High-end vehicles currently have up to 100 microprocessors implementing and controlling various parts of their functionality. The total market for semiconductors in vehicles is predicted to grow from $8.9 billions in 1998 to $21 billion in 2005, amounting to 10% of the total worldwide semiconductors market [2, 3].

At the same time, with the increased complexity, the type of functions implemented by embedded automotive electronics systems has also evolved. Thanks to the semiconductor revolution, in the late 1950s, electronic devices became small enough to be installed on board vehicles. In the 1960s the first analogue fuel injection system appeared, and in the 1970s analogue devices for controlling transmission, carburettor and spark advance timing were developed. The oil crisis of the 1970s 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 1980s, more sophisticated systems began to appear, like electronically controlled braking systems, dashboards, information and navigation systems, air conditioning systems, etc. In the 1990s, development and improvement have concentrated in areas like 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 [6, 7].

The complexity of electronics in modern vehicles is growing at a very high pace, and the constraints – in terms of functionality, performance, reliability, cost and time-to-market – are getting tighter. Therefore, the task of designing such systems is becoming increasingly important and difficult at the same time. New design techniques are needed, which are able to:

- successfully manage the complexity of embedded systems
- meet the constraints imposed by the application domain
- shorten the time-to-market
- reduce development and manufacturing costs.

The success of such new design methods depends on the availability of analysis and optimisation techniques, beyond those corresponding to the state-of-the-art, which are presented in the following Section.

2 Heterogeneous real-time embedded systems

2.1 Heterogeneous 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 Fig. 1). Each network has its own communication protocol, and inter-network 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.

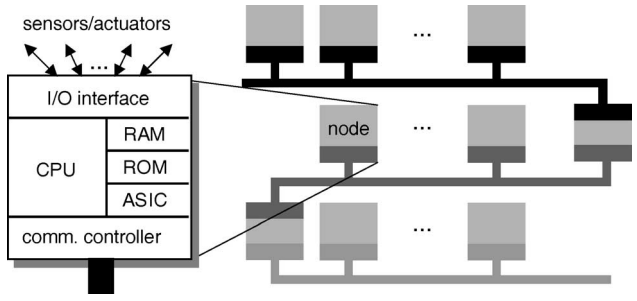

Fig. 1 *Distributed hard real-time systems*

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 area, 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, the mirrors, the windows and the dashboard. The constraints of the body electronic functions are determined by the reaction time of the human operator, which is in the range of 100 ms 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 like LIN. 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 to control actuators, and a simple network interface. Moreover, the memory size is growing by more than 25% each year [6, 8].
- *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 kbytes of RAM and 256 kbytes of ROM. These microcontrollers have built-in communication controllers (e.g. the 68HC11 and 68HC12 automotive family of microcontrollers have an on-chip CAN controller), I/O to sensors and actuators, and are interconnected by high bandwidth networks [6, 8].

Section 5 presents more details concerning the hardware and software architecture considered by our analysis and optimisation techniques.

2.2 Heterogeneous communication protocols

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

There are several communication protocols for real-time networks. Among the protocols that have been proposed for vehicle multiplexing, only the controller area network (CAN) [9], the local interconnection network (LIN) [10] and SAE's J1850 [11] are currently in use on a large scale. Moreover, only a few of them are suitable for safety-critical applications where predictability is mandatory [12]. A survey and comparison of communication protocols for safety-critical embedded systems is available in [12]. Communication activities can be triggered either dynamically, in response to an event, or statically, at predetermined moments in time.

• Therefore, on one hand, there are protocols that schedule the messages statically based on the progression of time, for example, the SAFEbus [13] and SPIDER [14] protocols for the avionics industry, and the TTCAN [15] and time-triggered protocol (TTP) [1] intended for the automotive industry.
• On the other hand, there are several communication protocols where message scheduling is performed dynamically, such as the controller area network (CAN) used in a large number of application areas including automotive electronics, LonWorks [16] and Profibus [17] 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.
• However, there is also a hybrid type of communication protocol, such as Byteflight [18] introduced by BMW for automotive applications and the FlexRay protocol [19], that allows the sharing of the bus by event-driven and time-driven messages.

The time-triggered protocols have the advantage of simplicity and predictability, while event-triggered protocols are flexible and have low cost. Moreover, protocols like TTP offer fault-tolerant services necessary in implementing safety-critical applications. However, it has been shown [20] that event-driven protocols like CAN are also predictable, and fault-tolerant services can also be offered on top of protocols like the TTCAN. A hybrid communication protocol like FlexRay offers some of the advantages of both worlds.

2.3 Heterogeneous scheduling policies

The automotive suppliers will select, based on their own requirements, the scheduling policy to be used in their ECU The main approaches to scheduling are:

• *Static cyclic scheduling* algorithms are used to build, off-line, a schedule table with activation times for each process, such that the timing constraints of processes are satisfied.
• *Fixed priority scheduling* (FPS). In this scheduling approach each process has a fixed (static) priority which is computed off-line. The decision about which ready process to activate is taken on-line according to their priority.
• *Earliest deadline first* (EDF). In this case, that process will be activated which has the nearest deadline.
Typically, processes scheduled off-line using static cyclic scheduling are non-pre-emptible, while processes scheduled using techniques such as FPS and EDF are pre-emptible. Another important distinction is between the event-triggered and time-triggered approaches.
• *Time-triggered*
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 synchronised to provide a global notion of time. Time-triggered systems are typically implemented using *non-pre-emptive static cyclic scheduling*, where the process activation or message communication is done based on a schedule table built off-line.
• *Event-triggered*
In the event-triggered approach activities happen when a significant change of state occurs. Event-triggered systems are typically implemented using pre-emptive fixed-priority based scheduling, or earliest deadline first, where, as a response to an event, the appropriate process is invoked to service it.

There has been a long debate in the real-time and embedded systems communities concerning the advantages of each approach [1, 21, 22]. Several aspects have been considered in favour of one or the other approach, such as flexibility, predictability, jitter control, processor utilisation and testability.

An interesting comparison of the ET and TT approaches, from a more industrial, in particular automotive perspective, can be found in [23]. The conclusion there is that one has to choose the right approach, depending on the particularities of the application.

For certain applications, 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.

2.4 Distributed safety-critical applications

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 is dedicated to the implementation of a single function or class of functions, allowing the system integrators to purchase nodes implementing required functions from different vendors, and to integrate them into their system [24]. There are several problems related to this restricted mapping of functionality:

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

This has created a huge pressure to reduce the number of nodes by integrating several functions in one node and, at the same time, to distribute certain functionality over several nodes (see Fig. 2). Although an application is typically distributed over one single network, we begin to see applications that are distributed across several networks. For example, in Fig. 2, the third application, represented as black dots, is distributed over two networks.

This trend is driven by the need to further reduce costs, improve resource usage, but also by application constraints

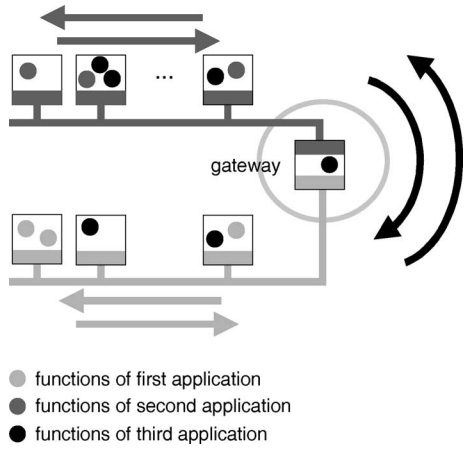

Fig. 2 *Distributed safety-critical applications*

like having to be physically close to particular sensors and actuators. Moreover, not only are these applications distributed across networks, but their functions can exchange critical information through the gateway nodes.

3 Schedulability analysis

There is a large quantity of research [1, 25, 26] related to scheduling and schedulability analysis, with results having been incorporated in analysis tools such as TimeWiz [27], RapidRMA [28], RTA-OSEK Planner [29] and Aires [30]. The tools determine if the timing constraints of the functionality are met, support the designer in exploring several design scenarios and help to design optimised implementations.

Typically, the timing analysis considers independent processes running on single processors. However, very often functionality consists of distributed processes that have data and control dependencies, exclusion constraints, etc. New schedulability analysis techniques are needed which can handle distributed applications, data and control dependencies, and accurately take into account the details of the communication protocols that have an important influence on the timing properties. Moreover, highly complex and safety critical applications can in the future be distributed across several networks, and can use different, heterogeneous, scheduling policies.

Pre-emptive scheduling of independent processes with static priorities running on single-processor architectures has its roots in the work of Liu and Layland [31]. The approach has been later extended to accommodate more general computational models and has also been applied to distributed systems [32]. The reader is referred to [25, 26, 33] for surveys on this topic. Static cyclic scheduling of a set of data dependent software processes on a multiprocessor architecture has also been intensively researched [1, 34].

In [35] an earlier deadline first strategy is used for non-pre-emptive scheduling of processes with possible data dependencies. Pre-emptive and non-pre-emptive static scheduling are combined in the cosynthesis environment described in [36, 37]. In many of the previous scheduling approaches researchers have assumed that processes are scheduled independently. However, processes can be sporadic or aperiodic, are seldom independent, and normally they exhibit precedence and exclusion constraints. Knowledge regarding these dependencies can be used in order to improve the accuracy of schedulability analyses and the quality of the produced schedules [38].

It has been claimed [22] that static cyclic scheduling is the only approach that can provide efficient solutions to applications that exhibit data dependencies. However, advances in the area of fixed priority pre-emptive scheduling show that such applications can also be handled with other scheduling strategies [39].

One way of dealing with data dependencies between processes in the context of static priority based scheduling has been indirectly addressed by the extensions proposed for the schedulability analysis of distributed systems through the use of the *release jitter* [32]. 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 [32, 40], time *offset* relationships and *phases*, respectively, are 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 show that by introducing such concepts into the computational model, the pessimism of the analysis is significantly reduced when bounding the time behaviour of the system. The concept of *dynamic offsets* has been later introduced and used to model data dependencies [41].

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

Researchers have often ignored or very much simplified the communication infrastructure. One typical approach is to consider communications as processes with a given execution time (depending on the amount of information exchanged) and to schedule them as any other process, without considering issues like communication protocol, bus arbitration, packaging of messages, clock synchronisation, etc. [40].

Tindell and Clark [32] integrate processor and communication scheduling and provide a 'holistic' schedulability analysis in the context of distributed real-time systems. The validity of the analysis has been later confirmed in [42].

In the case of a distributed system the response time of a process also depends on the communication delay due to messages. In [32] the analysis for messages is done in a similar way as for processes: a message is seen as a non-pre-emptible process that is 'running' on a bus. The response time analyses for processes and messages are combined by realising that the *jitter* (the delay between the *arrival* of a process – the time when it becomes ready for execution – and the start of its execution) of a destination process depends on the *communication delay* (the time it takes for a message to reach the destination process, from the moment it has been produced by the sender process) between sending and receiving a message. Several researchers have provided analyses that bound the communication delay for a given communication protocol:

- controller area network protocol [20]
- time-division multiple access protocol [32]
- asynchronous transfer mode protocol [43]
- token ring protocol [44]
- fibre distributed data interface protocol [45]
- time-triggered protocol [46]
- FlexRay protocol [47].

Based on their own requirements, the suppliers choose one particular scheduling policy to be used. However, for certain applications, several scheduling approaches can be used together.

One approach to the design of such systems is to allow ET and TT processes to share the same processor as well as static (TT) and dynamic (ET) communications to share the same bus. Bus sharing of TT and ET messages is supported by protocols which support both static and dynamic communication [19]. We have addressed the problem of timing analysis for such systems [47].

A fundamentally different architectural approach to heterogeneous TT/ET systems is that of heterogeneous multi-clusters, where each cluster can be either TT or ET. In a *time-triggered cluster*, processes and messages are scheduled according to a static cyclic policy, with the bus implementing a TDMA protocol such as, for example, the time-triggered protocol (introduced in Section 5.1). On *event-triggered clusters* the processes are scheduled according to a priority based pre-emptive approach, while messages are transmitted using the priority-based CAN bus.

In this context, we have proposed an approach to schedulability analysis for multi-cluster distributed embedded systems [48]. This analysis will be outlined in Section 7.

When several event-driven scheduling policies are used in a heterogeneous system, another approach to the verification of timing properties is to use the technique presented in [49] which couples the analysis of local scheduling strategies via an event interface model.

4 Design optimisation

4.1 Traditional design methodology

There are several methodologies for real-time embedded systems design. The aim of a design methodology is to co-ordinate the design tasks such that the time-to-market is minimised, the design constraints are satisfied, and various parameters are optimised.

The main design tasks that have to be performed are described in the following Sections.

4.1.1 Functional analysis and design: The functionality of the host system, into which the electronic system is embedded, is normally described using a formalism from that particular domain of application For example, if the host system is a vehicle, then its functionality is described in terms of control algorithms using differential equations, which are modelling the behaviour of the vehicle and its environment. At the level of the embedded real-time system which controls the host system, the functionality is typically described as a set of functions, accepting certain inputs and producing some output values.

The typical automotive application is a control application. The controller reads inputs from sensors, and uses the actuators to control the physical environment (the vehicle). A controller can have several modes of operation, and can interact with other electronic functions, or with the driver through switches and instruments.

During the functional analysis and design stage, the desired functionality is specified, analysed and decomposed into sub-functions based on the experience of the designer. Several suppliers and manufacturers have started to use tools like Statemate [50], Matlab/Simulink [51], ASCET/SD [52] and SystemBuild/MatrixX [53] for describing the functionality, to eliminate the ambiguities and to avoid producing incomplete or incoherent specifications.

At the level of functional analysis the exploration is currently limited to evaluating several alternative control algorithms for solving the control problem. Once the functionality has been captured using tools like Matlab/Simulink, useful explorations can involve simulations of executable specifications to determine the correctness of the behaviour, and to assess certain properties of chosen solutions.

4.1.2 Architecture selection and mapping: The architecture selection task decides what components to include in the hardware architecture and how these components are connected.

According to current practice, architecture selection is an ad hoc process, based on the experience of the designer and previous product versions.

The mapping task has to decide what part of the functionality should be implemented on which of the selected components.

The manufacturers integrate components from suppliers, and thus the design space is severely restricted in current practice, by the fact that the mapping of functionality to an ECU is fixed.

4.1.3 Software design and implementation: This is the phase in which the software is designed and the code is written.

The code for the functions is developed manually for efficiency reasons, and thus the exploration that would be allowed by automatic code generation is limited.

At this stage the correctness of the software is analysed through simulations, but there is no analysis of timing constraints, which is left for the scheduling and schedulability analysis stage.

4.1.4 Scheduling and schedulability analysis: Once the functions have been defined and the code has been written, the scheduling task is responsible for determining the execution strategy for the functions *inside an ECU*, such that the timing constraints are satisfied.

Simulation is extensively used to determine whether the timing constraints are satisfied. However, simulations are very time-consuming and provide no guarantees that the timing constraints are met.

In the context of static cyclic scheduling, deriving a schedule table is a complex design exploration problem. Static cyclic scheduling of a set of data dependent software processes on a multiprocessor architecture has been researched in [1, 34]. Such research has been used in commercial tools like TTP-Plan [54], which derives the static schedules for processes and messages in a time-triggered system using the time-triggered protocol for communication.

If fixed priority pre-emptive scheduling is used, exploration is used to determine how to allocate priorities to a set of distributed processes [55]. Their priority assignment heuristic is based on the schedulability analysis from [32]. For the earliest deadline, first the issue of distributing the global deadlines to local deadlines has to be addressed [56].

4.1.5 Integration: In this phase the manufacturer has to integrate the ECUs from different suppliers.

There is a lack of tools that can analyse the performance of the interacting functionality, and thus the manufacturer has to rely on simulation runs using the realistic environment of a prototype car. Detecting potential problems at such a late stage requires time-consuming extensive simulations. Moreover, once a problem is identified it takes a very long time to go through all the previous stages in order to fix it. This leads to large delays on the time-to-market.

To reduce the large simulation times, and to guarantee that potential violations of timing constraints are detected, manufacturers have started to use in-house analysis tools and commercially available tools such as Volcano Network Architect (for the CAN and LIN buses) [57].

Volcano makes inter-ECU communication transparent for the programmer. The programmer only deals with *signals* that have to be sent and received, and the details of the network are hidden. Volcano provides basic API calls for manipulating signals. To achieve interoperability between ECUs from different suppliers, Volcano uses a *publish/subscribe* model for defining the signal requirements. Published signals are made available to the system integrator by the suppliers, while subscribed signals are required as inputs to the ECU. The system integrator makes

the *publish/subscribe* connections by creating a set of CAN frames, and creating a mapping between the data in frames and signals [58]. Volcano uses the analysis in [20] for bounding the communication delay of messages transmitted using the CAN bus.

4.1.6 Calibration, testing and verification:

These are the final stages of the design process Because not enough analysis, testing and verification has been done in earlier stages of the design, these stages tend to be very time-consuming, and problems identified here lead to large delays in product delivery.

4.2 Function architecture co-design and platform based design

New design methodologies are needed, which can handle the increasing complexity of heterogeneous systems, and their competing requirements in terms of performance, reliability, low power consumption, cost, time-to-market, etc. As the complexity of the systems continues to increase, the development time lengthens dramatically, and the manufacturing costs become prohibitively high. To cope with this complexity, it is necessary to reuse as much as possible at all levels of the design process, and to work at higher and higher abstraction levels.

Function/architecture co-design is a design methodology proposed in [59, 60], which addresses the design process at higher abstraction levels. Function/architecture co-design uses a top-down synthesis approach, where trade-offs are evaluated at a high level of abstraction. The main characteristic of this methodology is the use, at the same time with the top-down synthesis, of a bottom-up evaluation of design alternatives, without the need to perform a full synthesis of the design. The approach to obtaining accurate evaluations is to use an accurate modeling of the behaviour and architecture, and to develop analysis techniques that are able to derive estimates and to formally verify properties relative to a certain design alternative. The determined estimates and properties, together with user-specified constraints, are then used to drive the synthesis process.

Thus, several architectures are evaluated to determine whether they are suited for the specified system functionality. There are two extremes in the degrees of freedom available for choosing an architecture. At one end, the architecture is already given, and no modifications are possible. At the other end of the spectrum, no constraints are imposed on the architecture selection, and the synthesis task has to determine, from scratch, the best architecture for the required functionality. These two situations are, however, not common in practice. Often, a *hardware platform* is available, which can be *parameterised* (e.g. size of memory, speed of the buses, etc.). In this case, the synthesis task is to derive the parameters of the platform architecture such that the functionality of the system is successfully implemented. Once an architecture is determined and/or parameterised, the function/architecture co-design continues with the mapping of functionality onto the instantiated architecture.

This methodology has been used in research tools like Polis [61] and Metropolis [62], and has also led to commercial tools such as the Virtual Component Co-design (VCC) [63].

To reduce costs, especially in the case of a mass market product, the system architecture is usually reused, with some modifications, for several product lines. Such a common architecture is denoted by the term *platform*, and consequently the design tasks related to such an approach are grouped under the term *platform-based design* [64]. The 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 the hope of reducing costs and the time-to-market.

The authors in [64] have proposed techniques for deriving such a platform for a given family of applications. Their approach can be used within any design methodology for determining a system platform that later on can be parameterised and instantiated to a desired system architecture.

Considering a given application or family of applications, the system platform has to be instantiated, deciding on certain parameters, and lower level details, to suit that particular application(s). The search for an architecture instance starts from a certain platform and a given application. The application is mapped and compiled on an architecture instance, and the performance numbers are derived, typically using simulation. If the designer is not satisfied with the performance of the instantiated architecture, the process is repeated.

In the remainder of the paper we will consider a platform consisting of event- and time-triggered clusters, using the CAN and TTP protocols for communication, respectively. We will discuss analysis and optimisation techniques for the configuration of the platform such that the given application is schedulable.

5 Multi-cluster systems

One class of heterogeneous real-time embedded systems is that of *multi-cluster* systems. We consider architectures consisting of two clusters – one time-triggered and the other event-triggered – interconnected by gateways (see Fig. 2):

- In a *time-triggered cluster* (TTC) processes and messages are scheduled according to a static cyclic policy, with the bus implementing a TDMA protocol such as, for example, the time-triggered protocol (TTP) [65].
- On *event-triggered clusters* (ETC) the processes are scheduled according to a priority based pre-emptive approach, while messages are transmitted using the priority-based CAN bus [9].

Sections 5.1 and 5.2 present the hardware and software architecture of a two-cluster system, respectively while Section 5.3 presents the application model used. Section 6 will introduce design problems characteristic for multi-cluster systems composed of time-triggered clusters interconnected with event-triggered clusters: the partitioning of functionality between the TT and ET clusters, the mapping of functionality to the nodes inside a cluster, and the packing of application message to frames on the TTP and CAN buses. Then, Section 8 will present two optimisation strategies for the frame packing problem.

5.1 Hardware architecture

A cluster is composed of nodes which share a broadcast communication channel Let $\mathcal{N}_T$ ($\mathcal{N}_E$) be the set of nodes on the TTC (ETC). Every node $N_i \in \mathcal{N}_T \cup \mathcal{N}_E$ includes a communication controller and a CPU, along with other components. The gateways, connected to both types of clusters, have two communication controllers, for TTP and CAN. The communication controllers implement the protocol services, and run independently of the node's CPU. Communication with the CPU is performed through a message base interface (MBI).

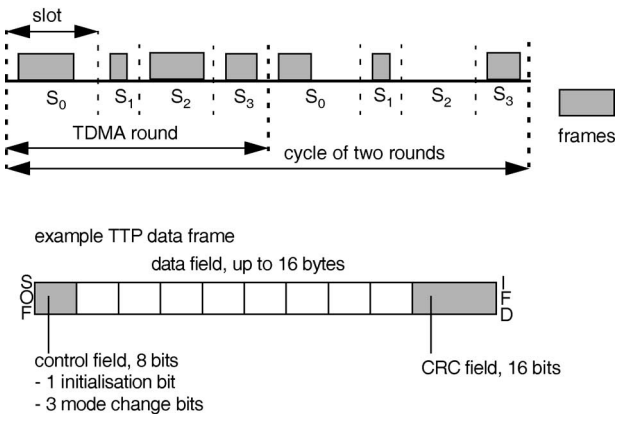

Fig. 3 *Time-triggered protocol*

Communication between the nodes on a TTC is based on the TTP [65]. The TTP integrates all the services necessary for fault-tolerant real-time systems. The bus access scheme is time-division multiple-access (TDMA), meaning that each node N_i on the TTC, including the gateway node, can transmit only during a predetermined time interval, the TDMA slot S_i. In such a slot, a node can send several messages packed in a frame. A sequence of slots corresponding to all the nodes in the architecture is called a TDMA round. A node can have only one slot in a TDMA round. Several TDMA rounds can be combined together in a cycle that is repeated periodically. The sequence and length of the slots are the same for all the TDMA rounds. However, the length and contents of the frames may differ.

The TDMA access scheme is imposed by a message descriptor list (MEDL) that is located in every TTP controller. The MEDL serves as a schedule table for the TTP controller which has to know when to send/receive a frame to/from the communication channel.

There are two types of frames in the TTP. The initialisation frames, or I-frames, which are needed for the initialisation 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 (Fig. 3) consists of the following fields: start of frame bit (SOF), control field, a data field of up to 16 bytes containing one or more messages, and a cyclic redundancy check (CRC) field. Frames are delimited by the inter-frame delimiter (IDF, 3 bits).

For example, the data efficiency of a frame that carries 8 bytes of application data, i.e. the percentage of transmitted bits which are the actual data bits needed by the application, is 69.5% (64 data bits transmitted in a 92-bit frame, without considering the details of a particular physical layer). Note that no identifier bits are necessary, as the TTP controllers know from their MEDL what frame to expect at a given point in time. In general, the protocol efficiency is in the range of 60–80% [66].

On an ETC, the CAN [9] protocol is used for communication. The CAN bus is a priority bus that employs a collision avoidance mechanism, whereby the node that transmits the frame with the highest priority wins the contention. Frame priorities are unique and are encoded in the frame identifiers, which are the first bits to be transmitted on the bus.

In the case of CAN 2.0A (Fig. 4), there are four frame types: data frame, remote frame, error frame and overload frame. We are interested in the composition of the data frame, depicted in Fig. 3. A data frame contains seven fields: SOF, arbitration field that encodes the 11-bit frame identifier, a control field, a data field up to 8 bytes, a CRC field, an acknowledgment (ACK) field, and an end of frame field (EOF).

In this case, for a frame that carries 8 bytes of application data, we will have an efficiency of 47.4% [67]. The typical CAN protocol efficiency is in the range of 25–35% [66].

5.2 Software architecture

A real-time kernel is responsible for activation of processes and transmission of messages on each node. On a TTC, the processes are activated based on the local schedule tables, and messages are transmitted according to the MEDL. On an ETC, we have a scheduler that decides on activation of ready processes and transmission of messages, based on their priorities.

In Fig. 5 we illustrate our message passing mechanism. Here we concentrate on the communication between processes located on different clusters. For message passing within a TTC the reader is directed to [68], while the infrastructure needed for communications on an ETC has been detailed in [20].

Let us consider the example in Fig. 5, where we have an application consisting of four processes and four messages (depicted in Fig. 5*b*) mapped on the two clusters in Fig. 5*c*. Processes P_1 and P_4 are mapped on node N_1 of the TTC, while P_2 and P_3 are mapped on node N_2 of the ETC. Process P_1 sends messages m_1 and m_2 to processes P_2 and P_3, respectively, while P_2 and P_3 send messages m_3 and m_4 to P_4. All messages have a size of 1 byte.

The transmission of messages from the TTC to the ETC takes place in the following way (see Fig. 5). P_1, which is statically scheduled, is activated according to the schedule table, and when it finishes it calls the send kernel function in order to send m_1 and m_2, indicated in the Figure by the number (1). Messages m_1 and m_2 have to be sent from node N_1 to node N_2. At a certain time, known from the schedule table, the kernel transfers m_1 and m_2 to the TTP controller by packing them into a frame in the MBI. Later on, the TTP controller knows from its MEDL when it has to take the frame from the MBI to broadcast it on the bus. In our example, the timing information in the schedule table of the kernel and the MEDL is determined in such a way that the broadcasting of the frame is done in the slot S_1 of round 2 (2). The TTP controller of the gateway node N_G

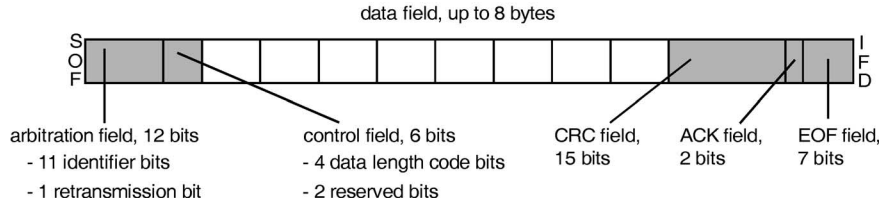

Fig. 4 *Controller area network data frame (CAN 2.0A)*

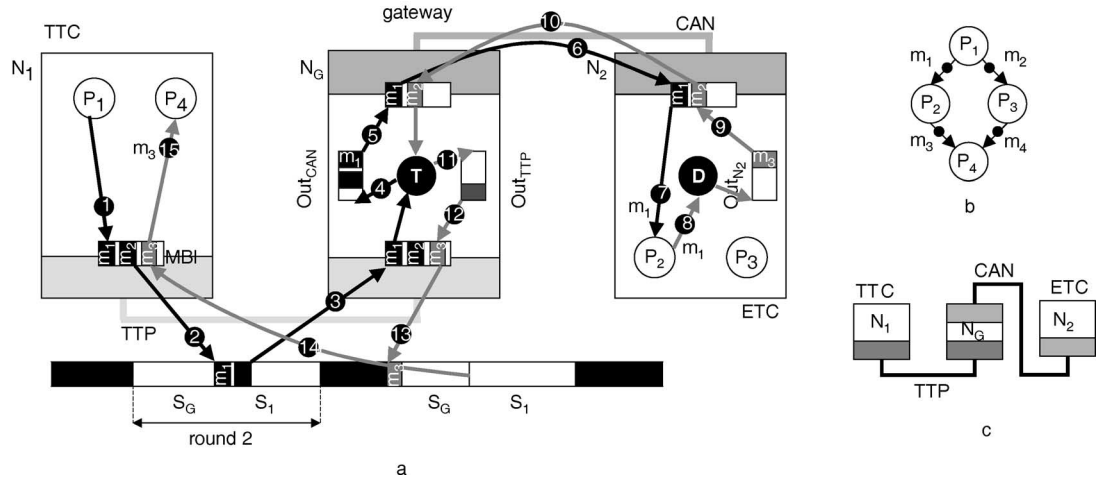

Fig. 5 *Message passing example*

knows from its MEDL that it has to read a frame from slot S_1 of round 2 and to transfer it into its MBI (3). Invoked periodically, having the highest priority on node N_G, and with a period which guarantees that no messages are lost, the gateway process T copies messages m_1 and m_2 from the MBI to the TTP-to-CAN priority-ordered message queue Out_{CAN} (4). Let us assume that on the ETC messages m_1 and m_2 are sent independently, one per frame. The highest priority frame in the queue, in our case the frame f_1 containing m_1, will tentatively be broadcast on the CAN bus (5). Whenever f_1 is the highest priority frame on the CAN bus, it will successfully be broadcast and will be received by the interested nodes, in our case node N_2 (6). The CAN communication controller of node N_2 receiving f_1 will copy it in the transfer buffer between the controller and the CPU, and raise an interrupt which will activate a delivery process, responsible to activate the corresponding receiving process, in our case P_2, and hand over message m_1 that finally arrives at the destination (7).

Message m_3 (depicted in Fig. 5 as a grey rectangle labelled 'm_3'), sent by process P_2 from the ETC, will be transmitted to process P_4 on the TTC. The transmission starts when P_2 calls its send function and enqueues m_3 in the priority-ordered Out_{N_2} queue (8). When the frame f_3 containing m_3 has the highest priority on the bus, it will be removed from the queue (9) and broadcast on the CAN bus (10). Several messages can be packed into a frame in order to increase the efficiency of data transmission. For example, m_3 can wait in the queue until m_4 is produced by P_3, in order to be packed together with m_4 in a frame. When f_3 arrives at the gateway's CAN controller it raises an interrupt. Based on this interrupt, the gateway transfer process T is activated, and m_3 is unpacked from f_3 and placed in the Out_{TTP} FIFO queue (11). The gateway node N_G is only able to broadcast on the TTC in the slot S_G of the TDMA rounds circulating on the TTP bus. According to the MEDL of the gateway, a set of messages not exceeding $size_{S_G}$ of the data field of the frame travelling in slot S_G will be removed from the front of the Out_{TTP} queue in every round, and packed in the S_G slot (12). Once the frame is broadcast (13) it will arrive at node N_1 (14), where all the messages in the frame will be copied in the input buffers of the destination processes (15). Process P_4 is activated according to the schedule table, which has to be constructed such that it accounts for the worst-case communication delay of message m_3, bounded by the analysis in Section 7.1, and thus, when P_4 starts executing, it will find m_3 in its input buffer.

As part of our frame packing approach, we generate all the MEDLs on the TTC (i.e. the TT frames and the sequence of the TDMA slots), as well as the ET frames and their priorities on the ETC such that the global system is schedulable.

5.3 Application model

We model an application Γ as a set of process graphs $G_i \in \Gamma$ (see Fig. 6). Nodes in the graph represent processes and arcs represent dependency between the connected processes. A process is a sequence of computations (corresponding to several building blocks in a programming language) which starts when all its inputs are available. When it finishes executing, the process produces its output values. Processes can be pre-emptible or non-pre-emptible. Non-pre-emptible processes are processes that cannot be interrupted during their execution, and are mapped on the TTC. Pre-emptible processes can be interrupted during their execution, and are mapped on the ETC. For example, a higher priority process has to be activated to service an event, in this case, the lower priority process will be temporary pre-empted until the higher priority process finishes its execution.

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

The communication time between processes mapped on the same processor is considered to be part of the process worst-case execution time and is not modelled explicitly. Communication between processes mapped to different processors is performed by message passing over the buses and, if needed, through the gateway. Such message passing is modelled as a communication process inserted on the arc connecting the sender and the receiver process (the black dots in Fig. 6).

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

Each process P_i is mapped on a processor $\mathcal{M}(P_i)$ (mapping represented by hashing in Fig. 6), and has

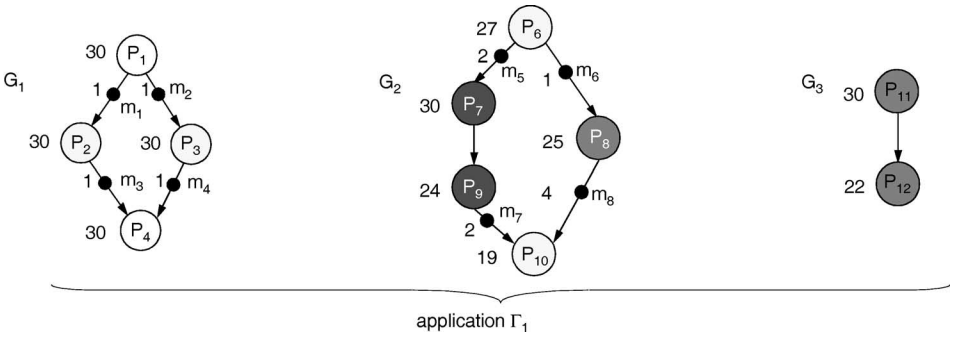

Fig. 6 *Application model*

a worst case execution time C_i on that processor (depicted to the left of each node). The designer can provide manually such worst-case times, or tools can be used in order to determine the worst-case execution time of a piece of code on a given processor [69].

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 a uniquely assigned priority – $priority_{P_i}$ for processes and $priority_{m_i}$ for messages.

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

6 Multi-cluster optimisation

Considering the type of applications and systems described in the preceding Section, and using the analysis outlined in Section 7, several design optimisation problems can be addressed.

In this Section, we present problems which are characteristic to applications distributed across multi-cluster systems consisting of heterogeneous TT and ET networks:

- Section 6.1 briefly outlines the problem of partitioning the processes of an application into time-triggered and event-triggered domains, and their mapping to the nodes of the clusters.
- Section 6.2 presents the problem of packing of messages to frames, which is of utmost importance in cost-sensitive embedded systems where resources, such as communication bandwidth, have to be fully utilised [58, 70, 71]. This problem will be discussed in more detail in Section 8.

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

To drive our optimisation algorithms towards schedulable solutions, we characterise a given frame packing configuration using the degree of schedulability of the application. The degree of schedulability [72] is calculated as

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

where n is the number of processes in the application, r_i is the worst-case response time of a process P_i and D_i its deadline. The worst-case response times are calculated by the MultiClusterScheduling algorithm using the response time analysis presented in Section 7.

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 optimisation parameters leading to a schedulable process set, a smaller c_2 means that we have improved the worst-case response times of the processes, so the application can potentially be implemented on a cheaper hardware architecture (with slower processors and/or buses). Improving the degree of schedulability can also lead to an improvement in the quality of control for control applications.

6.1 Partitioning and mapping

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

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

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

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

ET processes. The set $\mathcal{P}^* = \mathcal{P}_T^* \cup \mathcal{P}_E^* \cup \mathcal{P}^+$ then represents all the unmapped processes in the application.

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

Using the notation introduced, the partitioning and mapping problem can be described more exactly as follows. As an input we have an application Γ given as a set of process graphs and a two-cluster system consisting of a TT and an ET cluster. As introduced previously, $\mathcal{P}_T$ and $\mathcal{P}_E$ are the sets of processes already partitioned into TT and ET, respectively. Also, $\mathcal{P}_T^M \subseteq \mathcal{P}_T$ and $\mathcal{P}_E^M \subseteq \mathcal{P}_E$ are the sets of already mapped TT and ET processes. We are interested to find a partitioning for processes in $\mathcal{P}^+ = \mathcal{P} \backslash (\mathcal{P}_T \cup \mathcal{P}_E)$ and decide a mapping for processes in $\mathcal{P}^* = \mathcal{P}_T^* \cup \mathcal{P}_E^* \cup \mathcal{P}^+$, where $\mathcal{P}_T^* = \mathcal{P}_T \backslash \mathcal{P}_T^M$, and $\mathcal{P}_E^* = \mathcal{P}_E \backslash \mathcal{P}_E^M$ such that imposed deadlines are guaranteed to be satisfied.

6.2 Frame packing

In both the TTP and CAN protocols messages are not sent independently, but several messages having similar timing properties are usually packed into frames In many application areas, like automotive electronics, messages range from one single bit (e.g. the state of a device) to a couple of bytes (e.g. vehicle speed, etc.). Transmitting such small messages one per frame would create a high communication overhead, which can cause long delays leading to an unschedulable system. For example, 65 bits have to be transmitted on CAN for delivering one single bit of application data. Moreover, a given frame configuration defines the exact behaviour of a node on the network, which is very important when integrating nodes from different suppliers.

Let us consider the motivational example in Fig. 7, where we have the process graph from Fig. 7*d* mapped on the two-cluster system from Fig. 7*e*: P_1 and P_4 are mapped on node N_1 from the TTC, while P_2 and P_3 are mapped on N_2 from ETC. The data field of the frames is represented with a black rectangle, while the other frame fields are depicted with a grey colour. We consider a physical implementation of the buses such that the frames will take the time indicated in the Figure by the length of their rectangles. We are interested to find a frame configuration such that the application is schedulable.

In the system configuration of Fig. 7*a* we consider that, on the TTP bus, the node N_1 transmits in the first slot (S_1) of the TDMA round, while the gateway transmits in the second slot (S_G). Process P_3 has a higher priority than process P_2, hence P_2 will be interrupted by P_3 when it receives message m_2. In such a setting, P_4 will miss its deadline, which is depicted as a thick vertical line in Fig. 7. Changing the frame configuration as in Fig. 7*b*, so that messages m_1 and m_2 are packed into frame f_1 and slot S_G of the gateway comes first, processes P_2 and P_3 will receive m_1 and m_2 sooner and thus reduce the worst-case response time of the process graph, which is still larger than the deadline. In Fig. 7*c*, we also pack m_3 and m_4 into f_2. In such a situation, the sending of m_3 will have to be delayed until m_4 is queued by P_2. Nevertheless, the worst-case response time of the application is further reduced, which means that the deadline is met, and thus the system is schedulable.

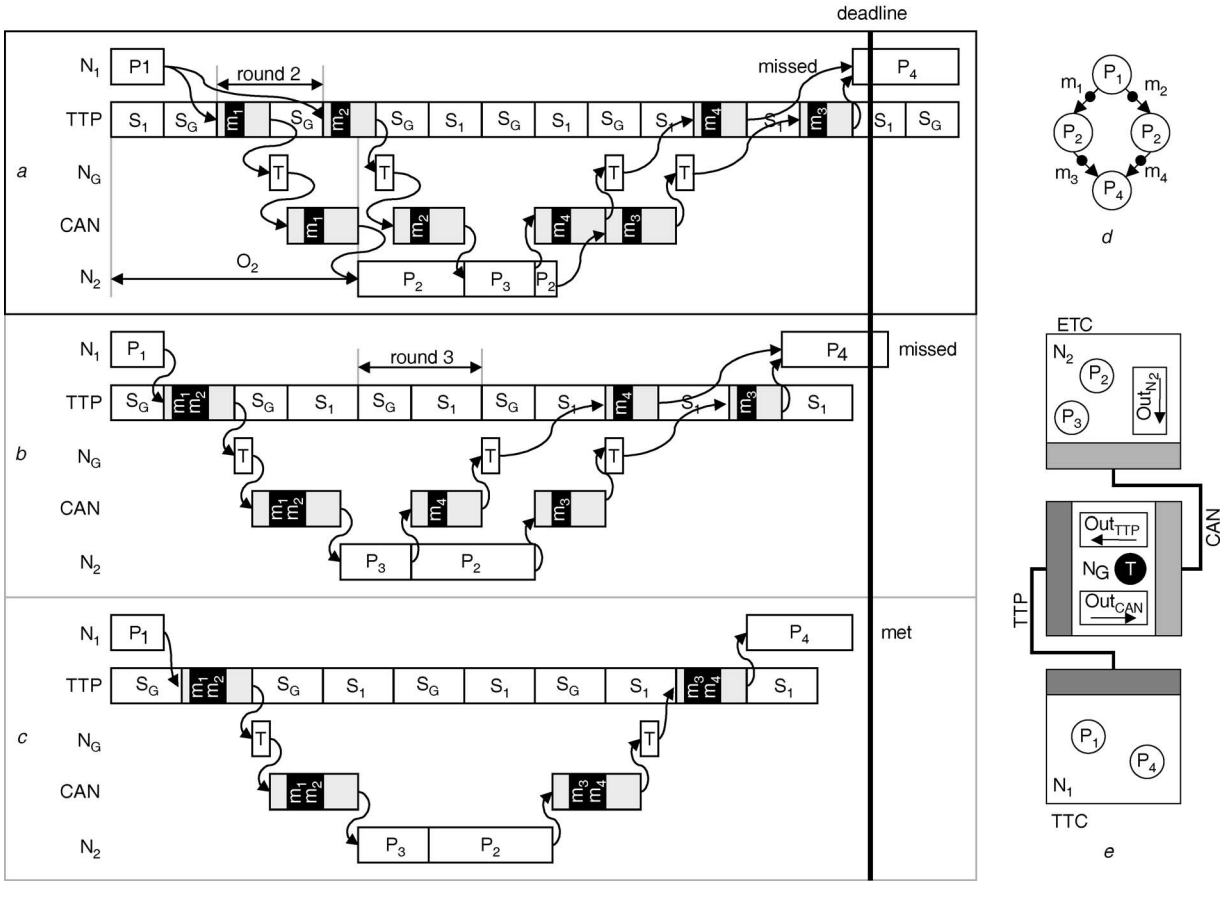

Fig. 7 *Frame-packing optimisation example*

However, packing more messages will not necessarily reduce the worst-case response times further, as it might increase too much the worst-case response times of messages that have to wait for the frame to be assembled, as is the case with message m_3 in Fig. 7*c*.

This design optimisation problem can be formulated more exactly as follows. As input to the frame-packing problem we have an application Γ given as a set of process graphs mapped on an architecture consisting of a TTC and an ETC interconnected through a gateway. We consider that the partitioning and mapping of processes has been already decided.

We are interested to find a mapping of messages to frames (a frame packing configuration) denoted by a 4-tuple $\psi = <\alpha, \pi, \beta, \sigma>$ such that the application Γ is schedulable. Once a schedulable system is found, we are interested to further improve the 'degree of schedulability' so the application can potentially be implemented on a cheaper hardware architecture (with slower buses and processors).

Determining a frame configuration ψ means deciding on:

- the mapping of application messages transmitted on the ETC to frames (the set of ETC frames α), and their relative priorities, π. Note that the ETC frames α have to include messages transmitted from an ETC node to a TTC node, messages transmitted inside the ETC cluster, and those messages transmitted from the TTC to the ETC
- the mapping of messages transmitted on the TTC to frames, denoted by the set of TTC frames β, and the sequence σ of slots in a TDMA round. The slot sizes are determined based on the set β, and are calculated such that they can accommodate the largest frame sent in that particular slot. We consider that messages transmitted from the ETC to the TTC are not statically allocated to frames. Rather, we will dynamically pack messages originating from the ETC into the 'gateway frame', for which we have to decide the data field length (see Section 5.2).

Several details related to the schedulability analysis were omitted from the discussion of the example. These details will be discussed in Section 7.

7 Multi-cluster analysis and scheduling

Once a partitioning and a mapping is decided, and a frame packing configuration is fixed, the processes and messages have to be scheduled. For the TTC this means building the schedule tables, while for the ETC the priorities of the ET processes have to be determined and their schedulability has to be analysed.

The analysis presented in this Section works under the following assumptions:

- All the processes belonging to a process graph G have the same period T_G. However, process graphs can have different periods.
- The offsets are *static* (as opposed to *dynamic* [42]), and are smaller than the period.
- The deadlines are arbitrary, i.e. they can be larger than the period.

The basic idea is that on the TTC an application is schedulable if it is possible to build a schedule table such that the timing requirements are satisfied.

On the ETC, the answer to whether or not a system is schedulable is given by a schedulability analysis. In this paper, for the ETC we use a response time analysis, where the schedulability test consists of the comparison between the worst-case response time r_i of a process P_i and its deadline D_i. Response time analysis of data dependent processes with static priority pre-emptive scheduling has been proposed in [39, 40, 42] and has also been extended to consider the CAN protocol [20]. The authors use the concept of *offset* to handle data dependencies. Thus, each process P_i is characterised by an offset O_i, measured from the start of the process graph, that indicates the earliest possible start time of P_i. Such an offset is, for example, O_2 in Fig. 7*a*, as process P_2 cannot start before receiving m_1. The same is true for messages, their offset indicating the earliest possible transmission time. The response time analysis employed is presented in Section 7.1.

However, determining the schedulability of an application mapped on a multi-cluster system cannot be addressed separately for each type of cluster, since the inter-cluster communication creates a circular dependency: the static schedules determined for the TTC influence through the offsets the worst-case response times of the processes on the ETC, which in their turn influence the schedule table construction on the TTC. In Fig. 7*b*, packing m_1 and m_2 in the same frame leads to equal offsets for P_2 and P_3. Because of this, P_3 will delay P_2 (which would not be the case if m_2 sent to P_3 were scheduled in round 3, for example) and thus the placement of P_4 in the schedule table has to be accordingly delayed to guarantee the arrivals of m_3 and m_4.

In our analysis we consider the influence between the two clusters by making the following observations:

- The start time of process P_i in a schedule table on the TTC is its offset O_i.
- The worst-case response time r_i of a TT process is its worst case execution time, i.e. $r_i = C_i$ (TT processes are not pre-emptible).
- The worst-case response times of the messages exchanged between two clusters have to be calculated according to the schedulability analysis described in Section 7.1.
- The offsets have to be set by a scheduling algorithm such that the precedence relationships are preserved. This means that, if process P_B depends on process P_A, the following condition must hold: $O_B \geq O_A + r_A$. Note that for the

MultiClusterScheduling$(\Gamma, \mathcal{M}, \psi)$
– determines the set of offsets ϕ and worst-case response times ρ
1 **for each** $O_i \in \phi$ **do** $O_i = 0$ **end for** – initially all offsets are zero
2 – determine initial values for the worst-case response times
3 – according to the analysis in Section 7.1
4 $\rho = \mathsf{ResponseTimeAnalysis}(\Gamma, \mathcal{M}, \psi, \phi)$
5 – determine new values for the offsets, based on the response times ρ
6 $\phi^{new} = \mathsf{ListScheduling}(\Gamma, \mathcal{M}, \psi, \rho)$
7 $\delta_\Gamma = \infty$ – consider the system unschedulable at first
8 **repeat** – iteratively improve the degree of schedulability δ_Γ
9 **for each** $O_i^{new} \in \phi^{new}$ **do** – for each newly calculated offset
10 $O_i^{old} = \phi.O_i, \phi.O_i = \phi^{new}.O_i^{new}$ – set the new offset, remember old
11 $\rho^{new} = \mathsf{ResponseTimeAnalysis}(\Gamma, \mathcal{M}, \psi, \phi)$
12 $\delta_\Gamma^{new} = \mathsf{SchedulabilityDegree}(\Gamma, \rho)$
13 **if** $\delta_\Gamma^{new} < \delta_\Gamma$ **then** – the schedulability has improved
14 – offsets are recalculated using ρ^{new}
15 $\phi^{new} = \mathsf{ListScheduling}(\Gamma, \mathcal{M}, \psi, \rho^{new})$
16 **break** – exit the for-each loop
17 **else** – the schedulability has not improved
18 $\phi.O_i = O_i^{old}$ – restore the old offset
19 **end for**
20 **until** δ_Γ has not changed
21 **return** $\rho, \phi, \delta_\Gamma$
end MultiClusterScheduling

Fig. 8 *MultiClusterScheduling algorithm*

processes on a TTC which receive messages from the ETC this translates to setting the start times of the processes such that a process is not activated before the worst-case arrival time of the message from the ETC. In general, offsets on the TTC are set such that all the necessary messages are present at the process invocation.

The MultiClusterScheduling algorithm in Fig. 8 receives as input the application Γ, the frame configuration ψ, and produces the offsets ϕ and worst-case response times ρ.

The algorithm sets initially all the offsets to 0 (line 1). Then, the worst-case response times are calculated using the ResponseTimeAnalysis function (line 4) using the analysis presented in Section 7.1. The fixed-point iterations that calculate the response times at line 3 will converge if processor and bus loads are smaller than 100% [39]. Based on these worst-case response times, we determine new values ϕ^{new} for the offsets using a list scheduling algorithm (line 6). We now have a schedule table for the TTC and worst-case response times for the ETC, which are pessimistic. The following loop will reduce the pessimism of the worst-case response times.

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

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

In the multi-cluster scheduling algorithm, the calculation of offsets is performed by the list scheduling algorithm presented in Fig. 9. In each iteration, the algorithm visits the processes and messages in the ReadyList. A process or a message in the application is placed in the ReadyList if all its predecessors have already been scheduled. The list is ordered based on the priorities presented in [73]. The algorithm terminates when all processes and messages have been visited.

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

1. The visited node is an ET message. The message m_i is packed into its frame f (line 9), and the offset O_f of the frame is updated. The frame can only be transmitted after all the sender processes that pack messages in this frame have finished executing. The offset of message m_i packed to frame f is equal to the frame offset O_f.
2. The node is a TT message. In this case, when the frame is ready for transmission, it is scheduled using the ScheduleTTFrame function (presented in Fig. 10), which returns the *round* and the *slot* where the frame has been placed (line 16 in Fig. 9). In Fig. 10, the round immediately following *offset* is the initial candidate to be considered (line 2). However, it can be too late to catch the allocated slot, in which case the next round is considered (line 4). For this candidate round, we have to check if the slot is not occupied by another frame. If so, the communication has to be delayed for another round (line 7). Once a frame has been scheduled, we can determine the offsets and worst-case response times (Fig. 9, line 18). For all the messages in the frame the offset is equal to the start of the slot in the TDMA round, and the worst-case response time is the slot length.
3. The algorithm visits a process P_i mapped on an ETC node. A process on the ETC can start as soon as its predecessors have finished and its inputs have arrived, hence $O_i = offset$ (line 22). However, P_i might experience, later on, interference from higher priority processes.
4. Process P_i is mapped on a TTC node. In this case, besides waiting for the predecessors to finish executing, P_i will also have to wait for its processor $\mathcal{M}(P_i)$ to become available (line 25). The earliest time when the processor is available is returned by the ProcessorAvailable function.

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

Fig. 9 *ListScheduling algorithm*

```
ScheduleTTFrame ( f, offset, ψ)
– returns the slot and the round assigned to frame f
1 slot = the slot assigned to the node sending f – the frame slot
2 round = offset / T_TDMA – the first round which could be a
    candidate
3 if offset – round * T_TDMA > O_slot then – the slot is missed
4   round = round + 1 – if yes, take the next round
5 end if
6 while slot is occupied do
7   round = round + 1
8 end while
9 return round, slot
end ScheduleTTFrame
```

Fig. 10 *Frame scheduling on TTC*

Let us now turn the attention back to the multi-cluster scheduling algorithm in Fig. 8. The algorithm stops when the δ_Γ of the application Γ is no longer improved, or when a limit imposed on the number of iterations has been reached. Since in a loop iteration we do not accept a solution with a larger δ_Γ, the algorithm will terminate when in a loop iteration we are no longer able to improve δ_Γ by modifying the offsets.

7.1 Schedulability analysis for ETC

For the ETC we use a response time analysis. A response time analysis has two steps. In the first step, the analysis derives the worst-case response time of each process (the time it takes from the moment it is ready for execution until it has finished executing). The second step compares the worst-case response time of each process to its deadline and, if the response times are smaller than or equal to the deadlines, the system is schedulable. The analysis presented in this Section is used in the ResponseTimeAnalysis function (line 4 of the algorithm in Fig. 8).

Thus, the response time analysis in [74] uses the following equation for determining the worst-case response time r_i of a process P_i:

$$r_i = C_i + \sum_{\forall P_j \in hp(P_i)} \left\lceil \frac{r_i}{T_j} \right\rceil C_j \quad (2)$$

where C_i is the worst-case execution time of process P_i, T_j is the period of process P_j and $hp(P_i)$ denotes the set of processes that have a priority higher than the priority of P_i.

The summation term, representing the interference I_i of higher priority processes on P_i, increases monotonically in r_i, and thus solutions can be found using a recurrence relation. Moreover, the recurrence relations that calculate the worst-case response time are guaranteed to converge if the processor utilisation is under 100%.

The previously presented analysis assumes that the deadline of a process is smaller than or equal to its period. This assumption has later been relaxed [32] to consider arbitrary deadlines (i.e. deadlines can be larger than the period). Thus, the worst-case response time r_i of a process P_i becomes

$$r_i = \max_{q=0,1,2\ldots} (J_i + w_i(q) - qT_i) \quad (3)$$

where J_i is the jitter of process P_i (the worst-case delay between the arrival of a process and the start of its execution), q is the number of busy periods being examined, and $w_i(q)$ is the width of the level-i busy period starting at time qT_i. The level-i busy period is defined as the maximum time a processor executes processes of priority greater than or equal to the priority of process P_i, and is calculated as [32]

$$w_i(q) = (q+1)C_i + B_i + \sum_{\forall P_j \in hp(P_i)} \left\lceil \frac{w_i(q) + J_j}{T_j} \right\rceil C_j \quad (4)$$

The pessimism of the previous analysis can be reduced by using the information related to the precedence relations between processes. The basic idea is to exclude certain worst-case scenarios, from the critical instant analysis, which are impossible due to precedence constraints.

Methods for schedulability analysis of data dependent processes with static priority pre-emptive scheduling have been proposed in [39–42]. They use the concept of *offset* (or *phase*) to handle data dependencies. In [39], Tindell shows that the pessimism of the analysis is reduced through the introduction of offsets. The offsets have to be determined by the designer.

In their analysis [39], the response time of a process P_i is

$$r_i = \max_{q=0,1,2\ldots} \left(\max_{\forall P_j \in G} \left(w_i(q) + O_j + J_j - T_G \times \left(q + \left\lceil \frac{O_j + J_j - O_i - J_i}{T_G} \right\rceil \right) - O_i \right) \right) \quad (5)$$

where T_G is the period of the process graph G, O_i and O_j are offsets of processes P_i and P_j, respectively, and J_i and J_j are the release jitters of P_i and P_j. In (5), the level-i busy period starting at time qT_G is

$$w_i(q) = (q+1)C_i + B_i + I_i \quad (6)$$

In the previous equation, the blocking term B_i represents interference from lower priority processes that are in their critical section and cannot be interrupted, and C_i represents the worst-case execution time of process P_i. The last term captures the interference I_i from higher priority processes in the application, including higher priority processes from other process graphs. The reader is directed to [39] for the details of the interference calculation.

Although this analysis is exact (both necessary and sufficient), it is computationally infeasible to evaluate. Hence, [39] proposes a feasible but not exact analysis (sufficient but not necessary) for solving (5). Our implementations use the feasible analysis provided in [39] for deriving the worst-case response time of a process P_i.

We are now interested to determine the worst-case response time of frames and the worst-case queueing delays experienced by a frame in a communication controller.

Regarding the worst-case response time of messages, we have extended the analysis from [20] and applied it for frames on the CAN bus:

$$r_f = \max_{q=0,1,2\ldots} (J_f + W_f(q) + (1+q)C_f) \quad (7)$$

In the previous equation, J_f is the jitter of frame f which in the worst case is equal to the largest worst-case response time $r_{S(m)}$ of a sender process $S(m)$ which sends message m packed into frame f:

$$J_f = \max_{\forall m \in f} (r_{S(m)}) \quad (8)$$

In (7), W_f is the worst-case queueing delay experienced by f at the communication controller, and is calculated as

$$W_f(q) = w_f(q) - qT_f \quad (9)$$

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

Moreover, in (7), C_f is the worst-case time it takes for a frame f to reach the destination controller. On CAN, C_f depends on the frame configuration and the size of the data field, s_f, while on TTP it is equal to the slot size in which f is transmitted.

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

The worst-case queueing delay for a frame (W_f in equation (7)) is calculated differently for each type of queue:

1. The output queue of an ETC node, in which case $W_f^{N_i}$ represents the worst-case time a frame f has to spend in the Out_{N_i} queue on ETC node N_i. An example of such a frame is the one containing message m_3 in Fig. 7*a*, which is sent by process P_2 from the ETC node N_2 to the gateway node N_G, and has to wait in the Out_{N_2} queue.
2. The TTP-to-CAN queue of the gateway node, in which case W_f^{CAN} is the worst-case time a frame f has to spend in the Out_{CAN} queue of node N_G. In Fig. 7*a*, the frame containing m_1 is sent from the TTC node N_1 to the ETC node N_2, and has to wait in the Out_{CAN} queue of gateway node N_G before it is transmitted on the CAN bus.
3. The CAN-to-TTP queue of the gateway node, where W_f^{TTP} captures the time f has to spend in the Out_{TTP} queue node N_G. Such a situation is present in Fig. 7*a*, where the

frame with m_3 is sent from the ETC node N_2 to the TTC node N_1 through the gateway node N_G, where it has to wait in the Out_{TTP} queue before it is transmitted on the TTP bus, in the S_G slot of node N_G.

On the TTC, the synchronisation between processes and the TDMA bus configuration is solved through the proper synthesis of schedule tables, and hence no output queues are needed. The frames sent from a TTC node to another TTC node are taken into account when determining the offsets, and are not involved directly in the ETC analysis.

The following Sections show how the worst queueing delays are calculated for each of the previous three cases.

7.1.1 Worst-case queueing delays in the Out_{N_i} and Out_{CAN} queues:

The analyses for $W_f^{N_i}$ and W_f^{CAN} are similar. Once f is the highest priority frame in the Out_{CAN} queue, it will be sent by the gateway's CAN controller as a regular CAN frame; therefore the same equation for w_f can be used:

$$w_f(q) = B_f + \sum_{\forall f_j \in hp(f)} \left\lceil \frac{w_f(q) + J_j}{T_j} \right\rceil C_j \qquad (10)$$

The intuition is that f has to wait, in the worst case, first for the largest lower priority frame that is just being transmitted (B_f) as well as for the higher priority $f_j \in hp(f)$ frames that have to be transmitted ahead of f (the second term). In the worst case, the time it takes for the largest lower priority frame $f_k \in lp(f)$ to be transmitted to its destination is

$$B_f = \max_{\forall f_k \in lp(f)} (C_k) \qquad (11)$$

Note that, in our case, $lp(f)$ and $hp(f)$ also include messages produced by the gateway node, transferred from the TTC to the ETC.

7.1.2 Worst-case queueing delay in the Out_{TTP} queue:

The time a frame f has to spend in the Out_{TTP} queue in the worst case depends on the total size of messages queued ahead of f (Out_{TTP} is a FIFO queue), $size_{S_G}$ of the data field of the frame fitting into the gateway slot responsible for carrying the CAN messages on the TTP bus, and the period T_{TDMA} with which this slot S_G is circulating on the bus [46]:

$$w_f^{TTP}(q) = B_f + \left\lfloor \frac{(q+1)s_f + I_f(w_f(q))}{S_G} \right\rfloor T_{TDMA} \qquad (12)$$

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

$$I_f(w) = \sum_{\forall f_j \in hp(f)} \left\lceil \frac{w_f + J_j}{T_j} \right\rceil s_j \qquad (13)$$

where the frame jitter J_j is given by (8).

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

8 Frame-packing optimisation strategy

The general multi-cluster optimisation strategy is outlined in Fig. 11. The MultiClusterConfiguration strategy has two steps:

1. In the first step, line 3, the application is partitioned on the TTC and ETC clusters, and processes are mapped to the nodes of the architecture using the `PartitioningAndMapping` function. The partitioning and mapping can be done with an optimisation heuristic like the one presented in [75]. As part of the partitioning and mapping process, an initial frame configuration $\psi^0 = <\alpha^0, \pi^0, \beta^0, \sigma^0>$ is derived. Messages exchanged by processes partitioned to the TTC will be mapped to TTC frames, while messages exchanged on the ETC will be mapped to ETC frames. For each message sent from a TTC process to an ETC process, we create an additional message on the ETC, and we map this message to an ETC frame. The sequence σ^0 of slots for the TTC is decided by assigning in order nodes to the slots ($S_i = N_i$). One message is assigned per frame in the initial set β^0 of TTC frames. For the ETC, the frames in the set α^0 initially hold each one single message, and we calculate the message priorities π^0 based on the deadlines of the receiver processes.
2. The frame packing optimisation is performed as the second step (line 5 in Fig. 11). The FramePackingOptimisation function receives as input the application Γ, the mapping $\mathcal{M}$ of processes to resources and the initial frame configuration ψ^0, and it produces as output the optimised frame packing configuration ψ. Such an optimisation problem is NP complete [76], so obtaining the optimal solution is not feasible. In this paper, we propose two frame packing optimisation strategies, one based on a simulated annealing approach, presented in Section 8.1, while the other, outlined in Section 8.2, is based on a greedy heuristic that uses the problem-specific knowledge intelligently in order to explore the design space.

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

Testing if the application Γ is schedulable is done using the MultiClusterScheduling (MCS) algorithm (line 7 in Fig. 11). The multicluster scheduling algorithm, presented in Fig. 8, takes as input an application Γ, a mapping $\mathcal{M}$ and an initial frame configuration ψ^0, builds the TT schedule tables, sets the ET priorities for processes and provides the global analysis.

8.1 Frame packing with simulated annealing

The first algorithm we have developed is based on a simulated annealing (SA) strategy [76], and is presented in Fig. 12. The algorithm takes as input the application Γ, a mapping $\mathcal{M}$ and an initial frame configuration ψ^0, and determines the frame configuration ψ which leads to the best degree of schedulability δ_Γ (the smaller the value, the more schedulable the system; see Section 6).

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

Fig. 11 *General frame packing strategy*

Determining a frame configuration ψ means finding the set of ETC frames α and their relative priorities π, and the set of TTC frames β, including the sequence σ of slots in a TDMA round.

The main feature of an SA strategy is that it tries to escape from a local optimum by randomly selecting a new solution from the neighbours of the current solution. The new solution is accepted if it is an improved solution (lines 9–10 of the algorithm in Fig. 12). However, a worse solution can also be accepted with a certain probability that depends on the deterioration of the cost function and on a control parameter called temperature (lines 12–13).

In Fig. 12 we give a short description of this algorithm. An essential component of the algorithm is the generation of a new solution ψ_{new} starting from the current one $\psi_{current}$. The neighbours of the current solution $\psi_{current}$ are obtained by performing transformations (called moves) on the current frame configuration $\psi_{current}$ (line 8). We consider the following moves:

- moving a message m from a frame f_1 to another frame f_2 (or moving m into a separate single-message frame)
- swapping the priorities of two frames in α
- swapping two slots in the sequence σ of slots in a TDMA round.

For the implementation of this algorithm, the parameters *TI* (initial temperature), *TL* (temperature length), ε (cooling ratio) and the stopping criterion have to be determined. They define the 'cooling schedule' and have a decisive impact on the quality of the solutions and the CPU time consumed. We are interested to obtain values for *TI*, *TL* and ε that will guarantee the finding of good quality solutions in a short time.

We performed long runs of up to 48 h with the SA algorithm, for ten synthetic process graphs (two for each graph dimension of 80, 160, 240 320, 400, see Section 9), and the best ever solution produced has been considered as the optimum. Based on further experiments we have determined the parameters of the SA algorithm so that the optimisation time is reduced as much as possible but the near-optimal result is still produced. For example, for the graphs with 320 nodes, *TI* is 700, *TL* is 500 and ε is 0.98. The algorithm stops if for three consecutive temperatures no new solution has been accepted.

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

Fig. 12 *Simulated annealing algorithm*

8.2 Frame packing greedy heuristic

The OptimizeFramePacking greedy heuristic (Fig. 13) constructs the solution by progressively selecting the best candidate in terms of the degree of schedulability.

We start by observing that all activities taking place in a multi-cluster system are ordered in time using the offset information, determined in the StaticScheduling function based on the worst-case response times known so far and the application structure (i.e. the dependencies in the process graph). Thus, our greedy heuristic outlined in Fig. 13 starts with building two lists of messages ordered according to the ascending value of their offsets, one for the TTC, $messages_\beta$, and one for ETC, $messages_\alpha$. Our heuristic is to consider for packing in the same frame messages which are adjacent in the ordered lists. For example, let us consider that we have three messages, m_1 of 1 byte, m_2 of 2 bytes and m_3 of 3 bytes, and that messages are ordered as m_3, m_1, m_2, based on the offset information. Also, assume that our heuristic has suggested two frames, frame f_1 with a data field of 4 bytes, and f_2 with a data field of 2 bytes. The PackMessages function will start with m_3 and pack it in frame f_1. It continues with m_2, which is also packed into f_1, since there is space left for it. Finally, m_3 is packed in f_2, since there is no space left for it in f_1.

The algorithm tries to determine, using the for-each loops in Fig. 13, the best frame configuration. The algorithm starts from the initial frame configuration ψ^0, and progressively determines the best change to the current configuration. The quality of a frame configuration is measured using the MultiClusterScheduling algorithm, which calculates the degree of schedulability δ_Γ (line 13). Once a configuration parameter has been fixed in the outer loops it is used by the inner loops:

- Lines 10–15: The innermost loops determine the best size S_α for the currently investigated frame f_α in the ETC frame configuration $\alpha_{current}$. Thus, several frame sizes are tried (line 11), each with a size returned by RecomendedSizes, to see if it improves the current configuration. The RecomendedSizes ($messages_\alpha$) list is built recognising that only messages adjacent in the $messages_\alpha$ list will be packed into the same frame. Sizes of frames are determined as a sum resulting from adding the sizes of combinations of adjacent messages, not exceeding 8 bytes. For the previous example, with m_1, m_2 and m_3 of 1, 2 and 3 bytes, respectively, the frame sizes recommended will be of 1, 2, 3, 4 and 6 bytes. A size of 5 bytes will not be recommended since there are no adjacent messages that can be summed together to obtain 5 bytes of data.
- Lines 9–16: This loop determines the best frame configuration α This means deciding on how many frames to include in α (line 9), and which are the best sizes for them. In α there can be any number of frames, from one single frame to n_α frames (in which case each frame carries one single message). Once a configuration α_{best} for the ETC, minimising δ_Γ has been determined (saved in line 16), the algorithm looks for the frame configuration β which will further improve δ_Γ.
- Lines 7–17: The best size for a frame f_β is determined similarly to the size for a frame f_α.
- Lines 6–18: The best frame configuration β_{best} is determined. For each frame configuration β tried, the algorithm loops again through the innermost loops to see if there are better frame configurations α in the context of the current frame configuration $\beta_{current}$.
- Lines 4–19: After a β_{best} has been decided, the algorithm looks for a slot sequence σ, starting with the first slot, and tries to find the node which, when transmitting in this slot,

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

Fig. 13 *OptimizeFramePacking algorithm*

will reduce δ_Γ. Different slot sequences are tried by swapping two slots within the TDMA round (line 5).

For the initial message priorities π^0 (initially, there is one message per frame) we use the 'heuristic optimized priority assignment' (HOPA) approach in [55], where priorities in a distributed real-time system are determined, using knowledge of the factors that influence the timing behaviour, such that the degree of schedulability of the system is improved (line 1). The ETC message priorities set at the beginning of the algorithm are not changed by our greedy optimisation loops. The priority of a frame $f_\alpha \in \alpha$ is given by the message $m \in f_\alpha$ with the highest priority.

The algorithm continues in this fashion, recording the best ever ψ_{best} configurations obtained, in terms of δ_Γ, and thus the best solution ever is reported when the algorithm finishes.

9 Experimental results

For the evaluation of our frame-packing optimisation algorithms we first used process graphs generated for experimental purposes. We considered two-cluster architectures consisting of 2, 4, 6, 8 and 10 nodes, half on the TTC and the other half on the ETC, interconnected by a gateway. Forty processes were assigned to each node, resulting in applications of 80, 160, 240, 320 and 400 processes.

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

The mapping of the applications to the architecture has been done using a simple heuristic that tries to balance the utilisation of processors while minimising communication. Execution times and message lengths were assigned randomly using both uniform and exponential distribution within the 10 to 100 ms and 1 bit to 2 bytes ranges, respectively. For the communication channels we considered a transmission speed of 256 kbit/s and a length below 20 m. All experiments were run on a SUN Ultra 10.

The first result concerns the ability of our heuristics to produce schedulable solutions. We have compared the degree of schedulability δ_Γ obtained from our OptimizeFramePacking (OFP) heuristic (Fig. 13) with the near-optimal values obtained by SA (Fig. 12). Obtaining solutions that have a better degree of schedulability means obtaining tighter worst-case response times, increasing the chances of meeting the deadlines.

Table 1 presents the average percentage deviation of the degree of schedulability produced by OFP from the near-optimal values obtained with SA. Together with OFP, a straightforward approach (SF) is presented. The SF approach does not consider frame packing, and thus each message is transmitted independently in a frame. Moreover, for SF we considered a TTC bus configuration consisting of a straightforward ascending order of allocation of the nodes to the TDMA slots; the slot lengths were selected to accommodate the largest message frame sent by the respective node, and the scheduling has been performed by the MultiClusterScheduling algorithm in Fig. 8.

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

$$deviation = \frac{\delta_\Gamma^{approach} - \delta_\Gamma^{SA}}{\delta_\Gamma^{SA}} 100 \qquad (14)$$

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

Table 1 shows that, when packing messages to frames, the degree of schedulability improves dramatically compared to

Table 1: Evaluation of frame-packing optimisation Algorithms

	Straightforward solution (SP)			Optimise frame packing (OFP)			Simulated annealing (SA)
No. of processes	Average, %	Max, %	Time, s	Average, %	Max, %	Time, s	Time, s
80	2.42	17.89	0.09	0.40	1.59	4.35	235.95
160	16.10	42.28	0.22	2.28	8.32	12.09	732.40
240	40.49	126.4	0.54	6.59	21.80	49.62	2928.53
320	70.79	153.08	0.74	13.70	30.51	172.82	7585.34
400	97.37	244.31	0.95	31.62	95.42	248.30	22099.68

the straightforward approach. The greedy heuristic OptimizeFramePacking performs well for all the graph dimensions, having, for example, run-times which are on average under 50 for applications with 240 processes.

When deciding on which heuristic to use for design space exploration or system synthesis, an important issue is the execution time. On average, our optimisation heuristics needed a couple of minutes to produce results, while the simulated annealing approach had an execution time of up to 6 h.

9.1 Vehicle cruise controller

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

The specification assumes that the CC will operate in an environment consisting of two clusters. There are four nodes which functionally interact with the CC system: the anti-lock braking system (ABS), the transmission control module (TCM), the engine control module (ECM), and the electronic throttle module (ETM) (see Fig. 14).

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

We have modelled the specification of the CC system using a set of 32 processes and 17 messages as described in [77], where the mapping of processes to the nodes is also given. The period was chosen as 250 ms, equal to the deadline.

In this setting, the straightforward approach SF produced an end-to-end worst-case response time of 320 ms, greater than the deadline, while both the OFP and SA heuristics produced a schedulable system with a worst-case response time of 172 ms.

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

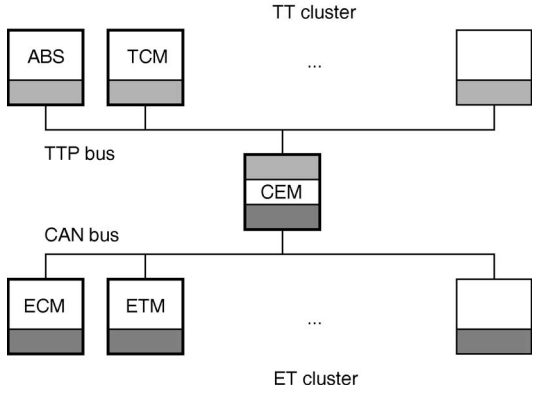

Fig. 14 *Hardware architecture for cruise controller*

10 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. To reduce costs and use the available resources more efficiently, the applications are distributed across several networks.

We have introduced the current state-of-the-art analysis and optimisation techniques available for such systems, and addressed in more detail a special class of heterogeneous distributed real-time embedded systems called multi-cluster systems.

We have presented an analysis for multi-cluster systems and outlined several characteristic design problems, related to the partitioning and mapping of functionality, and the optimisation of the access to the communication infrastructure. An approach to schedulability-driven frame packing for the synthesis of multi-cluster systems was presented as an example of solving such a design optimisation problem. We have developed two optimisation heuristics for frame configuration synthesis which are able to determine frame configurations that lead to a schedulable system. We have shown that, by considering the frame packing problem, we are able to synthesize schedulable hard real-time systems and to potentially reduce the overall cost of the architecture.

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

11 References

1 Kopetz, H.: 'Real-time systems - design principles for distributed embedded applications' (Kluwer Academic Publishers, 1997)
2 Kopetz, H.: 'Automotive electronics'. Proc. 11th Euromicro Conf. on Real-Time Systems, 1999, pp. 132–140

3 Hansen, P.: The Hansen report on automotive electronics, http://www.hansenreport.com/, July-August, 2002
4 Leen, G., and Heffernan, D.: 'Expanding automotive electronic systems', *Computer*, 2002, **35**, (1), pp. 88–93
5 Jost, K.: 'From fly-by-wire to drive-by-wire', *Automot. Eng. Int.*, 2001
6 Chiodo, M.: 'Automotive electronics: a major application field for hardware-software co-design', 'Hardware/software co-design' (Kluwer Academic Publishers, 1996), pp. 295–310
7 X-by-Wire Consortium, X-by-wire: safety related fault tolerant systems in vehicles, http://www.vmars.tuwien.ac.at/projects/xbywire/, 1998
8 Kopetz, H.: 'Automotive electronics–present state and future prospects'. 25th Int. Symp. on Fault-Tolerant Computing, 1995
9 Robert Bosch GmbH, CAN specification, version 2.0, http://www.can.bosch.com/, 1991
10 Local interconnect network protocol specification, http://www.lin-subbus.org, 2003
11 SAE Vehicle Network for Multiplexing and Data Communications Standards Committee, SAE J1850 Standard, 1994
12 Rushby, J.: 'Bus architectures for safety-critical embedded systems', *Lect. Notes Comput. Sci.*, 2001, **2211**, pp. 306–323
13 Hoyme, K., and Driscoll, K.: 'SAFEbus', *IEEE Aerosp. Electron. Syst. Mag.*, 1992, **8**, (3), pp. 34–39
14 Miner, P.S.: 'Analysis of the SPIDER fault-tolerance protocols'. Proc. 5th NASA Langley Formal Methods Workshop, 2000
15 International Organization for Standardization, 'Road vehicles–controller area network (CAN)–Part 4: Time-triggered communication', ISO/DIS 11898-4, 2002
16 Echelon, LonWorks: The LonTalk protocol specification, http://www.echelon.com, 2003
17 Profibus International, PROFIBUS DP specification, http://www.profibus.com/, 2003
18 Berwanger, J., Peller, M., and Griessbach, R.: 'A new high performance data bus system for safety-related applications', http://www.byteflight.de, 2000
19 The FlexRay Group, FlexRay Requirements Specification, Version 2.0.2, http://www.flexray-group.com/, 2002
20 Tindell, K., Burns, A., and Wellings, A.: 'Calculating CAN message response times', *Control Eng. Pract.*, 1995, **3**, (8), pp. 1163–1169
21 Audsley, N., Tindell, K., and Burns, A.: 'The end of line for static cyclic scheduling?'. Proc. Euromicro Workshop on Real-Time Systems, 1993, pp. 36–41
22 Xu, J., and Parnas, D.L.: 'On satisfying timing constraints in hard-real-time systems', *IEEE Trans. Softw. Eng.*, 1993, **19**, (1), pp. 70–84
23 Lönn, H., and Axelsson, J.: 'A comparison of fixed-priority and static cyclic scheduling for distributed automotive control applications'. Proc. Euromicro Conf. on Real-Time Systems, 1999, pp. 142–149
24 EAST-EEA project, ITEA Full Project Proposal, http://www.itea-office.org, 2002
25 Audsley, N., Burns, A., Davis, R., Tindell, K., and Wellings, A.: 'Fixed priority preemptive scheduling: an historical perspective', *Real-Time Syst.*, 1995, **8**, (2/3), pp. 173–198
26 Balarin, F., Lavagno, L., Murthy, P., and Sangiovanni-Vincentelli, A.: 'Scheduling for embedded real-time systems', *IEEE Des. Test Comput.*, 1998, **15**, pp. 71–82
27 TimeWiz, http://www.timesys.com
28 RapidRMA, http://www.tripac.com
29 RTA-OSEK Planner, http://www.livedevices.com
30 Aires, http://kabru.eecs.umich.edu/aires/
31 Liu, C.L., and Layland, J.W.: 'Scheduling algorithms for multiprogramming in a hard-real-time environment', *J. ACM*, 1973, **20**, (1), pp. 46–61
32 Tindell, K., and Clark, J.: 'Holistic schedulability analysis for distributed hard real-time systems', *Microprocess. Microprogram.*, 1994, **50**, (2-3), pp. 117–134
33 Stankovic, J.A., and Ramamritham, K.: 'Advances in real-time systems' (IEEE Computer Society Press, 1993)
34 Xu, J., and Parnas, D.L.: 'Priority scheduling versus pre-run-time scheduling', *Real Time Syst.*, 2000, **18**, (1), pp. 7–24
35 Lee, C., Potkonjak, M., and Wolf, W.: 'Synthesis of hard real-time application specific systems', *Des. Autom. Embedded Syst.*, 1999, **4**, (4), pp. 215–241
36 Dave, B.P., and Jha, N.K.: 'COHRA: hardware-software cosynthesis of hierarchical heterogeneous distributed systems', *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 1998, **17**, (10), pp. 900–919
37 Dave, B.P., Lakshminarayana, G., and Jha, N.J.: 'COSYN: hardware-software co-synthesis of heterogeneous distributed embedded systems', *IEEE Trans. VLSI Syst.*, 1999, **7**, (1), pp. 92–104
38 Burns, A., and Wellings, A.: 'Real-time systems and programming languages' (Addison Wesley, 2001)
39 Tindell, K.: 'Adding time-offsets to schedulability analysis', Department of Computer Science, University of York, Report No. YCS-94-221, 1994
40 Yen, T.Y., and Wolf, W.: 'Hardware-software co-synthesis of distributed embedded systems' (Kluwer Academic Publishers, 1997)
41 Palencia, J.C., and González Harbour, M.: 'Exploiting precedence relations in the schedulability analysis of distributed real-time systems'. Proc. 20th IEEE Real-Time Systems Symp., 1999, pp. 328–339
42 Palencia, J.C., and González Harbour, M.: 'Schedulability analysis for tasks with static and dynamic offsets'. Proc. 19th IEEE Real-Time Systems Symp., 1998, pp. 26–37
43 Ermedahl, H., Hansson, H., and Sjödin, M.: 'Response-time guarantees in ATM networks'. Proc. IEEE Real-Time Systems Symp., 1997, pp. 274–284
44 Strosnider, J.K., and Marchok, T.E.: 'Responsive, deterministic IEEE 802.5 token ring scheduling', *Real-Time Syst.*, 1989, **1**, (2), pp. 133–158
45 Agrawal, G., Chen, B., Zhao, W., and Davari, S.: 'Guaranteeing synchronous message deadlines with the token medium access control protocol', *IEEE Trans. Comput.*, 1994, **43**, (3), pp. 327–339
46 Pop, P., Eles, P., and Peng, Z.: 'Schedulability-driven communication synthesis for time-triggered embedded systems', *Real-Time Syst.*, 2004, **24**, pp. 297–325
47 Pop, T., Eles, P., and Peng, Z.: 'Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems'. Proc. Int. Symp. on Hardware/Software Codesign, 2002, pp. 187–192
48 Pop, P., Eles, P., and Peng, Z.: 'Schedulability analysis and optimisation for the synthesis of multi-cluster distributed embedded systems'. Proc. Design Automation and Test in Europe Conf., 2003, pp. 184–189
49 Richter, K., Jersak, M., and Ernst, R.: 'A formal approach to MpSoC performance verification', *Computer*, 2003, **36**, (4), pp. 60–67
50 Statemate, http://www.ilogix.com
51 Matlab/Simulink, http://www.mathworks.com
52 Ascet/SD, http://en.etasgroup.com/products/ascet_sd/
53 SystemBuild/MatrixX, http://www.ni.com/matrixx
54 TTP-Plan, http://www.tttech.com/
55 Gutiérrez García, J.J., and González Harbour, M.: 'Optimized priority assignment for tasks and messages in distributed hard real-time systems'. Proc. Workshop on Parallel and Distributed Real-Time Systems, 1995, pp. 124–132
56 Jonsson, J., and Shin, K.G.: 'Robust adaptive metrics for deadline assignment in distributed hard real-time systems', *Real-Time Syst.*, 2002, **23**, (3), pp. 239–271
57 Volcano Network Analyzer, http://www.volcanoautomotive.com/
58 Rajnak, A., Tindell, K., and Casparsson, L.: 'Volcano communications concept' (Volcano Communication Technologies AB, 1998)
59 Kienhuis, B., Deprettere, E., Vissers, K., and Van Der Wolf, P.: 'An approach for quantitative analysis of application-specific dataflow architectures'. Proc. IEEE Int. Conf. on Application-Specific Systems, Architectures and Processors, 1997, pp. 338–349
60 Tabbara, B., Tabbara, A., and Sangiovanni-Vincentelli, A.: 'Function/architecture optimisation and co-design of embedded systems' (Kluwer Academic Publishers, 2000)
61 Balarin, F., *et al.*: 'Hardware-software co-design of embedded systems: the POLIS approach' (Kluwer Academic Publishers, Boston, 1997)
62 Balarin, F., Watanabe, Y., Hsieh, H., Lavagno, L., Paserone, C., and Sangiovanni-Vincentelli, A.: 'Metropolis: an integrated electronic system design environment', *Computer*, 2003, **36**, (4), pp. 45–52
63 Virtual component co-design, http://www.cadence.com/
64 Keutzer, K., Malik, S., and Newton, A.R.: 'System-level design: orthogonalization of concerns and platform-based design', *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 2000, **19**, (12), pp. 1523–1543
65 TTTech, TTP/C specification version 0.5, 1999, availabe at http://www.tttech.com/
66 TTTech, Comparison CAN-Byteflight-FlexRay-TTP/C, Technical Report, availabe at http://www.tttech.com/
67 Nolte, T., Hansson, H., Norström, C., and Punnekkat, S.: 'Using bit-stuffing distributions in CAN analysis'. Proc. IEEE/IEE Real-Time Embedded Systems Workshop, 2001
68 Pop, P., Eles, P., and Peng, Z.: 'Scheduling with optimized communication for time triggered embedded systems'. Proc. Int. Workshop on Hardware-Software Codesign, 1999, pp. 178–182
69 Puschner, P., and Burns, A.: 'A review of worst-case execution-time analyses', *Real-Time Syst.*, 2000, **18**, (2/3)
70 Kopez, H., and Nossal, R.: 'The cluster-compiler - a tool for the design of time triggered real-time systems'. Proc. ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Real-Time Systems, 1995, pp. 108–116
71 Sandström, K., and Norström, C.: 'Frame packing in real-time communication'. Proc. Int. Conf. on Real-Time Computing Systems and Applications, 2000, pp. 399–403
72 Pop, P., Eles, P., and Peng, Z.: 'Bus access optimisation for distributed embedded systems based on schedulability analysis'. Proc. Design Automation and Test in Europe Conf., 2000, pp. 567–574
73 Eles, P., Doboli, A., Pop, P., and Peng, Z.: 'Scheduling with bus access optimisation for distributed embedded systems', *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, 2000, pp. 472–491
74 Audsley, N.C., Burns, A., Richardson, M.F., and Wellings, A.J.: 'Hard real-time scheduling: the deadline monotonic approach'. Proc. 8th IEEE Workshop on Real-Time Operating Systems and Software, 1991, pp. 127–132
75 Pop, P., Eles, P., Peng, Z., Izosimov, V., Hellring, M., and Bridal, O.: 'Design optimisation of multi-cluster embedded systems for real-time applications'. Proc. Design, Automation and Test in Europe Conf., 2004, pp. 1028–1033
76 Reevs, C.R.: 'Modern heuristic techniques for combinatorial problems' (Blackwell Scientific Publications, 1993)
77 Pop, P.: 'Analysis and Synthesis of Communication-Intensive Heterogeneous Real-Time Systems'. PhD dissertation, Linköping Studies in Science and Technology, 2003, No. 833, 2003