Modeling, Designing and Analyzing Resource Reservations in Distributed Embedded Systems

Mohammad Ashjaei, Nima Khalilzad, Saad Mubeen

Abstract Distributed embedded systems in many domains are becoming highly complex, mostly due to ever-increasing demand for advanced computer controlled functionality. These systems are realized by several embedded systems communicating through network channels. These systems are often required to be predictable, i.e., their responses to internal or external stimuli should be delivered within the constraints that are specified on them. Compositional development methods have been proposed by the research community to lower the software complexity, ensure predictability and allow flexibility during the development and execution of these systems. According to these methods, the compute and communication resources are allocated to each part (or sub-system) of the system, which in turn brings isolation among the parts and eases the system integration. This chapter presents a new end-to-end resource reservation model for distributed embedded systems that covers not only the computational nodes but also the communication channels. Moreover, timing analysis is presented to verify the predictability of the systems. This chapter also describes guidelines to distribute resources efficiently among different parts of the system. As a proof of concept, the end-to-end resource reservation model is implemented in the Rubus Component Model. This component model is already used for the development of control functionalities in vehicular embedded systems by several international companies. In order to show the usability of the proposed model, reservation design method, end-to-end timing analysis, and extended component model, a vehicular application case study is conducted and several experiments are performed.

Mohammad Ashjaei
Mälardalen University, Västerås, Sweden, e-mail: mohammad.ashjaei@mdh.se

Nima Khalilzad
Qamcom Research and Technology, Stockholm, Sweden e-mail: nima.khalilzad@qamcom.se

Saad Mubeen
Mälardalen University, Västerås, Sweden e-mail: saad.mubeen@mdh.se

1 Introduction

Embedded systems play a vital role in many domains. These systems are inevitably becoming part of our daily lives, ranging from small electronic devices to more sophisticated systems. Generally, small electronic devices contain only one computing unit that performs a particular function specific to the device. The computing unit is called embedded system if it is embedded inside the device that it controls. In large and sophisticated systems, such as aircrafts and modern vehicles, the functionality is developed over several embedded systems that usually communicate via one or more networks. Such a system is typically known as a *distributed embedded system*. Often, distributed embedded systems are constrained by end-to-end timing requirements, which requires them to react to the external or internal events within specified times. Hence, these systems should be *predictable*.

In the last decades, distributed embedded systems in many domains have become intricate. This intricacy is contributed not only by advanced functions demanded by the market, but also by other complementary requirements, such as significant amount of data exchange within heterogeneous networks, use of powerful computing platforms to fulfill the high resource demand, and the new trend towards developing the systems in a composable way. Furthermore, the size and complexity of software in such systems has been drastically increasing over time. For instance, the amount of software in a car has increased from 100 lines of code, in late 1970s, to more than 100 million lines of code in a span of approximately three decades [13, 61]. This software may consist of as many as 2000 software functions. One way to deal with such complexity during the development is to split the system into several sub-systems. The sub-systems are then developed by different teams or tier-1 suppliers independently. Nevertheless, the sub-systems still share the system resources, which are both processor and network resources. In order to support isolation and independence among the sub-systems after their deployment, a fraction of the system resources is reserved for each sub-system by means of so-called *resource reservation* (also called resource partitioning) techniques [57], [19]. For instance, a portion of processor time can be allocated to a specific sub-system using the hierarchical reservation framework [39], [51]. Similarly, there are several techniques to achieve temporal isolation among sub-systems in the network domain. For instance, a multi-level hierarchical framework is presented in [52] that supports resource reservations in switched Ethernet networks. As a result, the predictability requirements of each sub-system can be verified independently. The sub-system can be integrated together without a need for analyzing the complete system as long as the timing behavior of each part has been independently verified and the set of reservations is feasible.

As mentioned before, many industrial systems are in fact distributed real-time embedded systems. This means that a sub-system of such a system can span over more than one processor, while the processors communicate with each other via a field bus[1] or other kind of real-time networks. The resource reservations for the sub-

[1] Typical name given to real-time networks developed specifically for use in industrial plants.

system includes a joint reservation on computation and communication resources. Such a reservation is called *end-to-end resource reservation*. Development of distributed real-time embedded systems using the end-to-end resource reservations has received very less attention. One example can be found in [38], which proposes a distributed kernel to guarantee the end-to-end timeliness. Another example can be seen in [49] that provides a distributed resource management system, called D-RES.

From the software development point of view, the research community has proposed to use the principles of Model Based Development (MBD) and Component-Based Software Engineering (CBSE) [29, 15]. MBD uses models to describe functions, structures and other design artifacts. Whereas, CBSE raises the level of abstraction for software development by reuse and integration of software components and their architectures. The development models and techniques that comply with MBD and CBSE can further mitigate the complexity by allowing flexible development and execution of the sub-systems. In this chapter we consider the vehicular domain for a proof of concept. However, it should be noted that the software development techniques presented in this chapter are generally applicable to distributed real-time embedded systems. The existing support to model resource reservations at various levels during the development of these systems is shown in Fig. 1.

At the top level in Fig. 1, software architecture of the system is modeled in terms of software components and their interconnections. The existing component models and tools in the vehicular domain fully support the modeling of resource reservations in the network, e.g., SymTA/S tool [28] which is a part of the AUTOSAR [2] tool chain. There is a very limited support for modeling resource reservations in the computational nodes by the existing component models and tools. To the best of our knowledge, no existing component model in the vehicular domain supports modeling of end-to-end resource reservations. One of the main objectives of this chapter is to provide such a modeling support for the software architecture of these systems.

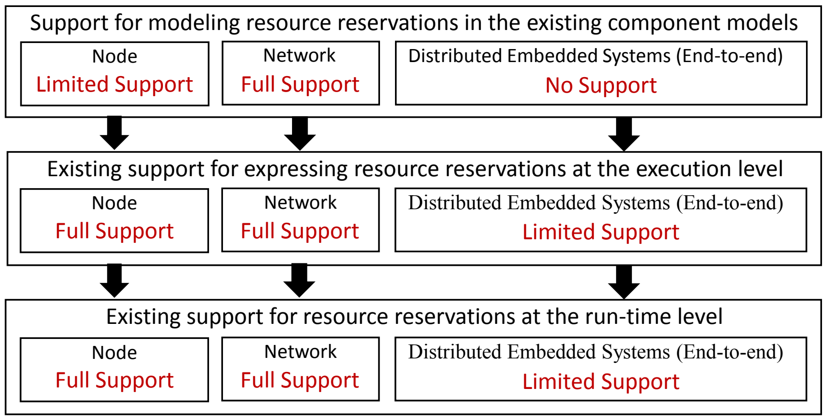

Support for modeling resource reservations in the existing component models
Node
Limited Support
Network
Full Support
Distributed Embedded Systems (End-to-end)
No Support
Existing support for expressing resource reservations at the execution level
Node
Full Support
Network
Full Support
Distributed Embedded Systems (End-to-end)
Limited Support
Existing support for resource reservations at the run-time level
Node
Full Support
Network
Full Support
Distributed Embedded Systems (End-to-end)
Limited Support

Fig. 1 Existing support for modeling resource reservations at various levels in the vehicular domain.

At the second level in Fig. 1, *execution model* of the system is developed. At this level, the focus is on tasks, messages, nodes and networks. An execution model provides the information regarding : (1) what executes? (2) where does it execute? and (3) when does it execute? There is a large number of execution-level frameworks in the vehicular domain that support resource reservations on nodes and networks. However, there are very few works that support end-to-end resource reservations on distributed embedded systems at the execution level. The third level, shown in Fig. 1, provides run-time environment for the executable entities from the second level. At this level, the focus is on operating system, scheduling, network protocol stack, interface drivers, etc. There is a strong existing support for resource reservations in nodes and networks at the run-time level. However, there is a limited support for end-to-end resource reservations at this level.

This chapter presents a new end-to-end resource reservation model that supports reservation on both computation and communication resources in predictable distributed embedded systems[2]. The model covers several real-time network protocols. The proposed model is the first comprehensive model to support end-to-end resource reservations in distributed transactions (chains of tasks and messages) with various activation patterns such as trigger chains, data chains and mixed chains. These activation patterns are commonly used in the industrial control systems [22, 46]. In order to guarantee the predictability of the model, an end-to-end timing analysis is presented. The analysis not only computes end-to-end response times, but also end-to-end data path delays in distributed reserved resources. In particular, the analysis computes the age and reaction delays which find their importance in the control systems and body electronics domains of the automotive industry. This chapter also presents a method to design reservations for both computation and communication resources in order to guide designers for allocating resources in an efficient way. Furthermore, this chapter extends a component model to enable the modeling and design of end-to-end reservations, albeit in the context of vehicular domain. This extension is carried out in the Rubus Component Model (RCM) [26], which is already used by several international companies in the vehicular domains. The main reason for selecting RCM is that it has a small run-time foot print (timing and memory overhead) compared to several other models such as AUTOSAR. Finally, this chapter presents an experiment to show the performance and usability of the presented models and techniques.

The rest of the chapter is organized as follows. Section 2 reviews the background and prior work. Section 3 presents a new end-to-end resource reservation model for distributed embedded systems, while Section 4 presents the response time and end-to-end delay analysis. Section 5 presents a design guide for resource reservations. Section 6 discusses extension of RCM to support the end-to-end resource reservation technique. An experiment is presented in Section 8 using a case study in the area of vehicular systems. Finally, Section 9 concludes the chapter and highlights future challenges in this area.

[2] This book chapter is adapted from previously published contributions [8], [44].

2 Background and Prior Work

This section introduces the background concepts that are required for understanding the models and techniques. In addition, the work presented in this chapter builds upon a large body of prior work in the area of real-time systems and component-based software engineering. Therefore, this section also reviews the relevant state of the art regarding the presented models and techniques.

2.1 Real-time systems

Embedded systems are mostly components of a larger system, where they often interact with the environment through sensors and actuators. Many embedded systems are constrained by timing requirements, i.e, the response by the system must be delivered within a specified time, e.g., a deadline or a delay constraint. In these systems the logical correctness of the response is as important as the time at which the response is delivered. Such systems are called real-time embedded systems. Depending upon the consequences that may occur because of a violated timing requirement, a real-time system can be categorized into two groups: Hard and Soft. If violation of a timing requirement (e.g., missing a deadline) results in the failure of the system, the system is referred to as hard real-time system. On the other hand, if the consequences of violating a timing cause performance degradation but the response has some utility for the system, the system is called soft real-time system.

2.2 Schedulability analysis

As the major requirement of real-time systems is to provide actions in a timely manner, an accurate prediction of timing behavior is required. In order to verify that the actions by the system meet their respective timing requirements, schedulability analysis techniques have been developed [56, 9, 10]. This section discusses two such techniques, namely response-time analysis and end-to-end delay analysis. These techniques are commonly used for the timing analysis of real-time systems.

2.2.1 Response time analysis

One schedulability analysis technique is based on calculating the *worst-case response times* of tasks/messages and comparing them to the corresponding deadlines. This technique is known as Response Time Analysis (RTA) [34]. The response time of a message is the time interval between its activation time at the source node and its reception at the destination node. RTA is a well-known method in the real-time community and it has been used in many domains including single-core systems, networks, multi-core systems and distributed systems. When using this technique the response time of a task/message is calculated iteratively with a pseudo-polynomial run-time complexity, with the aim of finding the worst-case scenario for the task/message. The interference from the other tasks/messages and their activation frequencies are also taken into account in the calculations.

In the context of single-core systems and assuming the fixed-priority preemptive scheduling [56, 10], the worst-case scenario for a task occurs when all higher priority tasks are released simultaneously with the task, known as the *critical instant* for the task [41]. In the above mentioned situation (i.e., at critical instant), the response time R_i of a task τ_i is calculated by the sum of the task computation time C_i, the interference from higher priority tasks denoted by I_i and blocking due to sharing a common resource with lower priority tasks, denoted by B_i. Eq. 1 shows the response-time calculation for task τ_i.

$$R_i = C_i + B_i + I_i \tag{1}$$

The interference from higher priority tasks is calculated according to Eq. 2 considering all activations of higher priority tasks during the response time of task τ_i, where $hp(i)$ is the set of tasks with priority higher than that of τ_i. T_j represents the period of a higher priority task.

$$I_i = \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \tag{2}$$

Combining equations 1 and 2 derives an iterative equation shown in Eq. 3, where $R_i^0 = C_i$ and the result is derived when $R_i^{n+1} = R_i^n$ or the deadline is violated (the system is unschedulable).

$$R_i^{n+1} = C_i + B_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j \tag{3}$$

Note that the response time calculations presented above are valid for the fixed-priority preemptive scheduling. In the case of Earliest Deadline First (EDF) scheduling, the computation is different [27].

The same algorithms can be applied for messages considering different way of dealing with interference, blocking and critical instants. Davis *et al.* [17] presented the calculations for worst-case response times for messages in Controller Area Network (CAN) [33]. In the case of Ethernet AVB networks, Bordoloi *et al.* [12] computed the response time of traffic class A and B. Furthermore, Marau *et al.* [43] and Santos *et al.* [52] used this technique to compute the worst-case response times of Ethernet messages in a switched Ethernet network.

2.3 End-to-end delays

In a real-time system, tasks may be executed with precedence constraints. This means that a task is only allowed to be executed after the finishing time of its predecessor task (among which the precedence constraint exist). Moreover, a task may get input data from the execution of another task even when they are executed independently. This makes a *task chain*, where each task may be triggered by its predecessor task or independently. When a system is modeled with task chains then the schedulability of the system can be determined by calculating end-to-end response time and end-to-end delays in each chain and comparing them with the corresponding deadlines. In order to understand the difference between the end-to-end response time and end-to-end delays consider two task chains as shown in Fig. 2. Both task chains contain three tasks and each task is assumed to have the Worst Case Execution Time (WCET) of 1 ms. The tasks have equal priorities and they communicate with each other by writing to and reading from the registers[3]. There is only one activating source for the first chain, shown in Fig. 2(a), that activates τ_1 periodically with a period of 8 ms. The rest of the tasks in the chain are activated by their predecessors. Such a chain is called a *trigger chain* [46]. The input of the chain corresponds to the data read by τ_1 from register Reg-1. The input may correspond to the data that arrives from a sensor. On the other hand, the data written by τ_3 to Reg-4 is considered as the output of the task chain. The output may correspond to the control data for an actuator. The end-to-end response time of the task chain is defined as the amount of time elapsed between the arrival of an event at the first task and production of the response by the last task in the chain. The arrival of an event corresponds to the periodic activation of task τ_1. Assuming no interferences, the end-to-end response time of the chain can be calculated by summing up the WCETs of all tasks in the chain, which is 3 ms in this example.

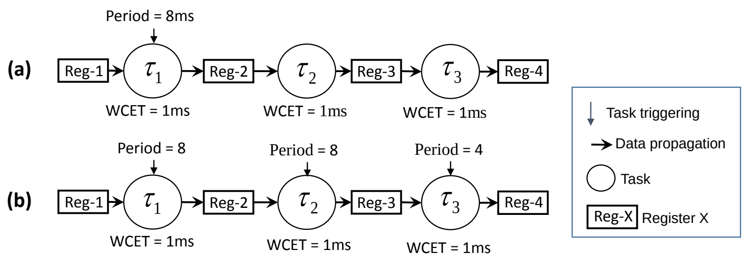


Fig. 2 Activation patterns: (a) a trigger chain, (b) a data chain.

Now consider the task chain shown in Fig. 2(b). In this chain, each and every task is activated independently, that means no task is activated by its predecessor. Such a chain is called a *data chain* [46]. It should be noted that a mixed chain includes

[3] A register may correspond to a port of a software component which is realized by the task at run-time.

both trigger and data sub-chains. The periods of activation for tasks τ_1, τ_2 and τ_3 are 8 ms, 8 ms and 4 ms, respectively. Since each task in the chain is activated independently with a different clock, there can be several time paths through which the data can traverse from the input to the output. Hence, there can be multiple outputs that correspond to one single input of the chain as shown by the uni-directional curved arrows in Fig. 3. As a result there are different delays between the arrival of the input data and production of the output data by the chain.

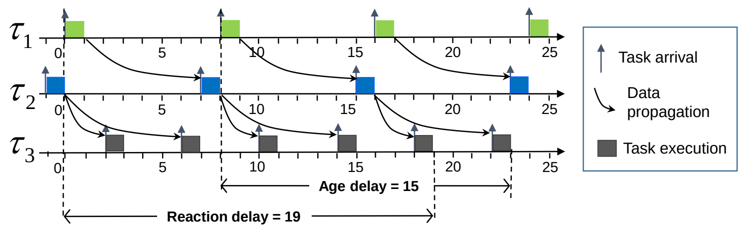


Fig. 3 A possible execution trace for the transaction in Fig. 2(b).

In this work, two such delays are described, namely *age* and *reaction* [22]. The data age delay is equal to the time elapsed between the arrival of data at the input and the latest availability of the corresponding data at the output. For example, assume that the data is available in Reg-1 (input of the chain) at time 8 when it is read by the first task (τ_1). Task τ_1 writes the corresponding data to Reg-2 at time 9. The data is read by task τ_2 from Reg-2 at time 15. Task τ_2 then writes the corresponding data to Reg-3 at time 16. Task τ_3 reads the same data from Reg-3 at two different times, i.e., at 18 and 22. Moreover, task τ_3 writes the corresponding data to Reg-4 (output of the chain) at times 19 and 23. It can be seen that there are two samples of output data corresponding to one single sample of the input data. In the age delay, the longest time difference between the input data and corresponding output data is of interest. On the other hand, the data reaction delay corresponds to the earliest availability of the data at the output corresponding to the data that *just missed* the read access at the input. Assume that just after task τ_1 started its execution at time 0, new data arrives in Reg-1. Hence, the new data just misses read access of τ_1. The new data has to wait in Reg-1 until it is read by τ_1 at time 8. The earliest effect of this data appears at the output at time 19 which corresponds to the data reaction delay. In this example, the age delay is 15 ms, while the reaction delay is 19 ms.

The data age delay is important, in particular, for control applications where freshness of the data is of value. Whereas, the data reaction delay is important in the applications where the first reaction to the input is of value.

2.4 Resource Reservation

The sub-systems, after being split from a system, still share the system resources in terms of processors and network bandwidth. In order to support isolation and independence among the subsystems after their development, resource reservation techniques are used. This section describes methods for reserving resources in processors, networks and distributed embedded systems.

2.4.1 Reservation in the processor domain

There are many resource reservation techniques in the processor domain such as virtual machines [11] and reservation support for time-sensitive multimedia applications [23]. Shared-driven scheduling is an alternative scheduling method in which different sub-systems can be scheduled in isolation and with different policies. This method of scheduling, i.e., using server-based scheduling, helps in reserving resources for each sub-system. In this scope, several proposals are presented in the literature based on different types of servers, such as using deferrable servers [60], sporadic servers [59] and hierarchical partitions [51]. In the above mentioned works, the proposals are complemented with a response time analysis. In order to present a tight analysis, many methods are adopted. Lipari *et al.* [39] proposed an analysis for periodic servers. The analysis for hierarchical partitioned scheduling is presented by Shin *et al.* [57], which considers a generic periodic server model for both fixed-priorities and EDF scheduling algorithms. The analysis presented by Almeida and Pedreiras [5] uses the same concept as in [57], however it covers a more realistic task model with release jitter.

2.4.2 Reservation in the network domain

Generally, the same concept as used in the processor domain is also used for resource management in networks. A general category of resource management in networks is traffic shapers [42]. Shapers limit the amount of load submitted to the network by using different policies such as credit-based shaping, e.g., as done by Ethernet AVB networks. Moreover, some real-time Ethernet protocols enforce a cyclic-based transmission in which each cycle is divided into several transmission phases. This way, a transmission window is allocated to a set of messages, resembling server-based scheduling inherent in the processor domain. In addition, the hierarchical scheduling model has been used in the network domain, such as the hierarchical scheduling framework implemented by the FTT-SE protocol [31] and the HaRTES architecture [52].

2.4.3 Reservation in distributed system domain

In the context of distributed embedded systems, very few works have addressed the reservation of resources, where the resources include both processors and networks. Some of these works consider soft real-time systems targeting multimedia applications, such as [58] and [20]. An adaptive QoS control is developed by Cucinotta *et* *al.* [16] to control the reservation during run-time. An alternative framework is proposed by Khalilzad *et al.* [36] that supports real-time components containing several tasks and messages. However, the above frameworks are presented for soft real-time systems.

A distributed kernel framework is developed by Lakshmanan *et al.* [38]. It guarantees the satisfaction of end-to-end timing requirements in distributed real-time applications. The application model is based on a distributed task graph, where each application is divided into several sub-tasks that communicate with each other. Each sub-task is allocated to one processor and the end-to-end deadline is decomposed to local deadlines. The reservation on the network is achieved by traffic shaping. A general model, called Q-RAM [50], provides sharing of resources among multiple applications by controlling QoS in the system. The Q-RAM model has been extended to cope with the systems with rapid changes [25]. However, the main focus in these works is on distributed tasks while the network protocols have not received much attention.

A global resource management model, called D-RES [49], supports reservations on both nodes and networks with end-to-end timing guarantees. In this model a sub-system contains tasks and message, albeit with one task executing in each node. Section 3 presents a distributed resource reservation model, which seemingly resembles the model in D-RES. Nevertheless, the model in this work allows a sub-system to contain multiple distributed transactions each of which consists of several tasks executing in a single node. Moreover, the model supports various activation patterns for transactions such as trigger, data and mixed chains.

2.5 Reservation design

Some of the above mentioned proposals already address the problem of designing servers. For instance, the work in [5] discusses the problem of assigning server parameters with the goal of using least system resources. Search-based approaches are proposed for this purpose. The work in [40] presents a technique to design periodic servers in a hierarchical framework targeting the schedulability of real-time applications. In order to find the best server parameters, the work defines a cost function according to the processor overhead. Within the hierarchical scheduling framework, the work in [18] investigates the problem of selecting the server parameters. Several techniques are presented in the work to select the server parameters given that the rest of the system parameters are known. The techniques include the binary search and greedy algorithms. Moreover, the recent work presented in [6] addresses a server-based reservation mechanism for control applications. The goal for designing the servers in these applications is to provide optimum bandwidth such that the control tasks are stabilized. To solve the problem, the work proposes to use the Karush-Kuhn-Tucker (KKT) necessary conditions which are used in nonlinear programing.

The work in [32] focuses on the design of server reservation for the multi-level hierarchical framework applied in the network, unlike the previous works that mostly consider a two-level hierarchy. In order to provide the server parameters, the approach in [6] uses two steps. The first step provides the parameters of the leaf servers. Whereas, the second step generates the parameters for the intermediate and root servers. The reservation design solution presented in this work considers the end-to-end reservation rather than one reservation.

The work in [24] presents a polynomial-time algorithm to allocate bandwidth to handle sporadic tasks that are scheduled by earliest deadline first with EDP server models. The algorithm gives an approximation on the partitions for real-time components. In the same context, the work presented in [62] uses real-time calculus to design the servers based on a given interface. The algorithm computes demand and service curves for components in real-time systems.

2.6 Component models

There are several component models and modeling languages for vehicular distributed embedded systems. EAST-ADL [4] is an architecture description in the automotive domain. It is complemented by the TADL2 language [1] to support modeling of timing information. EAST-ADL and EAST-ADL-like models[4] that are used in the industry do not support modeling of end-to-end resource reservations. AUTOSAR [2] is a standardized software architecture for automotive embedded systems. It is complemented with a strong tool chain that supports various development steps, i.e., from modeling of the software architecture to its execution. The tool chain supports the modeling of resource reservations but only in the networks. For example, the approach used in SymTA/S tool allows modeling and analysis of switched Ethernet networks. It supports modeling of resource reservation only at the network level. In comparison, we aim to support the modeling of end-to-end resource reservations at the distributed system level. There are several other component models, e.g., ProCom [55], COMDES [35] and middleware approaches like CORBA [3]. However, none of these models and approaches support end-to-end resource reservations in the software architectures.

The modeling technique discussed in this chapter is an extension of RCM and Rubus-ICE to support modeling of end-to-end resource reservation. The Rubus concept, developed by Arcticus Systems[5], provides a collection of methods and tools for model- and component-based development of control functionality in vehicles. It has been used in the vehicle industry for over 20 years [45]. The Rubus concept is based around RCM and its tool suite, Rubus-ICE that consists of modeling tools, code generators, analysis tools and run-time infrastructure. The main goal of Rubus is to be aggressively resource efficient and to provide means for develop-

[4] For example, SE Tool and SystemWeaver (http://www.systemweaver.se).

[5] http://www.arcticus-systems.com.

ing predictable, timing analyzable and synthesizable control functions in resource-constrained embedded systems. The highest-level hierarchical element in RCM is called the system. It contains nodes (ECUs) and networks. The lowest-level hierarchical component in RCM is called the Software Circuit (SWC). It encapsulates basic functions and has run-to-completion semantics. Note that the software circuit is synonymous to a software component in component-based software engineering.

3 End-to-end Reservation Model

We consider the model of a real-time distributed embedded system that consists of multiple nodes. The nodes are connected to a single network that allows real-time communication. The nodes are assumed to be single-core processors executing real-time tasks according to the fixed-priority preemptive scheduling. The system, denoted by $\mathcal{S}$, consists of N number of applications as shown in Eq. 4. An application performs a specific function and is commonly distributed over several nodes. Note that an application is a formal definition of a sub-system in this model. For example, the cruise control function in a car is one application.

$$\mathcal{S} = \{\mathcal{A}^{(1)}, ..., \mathcal{A}^{(N)}\} \tag{4}$$

3.1 Application and transactional models

An application consists of a set of M transactions as shown in Eq. 5. A transaction is denoted by $\Gamma_i^{(h)}$, where i denotes its identifier and h denotes the identifier of the application to which it belongs. The transaction identifiers are unique within one application. However, two transactions belonging to different applications can have same transaction identifiers.

$$\mathcal{A}^{(h)} = \{\Gamma_1^{(h)}, ..., \Gamma_M^{(h)}\} \tag{5}$$

A transaction consists of a chain (or sequence) of real-time tasks and messages which can have different activation patterns, i.e., a task or a message can be activated (released for execution or transmission) independently (e.g., either periodically by a clock or sporadically by an external event) or by its predecessor task or message. The tasks and messages in transaction $\Gamma_i^{(h)}$ are presented in Eq. 6.

$$\Gamma_i^{(h)} = \{\tau_{j,i,k}, ...; m_{i,k}, ...\} \tag{6}$$

The subscript, k, in $\tau_{j,i,k}$ represents the task identifier, whereas the first and second subscripts, j and i, specify the identifiers of the node and transaction to which this task belongs. We allow any two transactions to share the same task or message, provided that the transactions belong to the same application. Note that sharing of tasks or messages between two applications is not allowed. A transaction can include more than one task from the same node. The task identifiers are assumed to be unique within each node. For instance, $\tau_{1,2,2}$ is task 2 in transaction 2 and it executes in node 1, whereas $\tau_{2,2,2}$ is also task 2 in transaction 2, however it executes in node 2. Moreover, in Eq. 6, $m_{i,k}$ represents message k in transaction i. Since we consider one network and assume the message identifiers to be unique, we do not use network identifier with the message.

It should be noted that a transaction cannot initiate or terminate with a message because a message must have a sender task and, at least, one receiver task. The transactional model is general in the sense that it can accommodate any number of tasks and messages that are in a chain. A transaction can also be composed of only one task, e.g., $\Gamma_1^{(1)} = \{\tau_{1,1,1}\}$. Such a transaction does not generate any message.

In order to understand the sharing of tasks among transactions, consider the example of a system that consists of three nodes and one network as shown in Fig. 4. There are three tasks in Node 1; two tasks in Node 2; three tasks in Node 3; and two messages in the network. There is one application in the system that consists of two transactions denoted by $\Gamma_1^{(1)}$ and $\Gamma_2^{(1)}$ as depicted in Fig. 4. Both of these transactions are initiated by the same task (i.e., the task with identifier 1 are shared between two transactions) in Node 1. This task is denoted by $\tau_{1,1,1}$ in transaction $\Gamma_1^{(1)}$ and by $\tau_{1,2,1}$ in transaction $\Gamma_2^{(1)}$. A shared task among two or more transactions can be identified by comparing the first and third subscripts of each task in one transaction with the first and third subscripts of each task in every other transaction within the same application. The sequence of tasks and messages included in $\Gamma_1^{(1)}$ and $\Gamma_2^{(1)}$ are as follows.

$\Gamma_1^{(1)}$: $\tau_{1,1,1} \rightarrow \tau_{1,1,2} \rightarrow m_{1,1} \rightarrow \tau_{3,1,1} \rightarrow \tau_{3,1,2} \rightarrow \tau_{3,1,3}$
$\Gamma_2^{(1)}$: $\tau_{1,2,1} \rightarrow \tau_{1,2,3} \rightarrow m_{2,2} \rightarrow \tau_{2,2,1} \rightarrow \tau_{2,2,2}$.

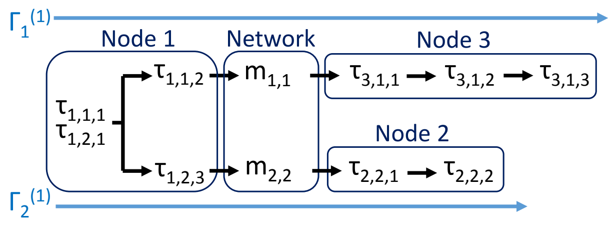


Fig. 4 An example of two transactions with task sharing.

3.2 *Task model*

Each task is characterized by the following tuple.

$$\tau_{j,i,k} =< T_{j,i,k}, C_{j,i,k}, P_{j,i,k}, J_{j,i,k}, O_{j,i,k}, V_{j,i,k} > \tag{7}$$

A task can be activated periodically or sporadically. In the case of periodic activation, $T_{j,i,k}$ represents its activation period. Whereas, in the case of sporadic activation, $T_{j,i,k}$ represents the minimum inter-arrival time between any two consecutive activations of the task. If the task is activated by its predecessor, a precedence constraint is specified between the task and its predecessor. Such information is stored in $V_{j,i,k}$, which is null if the task is not activated by its predecessor. Note that in this model each task can have only one predecessor which activates the current task. The terms $C_{j,i,k}$, $P_{j,i,k}$ and $J_{j,i,k}$ in Eq. 7 represent the WCET, priority and maximum release jitter of the task. Note that the lower the value of $P_{j,i,k}$, the higher the priority. The offset for a periodic task is denoted by $O_{j,i,k}$. An offset is an externally imposed delay between the times when the task can be released for execution. Offsets for sporadic tasks are assumed to be zero. We assume that there is no synchronization among tasks, i.e., the tasks do not use locks to protect shared data.

3.3 Message model

A message is characterized by the following tuple.

$$m_{i,k} =< T_{i,k}, C_{i,k}, P_{i,k}, J_{i,k}, O_{i,k}, V_{i,k}, \mathcal{L}_{i,k} > \tag{8}$$

A message can be queued for transmission either periodically or sporadically. If the message is periodic, $T_{i,k}$ represents its period. Whereas, if the message is sporadic then $T_{i,k}$ specifies the minimum inter-arrival time that should elapse between the queuing of its any two consecutive instances. The worst-case transmission time, priority and release jitter of the message are denoted by $C_{i,k}$, $P_{i,k}$ and $J_{i,k}$. Note that the higher priority is shown by the lower value for $P_{i,k}$. The offset of the message is denoted by $O_{i,k}$. The identifier of the task that sends the message $m_{i,k}$ is specified in $V_{i,k}$. The message model is applicable to several real-time network protocols such as real-time switched Ethernet extensions and Controller Area Network (CAN). In the case of switched Ethernet, we assume the communication is full duplex. This means that each network connection is composed of two independent unidirectional links with opposite directions. The message traverses through a set of links, denoted by $\mathcal{L}_{i,k}$, that are modeled in Eq. 9, where l_x and l_y are the first and last links the message crosses through. Note that the index of a link is unique within the network and they are shared among applications.

$$\mathcal{L}_{i,k} = \{l_x, ..., l_y\} \tag{9}$$

3.4 Resource model

Each application consumes a set of q resources. As shown in Eq. 10, $R^{(h)}$ denotes the resource set for h^{th} application, whereas r_q represents the q^{th} resource. The resource index, denoted by q, is unique within the system. Eq. 10 includes node and communication resources. In a switched Ethernet network, one resource is defined per link.

$$R^{(h)} = \{r_1, ..., r_q\} \tag{10}$$

The total resources consumed by the system are obtained by performing the union operation on the resources consumed by all applications as shown in Eq. 11.

$$\mathcal{R} = \bigcup_{\forall h \in \mathcal{S}} \{R^{(h)}\} \tag{11}$$

The resources are allocated to the applications using the periodic resource reservation policy. According to this policy, each application periodically receives a portion of bandwidth in each resource that it requires. We define the interface of an application $\mathcal{A}^{(h)}$ as the properties of its reservations on q resources, and we denote the application interface using $I^{(h)}$:

$$I^{(h)} =< \{\Pi_1^{(h)}, ..., \Pi_q^{(h)}\}, \{\Theta_1^{(h)}, ..., \Theta_q^{(h)}\}, \{\Psi_1^{(h)}, ..., \Psi_q^{(h)}\} > . \tag{12}$$

Above, the replenishment period for resource r_i is denoted by $\Pi_i^{(h)}$. The amount of budget to be replenished is denoted by $\Theta_i^{(h)}$. In addition, we define a priority for the resource reservation which is represented by $\Psi_i^{(h)}$ for resource r_i. The motivation behind the inclusion of the priority for each reservation is explained in the next subsection.

3.5 Node model

In order to support the resource reservation in nodes, we consider periodic servers [60]. We assume that there is one server per application per node that schedules the tasks assigned to it. Each node implements a two-level hierarchical scheduler, where both levels use the fixed-priority preemptive scheduling. The top-level scheduler, called the *global level scheduler*, schedules the servers based on the priority of each reservation (Ψ). The second-level scheduler is called the *local level scheduler* that schedules the tasks assigned to it. Fig. 5 shows an example of one node that implements a two-level hierarchical scheduler. The global level scheduler schedules two servers that schedule tasks belong to two applications within the node. Each server has its own local scheduler to schedule the tasks that are assigned to it.

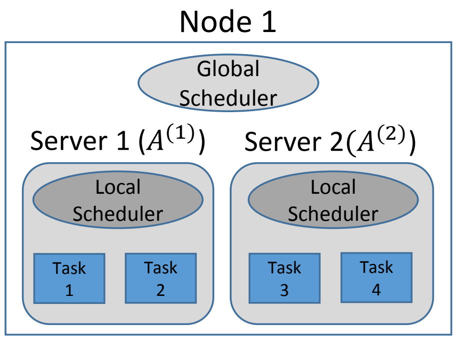


Fig. 5 Example of a node with a two-level hierarchical scheduler.

3.6 Network model

For the network model any communication protocol that can support a resource reservation mechanism can be used. The response-time analysis for messages is specific for each network protocol. Therefore, as a proof of concept, we chose the HaRTES architecture [7] because it allows independent activation of messages unlike the CAN and AVB networks. Moreover, the reservation mechanism in the HaRTES switch is more flexible than other technologies due to the hierarchical framework. Hence, we describe the HaRTES architecture in more details. The HaRTES switch is a modified Ethernet switch that provides real-time communication with a resource management mechanism. The HaRTES architecture is developed by connecting several HaRTES switches in a tree topology. The HaRTES architecture separates the traffic into two classes, synchronous (periodic) and asynchronous (sporadic). The transmission is performed within pre-configured Elementary Cycle (EC), which is a fixed-duration time slot. Each EC is divided into two transmission windows, one per traffic class. The EC along with the two windows for a link is shown in Fig. 6. Note that the size for the windows in each link can be different according to the load on the link.

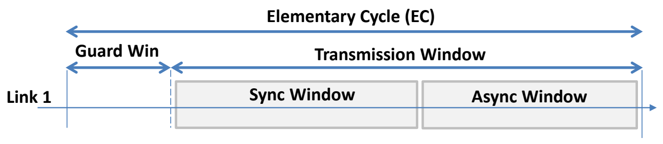


Fig. 6 The EC in the HaRTES architecture for two links.

The switch that is connected to the producer of a periodic message determines its transmission, making sure that it occurs within the associated window. The actual transmission is triggered by the switch with a Trigger Message (TM), transmitted within the Guard Window (see Fig. 6). The sporadic messages are transmitted whenever they are activated without waiting for the TM, yet within Asynchronous

Window. The arriving message in the switch is buffered in a priority queue. The switch forwards the message as long as there is enough time in the associated window within the current EC. Otherwise, it is kept in the queue for the next EC. The HaRTES architecture allows consistent ECs in all links by time synchronizing all nodes and switches. For more details the reader is referred to [7]. For the purpose of this model, we implement two servers per application per network resource (i.e., per link), one for Synchronous Window and the other for Asynchronous Window. It is important to dedicate a server per link for an application as each link in the network may share between several applications. Therefore, better tuning for the application resource allocation is possible. The reason behind using two servers per application per link is that the synchronous and asynchronous transmissions should be isolated to keep the fundamental feature of the HaRTES architecture. Therefore, if there are two messages in one application crossing a link and from different types, they are served by two different servers. If there is an application with one type of message (i.e., all messages are synchronous) only, then one server for the application per link is sufficient. Note that we may combine synchronous and asynchronous traffic that belong to the same application in one link to allocate one server for them. In such a case we loose the possibility of isolating the two types of traffic, while from the application perspective it is tolerated as both types belong to the same application.

3.7 Alternative network models

The end-to-end resource reservation model allows for alternative network technologies, provided these technologies support resource reservation mechanisms. In this section we discuss two network alternatives, Ethernet AVB [30] representing an Ethernet-based technology and CAN with server-based scheduling [48] representing a broadcast communication scheme.

The Ethernet AVB standard foresees up to 8 classes of traffic. Commonly, two classes of traffic, i.e., classes A and B, are being used in automotive and automation domains, where class A has higher priority than class B. Within each class the transmission of messages is based on a single FIFO queue per output port. According to the standard, each class of traffic has a separate reservation per output port that is enforced by a traffic shaper called the Credit-Based Shaper (CBS). The CBS is allocated a part of total bandwidth to each class using *credits*. While the class has zero or positive credit, it can access the port for transmission, otherwise it is has to wait for replenishment of the credit. The increasing and decreasing rate of the credit is constant which is defined by a designer for each port according to the reserved bandwidth. In this technology, the link and message are defined according to the network model discussed in Section 3.6. However, in Ethernet AVB, two priority levels can be used only for classes A and B. The resource reservation model can be applied to Ethernet AVB by assigning each application to separate class with a bandwidth reservation in each link. If two applications use the same class of traffic in one link, intuitively speaking, the applications loose their temporal isolation as they can interfere each other by using the same FIFO queue. Although the case of using Ethernet AVB introduces limitations on the number of applications per link, it can still be considered as an alternative technology for the proposed end-to-end reservation model.

In the server-scheduled CAN, there is one shared network resource corresponding to the CAN bus, which is in contrast to the Ethernet-based technologies with several network resources. According to the proposal presented in [48], multiple network servers (N-Servers) can be implemented on each node connected to the CAN bus. This concept resembles implementing multiple servers mediating access to a single resource (CAN bus). Each server is characterized by a period and it is allowed to send one message within the period. These servers are scheduled by a master server (M-Server) which is implemented in a particular node connected to the CAN bus. The M-Server is responsible for keeping track of the deadline that is defined for each N-Server. The M-Server is also responsible for allocating bandwidth for N-Servers. The M-Server further coordinates the access of N-Servers and provides isolations among them. We refer the reader to [48] for the details about the algorithms. The presented message model still holds for this network technology with a difference that $\mathcal{L}_{i,k}$ contains no element. The resource model also holds where the resources are only nodes. There is no need to define a server with period, budget and priority for the network because the CAN bus is controlled through the nodes via N-Servers.

3.8 End-to-end timing requirements

The end-to-end timing requirements on each transaction $\Gamma_i^{(h)}$ belonging to application $\mathcal{A}^{(h)}$ are specified as a set of constraints, denoted by $Cr_i^{(h)}$. These constraints include an end-to-end deadline, denoted by $D_i^{(h)}$; an age constraint, denoted by $Age_i^{(h)}$; and a reaction constraint, denoted by $Reac_i^{(h)}$. It is up to the user to specify one or more of these constraints on each transaction.

$$Cr_i^{(h)} = \{D_i^{(h)}, Age_i^{(h)}, Reac_i^{(h)}\} \tag{13}$$

3.9 System development model

We assume a system development model which involves the following two roles: (i) application developer; (ii) system integrator. Applications that are followed with the above model are developed by application developers. The application developers abstract the timing requirements of their application using the application interfaces (Eq. 12). The timing behavior of the applications, provided in an interface, is investigated using the analysis presented in Section 4. Designing interfaces, however, is addressed in Section 5. The system integrators, on the other hand, are responsible for integrating various applications while examining the timing correctness of the entire system. In this work we propose a compositional development model in the sense that the system integrators only need to use the application interfaces for investigating the timing properties of the system composed of several applications. The schedulability of the entire system can be done by a state of the art analysis given the interfaces by the application integrators.

3.10 Illustrative Example

Consider an example of a distributed system with two applications, $\mathcal{A}^{(1)}$ and $\mathcal{A}^{(2)}$. The system consists of two HaRTES switches that connect four nodes as shown in Fig. 7.

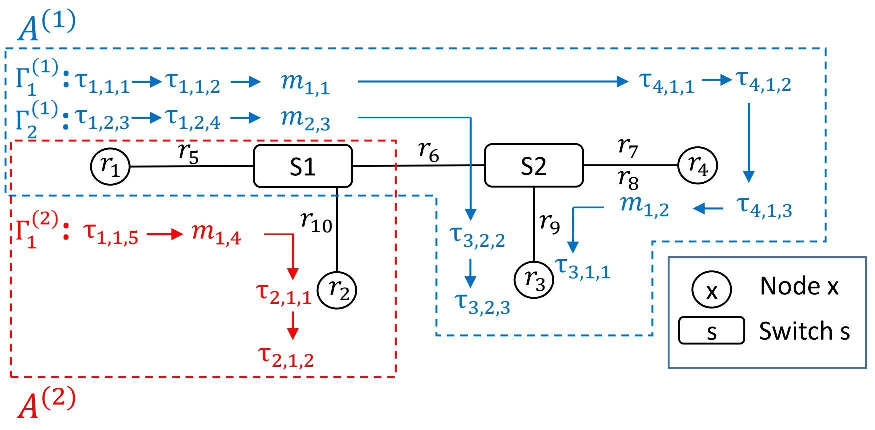

$A^{(1)}$
$\Gamma_1^{(1)}$: $\tau_{1,1,1}$ $\tau_{1,1,2}$ $m_{1,1}$ $\tau_{4,1,1}$ $\tau_{4,1,2}$
$\Gamma_2^{(1)}$: $\tau_{1,2,3}$ $\tau_{1,2,4}$ $m_{2,3}$
r_1 r_5 S1 r_6 S2 r_7 r_8 r_4
r_{10} r_9 $m_{1,2}$ $\tau_{4,1,3}$
$\Gamma_1^{(2)}$: $\tau_{1,1,5}$ $m_{1,4}$
$\tau_{3,2,2}$ $\tau_{3,2,3}$ r_3 $\tau_{3,1,1}$
$\tau_{2,1,1}$ r_2 $\tau_{2,1,2}$
x Node x
s Switch s
$A^{(2)}$

Fig. 7 An example to illustrate the system model.

There are two transactions in $\mathcal{A}^{(1)}$ and one transaction in $\mathcal{A}^{(2)}$. The transactions for the two applications are as follows.

$$\Gamma_1^{(1)} = \{\tau_{1,1,1}, \tau_{1,1,2}, m_{1,1}, \tau_{4,1,1}, \tau_{4,1,2}, \tau_{4,1,3}, m_{1,2}, \tau_{3,1,1}\}$$
$$\Gamma_2^{(1)} = \{\tau_{1,2,3}, \tau_{1,2,4}, m_{2,3}, \tau_{3,2,2}, \tau_{3,2,3}\}$$
$$\Gamma_1^{(2)} = \{\tau_{1,1,5}, m_{1,4}, \tau_{2,1,1}, \tau_{2,1,2}\}$$

In Fig. 7, the node resources are identified by r_1 to r_4, whereas the communication resources are identified by r_5 to r_{10}. Note that the resources r_7 and r_8 represent two directions of the network connection (i.e., one for each link). The two applications share resources in Node 1, denoted by r_1, and the link between Node 1 and Switch 1, denoted by r_5. The resources used by the two applications are represented as follows.

$R^{(1)} = \{r_1, r_3, r_4, r_5, r_6, r_7, r_8, r_9\}$
$R^{(2)} = \{r_1, r_2, r_5, r_{10}\}$

According to the reservation model, a reservation should be defined for each resource that an application uses. For instance, the reservation in resource r_4 for application $\mathcal{A}^{(1)}$ is $\{\Pi_4^{(1)}, \Theta_4^{(1)}, \Psi_4^{(1)}\}$. This means that a server should be implemented in Node 4 with a period of $\Pi_4^{(1)}$, a budget of $\Theta_4^{(1)}$ and a priority of $\Psi_4^{(1)}$. This server schedules three tasks, namely $\tau_{4,1,1}$, $\tau_{4,1,2}$ and $\tau_{4,1,3}$ as shown in Fig. 7. Since r_1 is shared between the two applications, two servers should be implemented in Node 1. The first server, with parameters $\{\Pi_1^{(1)}, \Theta_1^{(1)}, \Psi_1^{(1)}\}$, is responsible for scheduling the four tasks in $\mathcal{A}^{(1)}$, namely $\tau_{1,1,1}$, $\tau_{1,1,2}$, $\tau_{1,2,3}$ and $\tau_{1,2,4}$, that belong to Node 1. Whilst, the second server, with parameters $\{\Pi_1^{(2)}, \Theta_1^{(2)}, \Psi_1^{(2)}\}$, schedules the only task, namely $\tau_{1,1,5}$, in $\mathcal{A}^{(2)}$ that belongs to Node 1.

4 Timing Analysis

The proposed system model strictly isolates the applications by resource reservations and by restricting sharing of tasks and messages among the applications. Therefore, schedulability of an application can be evaluated independently of the other applications. An application is said to be schedulable if all of its transactions are schedulable. That is, the end-to-end response times, age and reaction delays in each transaction satisfy their corresponding constraints. In this section, first we present the response-time analyses for tasks and messages conforming to the model discussed in the previous section. In these analyses, we consider the implicit deadline model, i.e., the task and message deadlines are assumed to be equal to the corresponding periods. Then we build upon these analyses and present the end-to-end response-time and delay analyses with resource reservations.

4.1 Response time analysis of tasks

We adapt the analysis presented in [5] as it is compatible with the presented model, e.g., the analysis is based on the assumption that the release jitter and deadline of each task are equal to or less than the task period. The analysis is based on a *supply bound function (sbf)* and a *request bound function (rbf)*. The $sbf(t)$ is the minimum effective capacity that the resource provides within a time interval of $[0,t]$. On the other hand, the $rbf_i(t)$ is the maximum load generated by one task (τ_i) including the load of its higher priority tasks within the time interval of $[0,t]$. The earliest time where the $sbf(t)$ becomes equal to or larger than the $rbf_i(t)$ is the response time of τ_i. This method has been used in various models with different names for the supply and request bound functions.

The *sbf* for an application $\mathcal{A}^{(h)}$ in node j corresponding to the resource r_q is the lower bound on the availability of a server to handle the tasks belonging to the application. In order to sketch the lower bound, we must assume that the server is always available at the end of its period, except the first activation which is available at the beginning to make the worst case when the server is not available. Such a lower bound is depicted in Fig. 8. In the first period the server is available at the beginning of the period, while in the second period the server is available at the end of the period. Then, this patterns continues. Therefore, the *sbf* for each point in time has a value. As we assumed that the server is not available for almost two periods of the server, the supply bound function is zero.

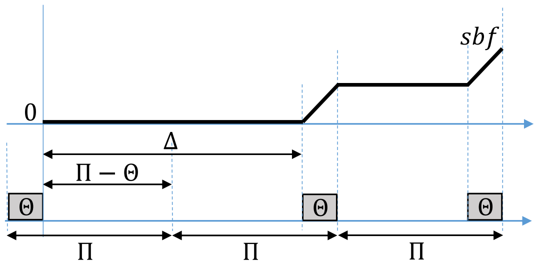


Fig. 8 The lower bound of the server availability.

Such a supply bound function, which is depicted in Fig. 8 is formulated in Eq. 14.

$$sbf_q^{(h)}(t) = \begin{cases} 0 & \text{if } t < \Delta \\ t - (\Delta + k \times (\Pi_q^{(h)} - \Theta_q^{(h)})) & \text{if } \Delta + k \times \Pi_q^{(h)} \le t < \Delta + k \times \Pi_q^{(h)} + \Theta_q^{(h)} \\ (k+1) \times \Theta_q^{(h)} & \text{if } \Delta + k \times \Pi_q^{(h)} + \Theta_q^{(h)} \le t < \Delta + (k+1) \times \Pi_q^{(h)} \end{cases} \tag{14}$$

where $k = \lfloor (t - \Delta)/\Pi_q^{(h)} \rfloor$; while Δ is the worst-case latency of the server which is $2 \times (\Pi_q^{(h)} - \Theta_q^{(h)})$.

The *rbf* for the task $\tau_{j,i,k}$ that belongs to the application $\mathcal{A}^{(h)}$ in node j is presented in Eq. 15. The tasks that are shared between transactions are excluded by checking the condition $(k \neq v)$, i.e., if the task identifiers are not the same. If the interfering task $\tau_{j,p,v}$ is triggered by its predecessor, the period of its predecessor is extracted from $V_{j,p,v}$. The extracted period is then used instead of $T_{j,p,v}$.

$$rbf_{j,i,k}^{(h)}(t) = C_{j,i,k} + \sum_{\substack{\forall \tau_{j,p,v} \in \mathcal{A}^{(h)} \\ \wedge\, k \neq v \wedge\, P_{j,p,v} \le P_{j,i,k}}} \left\lceil \frac{t + J_{j,p,v}}{T_{j,p,v}} \right\rceil C_{j,p,v} \tag{15}$$

The response time of the task $\tau_{j,i,k}$, denoted by $RT_{j,i,k}$, is derived from Eq. 16. The inequality should be evaluated for all t in the interval that is equal to the period of $\tau_{j,i,k}$.

$$RT_{j,i,k} = min(t > 0) : sbf_q^{(h)}(t) \geq rbf_{j,i,k}^{(h)}(t) \tag{16}$$

4.2 Response time analysis of messages

Among the existing analyses, the closest analysis that could be applied to our model is the response-time analysis for a single-switch architecture [52], which is based on Explicit Deadline Periodic (EDP) resource model [21]. However, the presented model considers that the messages traverse through multiple HaRTES switches with resource reservations. We present analysis for the multi-hop HaRTES architecture by adapting the existing analysis and using the *sbf* for the servers presented in [57]. A server is implemented for each network resource to schedule the messages that belong to one application, provided that the server is used during either Synchronous or Asynchronous Windows which depends upon the activation patterns of the messages. The scheduling in the network link is performed in a hierarchical fashion, where the EC and its two windows constitute the top level while the servers within the windows represent the second level in the hierarchy. We provide the *sbf* for a message $m_{i,k}$ belonging to transaction i in application $\mathcal{A}^{(h)}$ that crosses link l_q, which is denoted by $sbf_{i,k,q}^{(h)}(t)$. Note that in the analysis we consider that messages are not larger than the maximum Ethernet size, hence there is no fragmentation of messages.

Since the messages cannot be preempted during their transmission, a portion of the budget in the server can be wasted. We define this wasted time as the *idle time*. Fig. 9 shows a scenario in which a message cannot fit within the remaining budget in the first period of the server, hence it has to be sent after the replenishment during the next period. The idle time is upper bounded by the maximum size of the message in the set that includes $m_{i,k}$ and all the higher priority messages that cross the link l_q; belong to the same application; and have the same activation pattern as that of $m_{i,k}$. The idle time is denoted by $Id_{i,k,q}^{(h)}$ and is calculated in Eq. 17, where $AP(m_{i,k})$ presents the activation pattern for $m_{i,k}$.

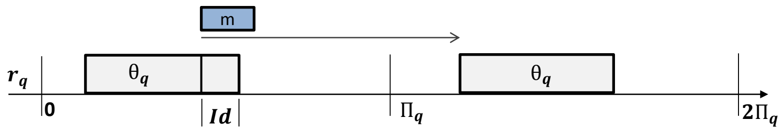


Fig. 9 Example demonstrating the inserted idle time.

$$Id_{i,k,q}^{(h)} = \max_{\substack{\forall m_{z,p} \in \mathcal{A}^{(h)} \wedge P_{z,p} \leq P_{i,k} \wedge l_q \in \mathcal{L}_{z,p} \\ \wedge p \neq k \wedge AP(m_{z,p}) = AP(m_{i,k})}} \{C_{z,p}\} \tag{17}$$

Therefore, in the calculation of the *sbf*, the idle time should be subtracted from the budget, i.e., $\Theta_q'^{(h)} = \Theta_q^{(h)} - Id_{i,k,q}^{(h)}$. The *sbf* can be computed as follows.

$$sbf_{i,k,q}^{(h)}(t) = \begin{cases} y.\Theta_q'^{(h)} + max\{t - x - y.\Pi_q^{(h)}, 0\} & \text{if } t \geq \Pi_q^{(h)} - \Theta_q'^{(h)} \\ 0 & \text{otherwise} \end{cases} \tag{18}$$

where $x = 2(\Pi_q^{(h)} - \Theta_q'^{(h)})$ and $y = \left\lfloor \frac{t-(\Pi_q^{(h)} - \theta_q'^{(h)})}{\Pi_q^{(h)}} \right\rfloor$.

The *rbf* is defined as the maximum load generated by a message with respect to its critical instant. The critical instant, in this case, corresponds to releasing the message with all its higher priority messages at the same time. The interference and blocking received by a message in the HaRTES architecture has been derived in [7]. These interferences for $m_{i,k}$ are categorized as follows: (i) the interference from the higher and equal priority messages that share the link with the message, denoted by $I_{i,k,q}^{(h)}$, and (ii) the blocking from the lower priority messages that share the link with the message, denoted by $B_{i,k,q}^{(h)}$. The *rbf* for the message $m_{i,k}$ belonging to the application $\mathcal{A}^{(h)}$ crossing the link l_q is presented in Eq. 19.

$$rbf_{i,k,q}^{(h)}(t) = C_{i,k} + I_{i,k,q}^{(h)} + B_{i,k,q}^{(h)} \tag{19}$$

The interference of higher or equal priority messages is calculated according to Eq. 20. Similar to the case of response-time analysis for tasks, the messages that are shared among transactions should be excluded by checking the message identifier, i.e., $p \neq k$.

$$I_{i,k,q}^{(h)} = \sum_{\substack{\forall m_{z,p} \in \mathcal{A}^{(h)} \land P_{z,p} \leq P_{i,k} \land p \neq k \\ \land l_q \in \mathcal{L}_{z,p} \land AP(m_{z,p}) = AP(m_{i,k})}} \left\lceil \frac{t}{T_{z,p}} \right\rceil C_{z,p} \tag{20}$$

When a message is ready to be forwarded in the switch, it might be blocked by a lower priority message that is already under transmission. The blocking delay is the maximum size of the message within the messages that belong to the same application as $m_{i,k}$ and cross the link l_q.

$$B_{i,k,q}^{(h)} = \max_{\substack{\forall m_{z,p} \in \mathcal{A}^{(h)} \land P_{z,p} > P_{i,k} \land p \neq k \\ \land l_q \in \mathcal{L}_{z,p} \land AP(m_{z,p}) = AP(m_{i,k})}} \{C_{z,p}\} \tag{21}$$

The response time of $m_{i,k}$ crossing the link l_q is computed in Eq. 22. The period of the message should be decomposed for each link. This can be done in proportion to the load of the links or by equally distributing the load over the links. We refer the reader to the existing works (e.g., [14]) for details about the decomposition of deadlines. Eq. 22 should be evaluated for all t until the decomposed deadline for link l_q.

$$RT_{i,k,q} = min(t > 0) : sbf^{(h)}_{i,k,q}(t) \geq rbf^{(h)}_{i,k,q}(t) \tag{22}$$

The response time of $m_{i,k}$ is derived in Eq. 23, where ε represents the switch fabric delay.

$$RT_{i,k} = \sum_{q=1}^{|\mathcal{L}_{i,k}|} RT_{i,k,q} + |\mathcal{L}_{i,k}| \times \varepsilon \tag{23}$$

4.3 Timing analysis of transactions

The end-to-end response time for a transaction is the sum of the response times of tasks and messages that belong to it. The transaction is schedulable if its end-to-end response time, denoted by $RT^{(h)}_i$, is less than or equal to its end-to-end deadline, denoted by $D^{(h)}_i$. The end-to-end response time for transaction $\Gamma^{(h)}_i$ is computed as follows.

$$RT^{(h)}_i = \sum_{\forall \tau_{j,i,k} \in \Gamma^{(h)}_i} RT_{j,i,k} + \sum_{\forall m_{i,k} \in \Gamma^{(h)}_i} RT_{i,k} \tag{24}$$

When the tasks and messages in a transaction have independent activation sources, the age and reaction delays should be computed. In order to calculate these delays, all *reachable* time paths for a transaction should be derived. For example, in Fig. 3 one reachable time path is τ_1 (first instance), τ_2 (second instance) and τ_3 (third instance). Note that the time paths that are not reachable are excluded from the analysis, e.g., τ_1 (second instance), τ_2 (second instance) and τ_3 (third instance). A set of Boolean functions are presented in [22] to derive such reachable time paths. However, time paths presented in [22] are only applicable for networks in which the messages cannot be independently initiated. In the HaRTES architecture a message may be triggered independent of the sender task in case the message is synchronous message in the HaRTES. Therefore, independent activations of the messages in the transaction should be included in the time paths. Fig. 10 shows a transaction with two tasks and a message. The message is triggered by the switch regardless of the sender task (τ_1). In this figure, five time paths (identified by A to E) are illustrated. However, not all of these time paths are reachable. For example, time path D is not reachable as the data cannot travel back in time.

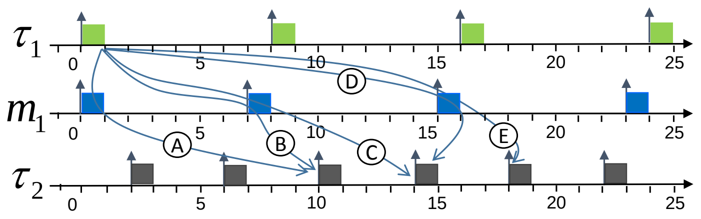


Fig. 10 An example of time paths including a message

The algorithm to compute the delays for $\Gamma_i^{(h)}$ is presented in Algorithm 1. In order to compute the longest time path, all reachable time paths are extracted based on the activation patterns of the tasks and messages in the transaction. The extraction is done by the two functions in lines 1 and 2. Basically, every valid time path is verified against the reachability condition. This condition for two neighboring tasks (or a task and a message) is as follows. Let $\alpha_w(i)$ and $\alpha_r(i)$ denote the activation times of the ith instances of the reader and writer tasks respectively. Also, let $RT_w(i)$ denotes the response time of the writer task.

1. The activation time of the writer should be earlier than the reader, i.e., $\alpha_r(i) \geq \alpha_w(i)$. For instance, time path D in Fig. 10 does not satisfy this condition between m_1 and τ_2.
2. The execution of the writer and reader should not overlap, i.e., $\alpha_r(i) \geq \alpha_w(i) + RT_w(i)$. For example, time path A in Fig. 10 does not satisfy this condition between τ_1 and m_1.
3. The writer and reader can have overlap only if both of them execute on the same node or link, and the priority of the reader is lower than the priority of the writer.
4. There could be a time path in which the output of an instance of the writer is over-written by its next instance. For example, time path E in which the second instance of τ_1 over-writes the data from its previous instance. Such time paths are excluded from the list of reachable time paths.

In order to compute the reaction delay, a subset of reachable time paths is extracted. In this subset, no time path exists that shares the same start instance of the first task in the transaction and has an earlier end instance of the last task in the transaction.

The function in line 5 of the algorithm provides the age delay of the transaction for a given time path. The function uses Eq. 25 to compute the age delay [22], where α_n and α_1 represent the activation times of the last task and the first task in the transaction. Note that a transaction cannot start or finish with a message.

$$New\_Age = \alpha_n(TP) + RT_{j,i,k} - \alpha_1(TP) \tag{25}$$

The reaction delay derived in line 11 of the algorithm for a given time path is computed using Eq. 26, where $Pred(TP)$ is the instance of the first task in the transaction that belongs to the previous reachable time path.

Algorithm 1 Find the age and reaction delays for $\Gamma_i^{(h)}$

```
1: $TP\_Age = FindAllValidAgeTimePaths(i)$
2: $TP\_Reac = FindAllValidReacTimePaths(i)$
3: $Age\_Delay_i^{(h)} = 0; Reac\_Delay_i^{(h)} = 0$
4: for all $TP\_Age$ do
5:     $New\_Age = ComputeAge(TP\_Age)$
6:     if $New\_Age > Age\_Delay_i^{(h)}$ then
7:         $Age\_Delay_i^{(h)} = New\_Age$
8:     end if
9: end for
10: for all $TP\_Reac$ do
11:     $New\_Reac = ComputeReac(TP\_Reac)$
12:     if $New\_Reac > Reac\_Delay_i^{(h)}$ then
13:         $Reac\_Delay_i^{(h)} = New\_Reac$
14:     end if
15: end for
16: return $Age\_Delay_i^{(h)}, Reac\_Delay_i^{(h)}$
```

$$New\_Reac = \alpha_n(TP) + RT_{j,i,k} - \alpha_1(Pred(TP)) \tag{26}$$

In Eq. 25 and Eq. 26, $RT_{j,i,k}$ represents the response time of the last task in $\Gamma_i^{(h)}$ which executes in node j. The transaction meets its constraints if:

$$Age\_Delay_i^{(h)} \leq Age_i^{(h)} \; \& \; Reac\_Delay_i^{(h)} \leq Reac_i^{(h)} \tag{27}$$

4.4 Schedulability of applications

In order to verify the schedulability of applications, the response time of servers corresponding to each application should be evaluated on both nodes and network links. The servers are schedulable if their response times are less than or equal to their respective periods. The response time of a server for $\mathcal{A}^{(h)}$ in a node corresponds to resource r_q. It is recursively calculated using Eq. 28.

$$RT_q^{(h)} = \sum_{\forall \mathcal{A}^{(f)} \in \mathcal{S} \wedge \Psi_q^{(f)} \leq \Psi_q^{(h)}} \left\lceil \frac{RT_q^{(h)}}{\Pi_q^{(f)}} \right\rceil \Theta_q^{(f)} \tag{28}$$

On the network links, the servers use a portion of bandwidth, i.e., in the Synchronous or Asynchronous Windows. Therefore, the analysis based on the *sbf* and *rbf* is used. The Synchronous Window becomes available every EC at a known point in time, which is always after the Guard Window. Fig.11 shows the availability of windows and the *sbf* for the Synchronous Window in link l_q, where L_{EC} is the size of the EC; and LS_q and LA_q are the sizes of the Synchronous and Asynchronous Windows in link l_q respectively. In order to calculate the supply for a given time interval t, we consider that the interval starts at the beginning of the EC as the resource is periodic in a fixed position within the EC. There are three scenarios for the time interval: (a) it finishes before the window, (b) it finishes within the window, and (c) it finishes before the end of the EC but consumes the whole window.

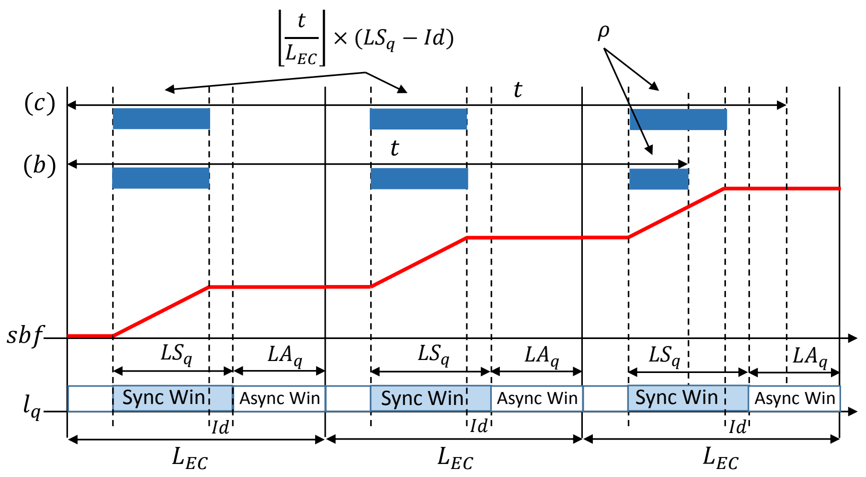


Fig. 11 Supply bound function for the Synchronous Window.

The time interval t in Fig. 11 is calculated using Eq. 29, where ρ is a part of the supply in the last EC and Id_q is the idle time. The calculations for the idle time in the servers will be explained later in this section.

$$sbf_q^{(h)}(t) = \left\lfloor \frac{t}{L_{EC}} \right\rfloor \times (LS_q - Id_q) + \rho \tag{29}$$

We compute the supply in the last EC for the scenarios (b) and (c) as depicted in Fig. 11. If the interval finishes in the middle of the Synchronous Window (i.e., scenario (b)), ρ is computed using Eq. 30. Note that this scenario also covers scenario (a) by using the *max* operation in Eq. 30.

$$\begin{aligned} &\rho = max\{t - \left\lfloor \tfrac{t}{L_{EC}} \right\rfloor \times L_{EC} - (L_{EC} - LS_q - LA_q), 0\}, \\ &\text{if } \left\lfloor \tfrac{t}{L_{EC}} \right\rfloor \times L_{EC} \leq t \leq \left\lfloor \tfrac{t}{L_{EC}} \right\rfloor \times L_{EC} + (L_{EC} - LA_q - Id_q) \end{aligned} \tag{30}$$

However, if the time interval finishes after the Synchronous Window but before the end of the EC (i.e., scenario (c)), the calculations for ρ are different as shown in Eq. 31.

$$\begin{aligned} &\rho = max\{t - \left\lfloor \tfrac{t}{L_{EC}} \right\rfloor \times L_{EC} - (L_{EC} - LS_q + Id_q), 0\}, \\ &\text{if } t > \left\lfloor \tfrac{t}{L_{EC}} \right\rfloor \times L_{EC} + (L_{EC} - LA_q - Id_q) \end{aligned} \tag{31}$$

The Asynchronous Window becomes periodically available at known point in time within the EC. Hence, the *sbf* for it is computed in a similar fashion as follows.

$$sbf_q^{(h)}(t) = \left\lfloor \frac{t}{L_{EC}} \right\rfloor \times (LA_q - Id_q) + \rho \tag{32}$$

The resource in the last EC (ρ) is computed for the scenarios using Eq. 33 and Eq.34.

$$\begin{aligned} &\rho = max\{t - \left\lfloor \tfrac{t}{L_{EC}} \right\rfloor \times L_{EC} - (L_{EC} - LA_q), 0\}, \\ &\text{if } \left\lfloor \tfrac{t}{L_{EC}} \right\rfloor \times L_{EC} \leq t \leq \left\lfloor \tfrac{t}{L_{EC}} \right\rfloor \times L_{EC} + (L_{EC} - Id_q) \end{aligned} \tag{33}$$

$$\begin{aligned} &\rho = max\{t - \left\lfloor \tfrac{t}{L_{EC}} \right\rfloor \times L_{EC} - (L_{EC} - LA_q + Id_q), 0\}, \\ &\text{if } t > \left\lfloor \tfrac{t}{L_{EC}} \right\rfloor \times L_{EC} + (L_{EC} - Id_q) \end{aligned} \tag{34}$$

In order to find the idle time, we derive the maximum message size that crosses the window in link l_q and belongs to $\mathcal{A}^{(h)}$. Eq. 35 calculates the idle time, where ω represents the periodic and sporadic activation pattern if the idle time is calculated in the Synchronous and Asynchronous Windows respectively.

$$Id_q = \max_{\substack{\forall m_{z,p} \in \mathcal{A}^{(h)} \wedge\, l_q \in \mathcal{L}_{z,p} \\ \wedge\, AP(m_{z,p}) = \omega}} \{C_{z,p}\} \tag{35}$$

The *rbf* for the servers within the transmission windows is calculated using Eq. 36, where the interference from the higher priority servers is accounted for.

$$rbf_q^{(h)}(t) = \sum_{\forall \mathcal{A}^{(f)} \in \mathcal{S} \wedge\, \Psi_q^{(f)} \leq \Psi_q^{(h)}} \left\lceil \frac{t}{\Pi_q^{(f)}} \right\rceil \Theta_q^{(f)} \tag{36}$$

Finally, the response time of the server is calculated when the *sbf* is equal to or larger than the *rbf*, as shown in Eq. 37.

$$RT_q^{(h)} = min(t > 0) : sbf_q^{(h)} \geq rbf_q^{(h)} \tag{37}$$

The system $\mathcal{S}$ is schedulable if all the servers in the nodes and network links are schedulable. Therefore, the applications should be schedulable in each node and network link. The schedulability of servers can be verified by the existing analysis depending on the scheduler algorithm for the global level scheduler. For instance, if the global level scheduler is the fixed-priority preemptive scheduler according to RM, the response time analysis corresponding to that is used.

5 Reservation Design

In this section we address the problem of designing application interfaces, i.e., the end-to-end reservations. The interface design is performed by application developers. Therefore, we consider the problem of designing an end-to-end reservation for one application at a time and, for notational convenience, we drop the application index in this section. The problem of interface design for an application involves designing reservation periods and budgets. The system integrator assigns the priorities of the reservations such that the reservations are schedulable in the global level. The priority assignment is out of the scope of this chapter.

A common practice in designing reservation for a single resource is to compute the bandwidth reservation in which the response times are equal to the corresponding deadlines. In the case of end-to-end resource reservations, however, the problem is more complex. It is not possible to compute each reservation separately because the end-to-end response times depend on all application reservations.

We model the end-to-end reservation design problem as a Constraint Satisfaction Problem (CSP) [37]. The reservation design using CSP formulation provides separation of concerns in the sense that the problem formulation is independent from the solving technique. To this end, we can use different solving techniques that are provided by the solver and evolved over the time to solve the same problem formulation. This approach is potentially complete and finds the optimum solution provided enough time.

A CSP is a triple of finite set of variables, values and constraints. A CSP solver searches for feasible variable assignments, i.e., a function from the variables to the values such that the constraints are satisfied. The CSP formulation allows considering all reservations simultaneously and designing an end-to-end reservation which is overall efficient. The solution for a CSP is the set of all assignments which satisfy all constraints. A CSP solver performs constraint propagation, branching and search in an intertwined manner. Propagation uses the constraints and removes infeasible values from the value set. For instance, in our case, constraint propagation removes budget values that result in deadline misses. Branching constructs a search tree based on the remaining values in the value set. The search selects a node from the tree that is constructed by branching to be explored. In the following we describe the variables, values and constraints in our CSP formulation.

5.1 Variables

The design problem is to select values for the reservation budgets Θ_j and reservation periods Π_j for all reservations corresponding to an application. Therefore, the variable set of the CSP is as follow:

$$\{\Theta_1, \dots, \Theta_q, \Pi_1, \dots, \Pi_q\}.$$

We define the bandwidth of a reservation as follows: $\beta_j = \frac{\Theta_j}{\Pi_j}$. The application footprint γ on the resources is the sum of all reservation bandwidths that belong to the application, i.e. $\gamma = \sum_{j \in R} \beta_j$.

5.2 Values and constraints

We assume that the system designer imposes the minimum reservation periods considering the context switch related overheads and the time resolution of the operating system. Also, we assume that the reservation periods should never exceed the shortest task/message period within the application since larger periods will require significanly higher budgets. Therefore, we have:

$$\forall j \in R \quad \Pi^{\mathtt{min}} \leq \Pi_j \leq P^{\mathtt{min}}, \tag{38}$$

where $\Pi^{\mathtt{min}}$ represents the minimum allowed period and $P^{\mathtt{min}}$ is the minimum value among all task/message periods within the application. We also assume that the reservation periods are integer multiples of $\Pi^{\mathtt{min}}$.

The reservation budgets cannot exceed the corresponding periods:

$$\forall j \in R \quad \Theta_j \leq \Pi_j. \tag{39}$$

The reservation bandwidth corresponding to the application on the processor resource r_j should be more than the utilization of the tasks assigned to r_j:

$$\beta_j \geq \sum_{\forall i,k} \frac{C_{j,i,k}}{T_{j,i,k}}. \tag{40}$$

We have a similar constraint for the network resource. However, in the case of network resource r_j, the corresponding reservation bandwidth β_j has to be more than or equal to the sum of the bandwidths of all the involved messages:

$$\beta_j \geq \sum_{\forall i,k \,|\, l_j \in \mathcal{L}_{i,k}} \frac{C_{i,k}}{T_{i,k}}. \tag{41}$$

It is also possible to impose an upper limit for the reservation bandwidth on a particular resource. For instance, when a new application is being developed on top of an existing system, the resources are already allocated to the existing applications. In this case, the new application can only use the remaining slacks on the resources to meet its timing requirements. Therefore, the system integrator may impose maximum allowed bandwidth on the resources for the new application based on the slacks.

The timing requirements of the applications must be respected, i.e., the corresponding deadlines of all transactions have to be respected:

$$\forall \Gamma_i \in \mathcal{A} \quad RT_i \leq D_i \tag{42a}$$
$$Age\_Delay_i \leq Age_i \tag{42b}$$
$$Reac\_Delay_i \leq Reac_i, \tag{42c}$$

where RT_i, Age_Delay_i and $Reac\_Delay_i$ represent the response time, age delay and reaction delay of transaction Γ_i within application $\mathcal{A}$.

5.3 *Optimization*

The CSP formulation allows to search for the optimal solution with respect to a criterion. In this case the solver uses the branch-and-bound algorithm to prune the search tree. Basically, instead of evaluating the entire search tree, the solver prunes the nodes that do not improve over the best solution found until the current stage of the search. In this work we consider the following four optimization cases.

5.3.1 Case 1: Minimum Footprint

Each application occupies a bandwidth (β_j) of the resources in our end-to-end reservation scheme. It is desirable to reserve minimum bandwidths for the applications to allow integration of more applications on the same underlying resources. Therefore, the optimization objective is:

$$\begin{aligned} \text{Minimize:} \quad & \gamma = \sum_{j \in R} \beta_j, \\ \text{Subject to:} \quad & (38)\ (39)\ (40)\ (41)\ (42a)\ (42b)\ (42c). \end{aligned}$$

In this case, we assume that the system integrator has not imposed any upper bound on the allowed reservation bandwidths while the application under analysis has deadlines for response times, age delays and reaction delays. Then, the objective is to minimize the application footprint.

5.3.2 Case 2: Best Performance

In this case the system integrator has imposed an upper bound on the application footprint γ and the application designer is interested in finding a design in which the application performance is maximized, i.e., the response times, age delays or reaction delays are minimized. For instance, if the response times are of interest, the optimization objective is as follows:

$$\begin{aligned} \text{Minimize:} \quad & \sum_{\forall \Gamma_i \in \mathcal{A}} RT_i, \\ \text{Subject to:} \quad & (38)\ (39)\ (40)\ (41)\ (42b)\ (42c)\ , \\ & \gamma \leq \text{maximum allowed bandwidth.} \end{aligned}$$

When multiple objectives are of interest for optimizations, then we combine the criteria in one objective function and optimize based on a new criterion which is a weighted sum of all the criteria.

5.3.3 Case 3: Minimum Overhead

Another possible target is to minimize the overhead imposed by the reservations management mechanism in processor resources, namely their periodic activation and associated context switching. In such cases we can optimize for the following objective:

$$\begin{aligned} \text{Maximize:} \quad & \sum_{\forall j \in R_{cpu}} \Pi_j, \\ \text{Subject to:} \quad & (38)\ (39)\ (40)\ (41)\ (42a)\ (42b)\ (42c), \end{aligned}$$

where, R_{cpu} is the set of all processor resources of the application.

5.3.4 Case 4: Combined Footprint and Performance

Often multiple objectives are of interest when designing an end-to-end reservation for an application. In such cases, we use a linear combination of different criteria as the optimization objective. For instance, the application footprint and performance are two important yet contradicting objectives since larger footprints usually result in designs with better performance than small footprint. Therefore, the goal is to combine these two objectives and search for designs which have smaller footprints as well as good performance.

$$\begin{aligned} \text{Minimize:} \quad & w_\gamma \gamma + \sum_{\forall \Gamma_i \in \mathcal{A}} w_i RT_i, \\ \text{Subject to:} \quad & (38)\ (39)\ (40)\ (41)\ (42a)\ (42b)\ (42c), \end{aligned}$$

where w_γ and w_i are the weights associated to the footprint and response time objectives respectively. These weights are used to adjust the trade off among different objectives.

5.4 Design tool

We have developed an open-source C++ exploration tool which implements the above CSP formulation[6]. The set of tasks and messages as well as the set of transactions are provided as inputs to the tool in XML format. The users can select the optimization criterion by specifying it in a setting file. We have used the Gecode toolkit [54] for modeling and solving the CSP. The analysis presented in Section 4 should be encoded in our constraint model, however, this analysis is very complex and cannot be expressed using the standard constraints provided by Gecode. To this end, we have implemented a new constraint propagator which implements the schedulability analysis.

6 End-to-end Resource Reservations in RCM

In this section, we discuss how an existing component model can be extended to support the proposed end-to-end resource reservation model. As a proof of concept, we select RCM. First, we discuss what is missing in RCM to support the modeling of end-to-end resource reservations as shown in Fig. 12. Then we describe the extensions in RCM.

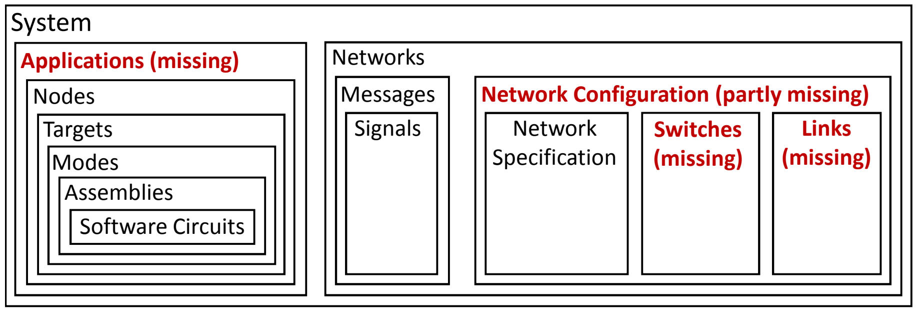


Fig. 12 Structural hierarchy in RCM along with missing elements.

6.1 Extending the structural hierarchy in RCM

The highest-level hierarchical element in the structural hierarchy of RCM is the system model that contains the models of nodes and networks. The node contains one or more targets. A target is a hardware and operating system specific instance of a

[6] The tool is available at: https://github.com/nimazad/e2e-res

node. It defines the run-time environment. Several targets can be assigned to a single node such as the PowerPC processor, ARM processor and simulation target. Each target contains one or more modes that define different states of the system, e.g., start-up mode and running mode. A mode may contain one or more assemblies. An assembly is a container to encapsulate software circuits (synonymous to software components). The network model contains models of messages and network configuration. Each message contains one or more signals that are mapped to the message. The network configuration contains the network specification object that specifies both protocol-dependent and protocol-independent information. We refer the reader to [47] for details about the network specification.

The structural hierarchy in RCM contains most of the elements that are required to support resource reservations in the software architecture. However, the model of the application is missing from the RCM hierarchy. We introduce a new object in RCM to represent the model of an application as shown in Fig. 13. This object is introduced as the second-highest hierarchical element in RCM. Note that the network object in RCM is also the second-highest hierarchical object. There are only two properties associated to this object, i.e., a name and an unique identifier.

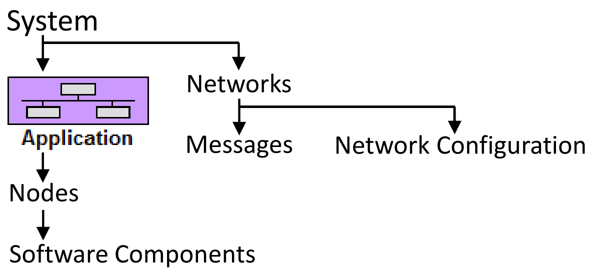
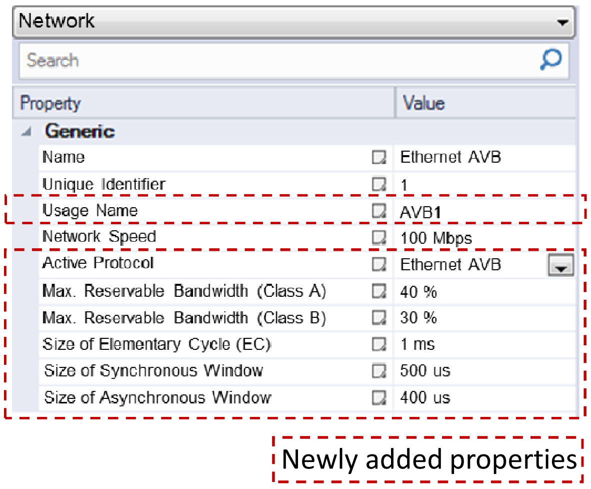


Fig. 13 Application model in RCM.

In RCM, the nodes are assumed to have unique identifiers. However, with the addition of the application model, this assumption does not remain valid because several applications can share the same node. In order to be consistent with the RCM approach that identifies objects with unique identifiers, we still assign a unique identifier to each node in the system. However, we add a new property to the node model, denoted by “Usage Name” in Fig. 16. For example, consider two applications. The first application contains two nodes with identifiers 1 and 2; whereas, the second application contains three nodes with identifiers 3, 4 and 5. If node 1 and node 3 have the same usage name then these two nodes correspond to one physical node which is shared by the two applications. This allows the applications to be developed in isolation and independent of each other, despite sharing resources.

6.2 Augmenting the network model with resource reservation

In order to specify resource reservations at the network level, the existing network model in RCM is extended by augmenting it with several new properties as shown in Fig. 14. Since a network can be shared by two or more applications, each application is provided with a copy of the network. The application is not allowed to modify global properties of the network, e.g., network speed and maximum reservable bandwidth. Hence, we add the usage name property to the network model. We also add two new protocols, namely Ethernet AVB and HaRTES, to the "Active Protocol" property. Two more properties are added to reserve the maximum bandwidth of the network for class A and class B traffic in AVB. Another new property corresponds to the size of the elementary cycle. In addition, two properties are added that represent sizes of the synchronous and asynchronous windows in the elementary cycle. The last three properties in Fig. 14 are specific to the HaRTES protocol. The last five properties in Fig. 14 are not used by CAN and its higher-level protocols. Hence, these properties are disabled by the development environment if such protocols are selected. Similarly, the two properties for bandwidth reservation in class A and class B traffic are disabled if HaRTES is selected. Whereas, the last three properties are disabled if Ethernet AVB is selected.

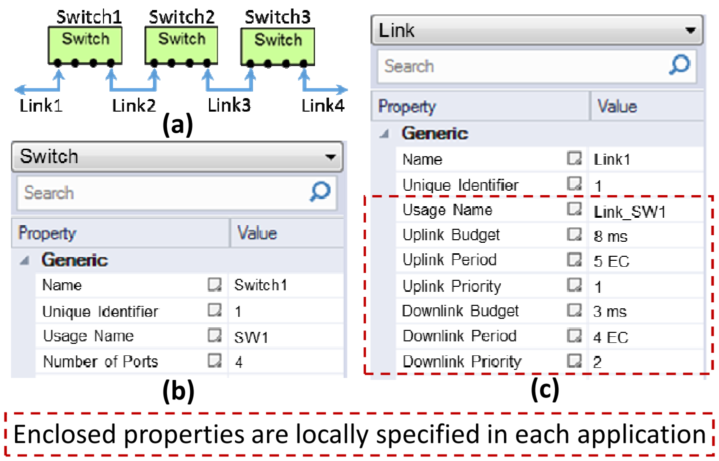


Property	Value
Name	Ethernet AVB
Unique Identifier	1
Usage Name	AVB1
Network Speed	100 Mbps
Active Protocol	Ethernet AVB
Max. Reservable Bandwidth (Class A)	40 %
Max. Reservable Bandwidth (Class B)	30 %
Size of Elementary Cycle (EC)	1 ms
Size of Synchronous Window	500 us
Size of Asynchronous Window	400 us

Fig. 14 Newly added properties to the network model in RCM.

6.3 Extending the network configuration

The network configuration object that is required to support the modeling of resource reservations is partly available in RCM as indicated in Fig. 12. In order to enable the modeling of Ethernet networks in RCM, we extend the network configuration object by introducing the models of a switch and a link as shown in Fig. 15(a). The figure shows internal architecture of a switched Ethernet network where three switches are connected via links. The properties of the switch model include “Name”, “Unique Identifier”, “Usage Name” and “Number of Ports” as depicted in Fig. 15(b). Since a switch can be shared among applications, the model of the shared switch in each application must have the same usage name regardless of its other properties. A switch can have any number of ports (typically, between 2 and 32 ports).



Property	Value
Generic	
Name	Switch1
Unique Identifier	1
Usage Name	SW1
Number of Ports	4

Property	Value
Generic	
Name	Link1
Unique Identifier	1
Usage Name	Link_SW1
Uplink Budget	8 ms
Uplink Period	5 EC
Uplink Priority	1
Downlink Budget	3 ms
Downlink Period	4 EC
Downlink Priority	2

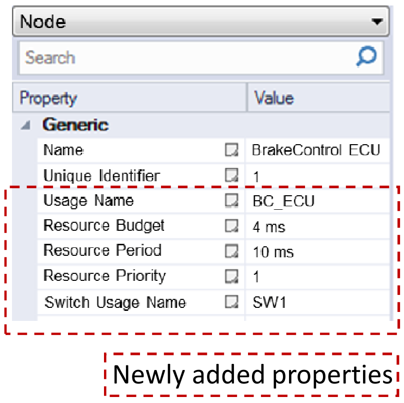
Enclosed properties are locally specified in each application

Fig. 15 (a) Models of switches and links in RCM, (b) properties of the switch model and (c) properties of the link model.

Four link models are shown in Fig. 15(a). Note that these links are entirely different from the legacy connecters in RCM that connect any two software circuits. The main difference between the link and a connecter is that the user-defined properties are associated to the model of the link and not to the connector. Note that each of the two links, denoted by Link1 and Link4 in Fig. 15(a), is located between a switch and a node. The connections between the node and the switch via the link are established by specifying the “Switch User Name” in the last property of the node shown in Fig. 16. Most of the switched Ethernet protocols support full-duplex communication. Hence, each link supports independent communication in the opposite directions, denoted by the uplink and downlink. The uplink corresponds to the transmission of data from the switch. The downlink corresponds to the reception of data at the switch. A link has a name, a unique identifier and a usage name. In order to specify reservation parameters for the servers that schedule traffic in the uplink and downlink, the link model includes the properties corresponding to the budget, period and priority of the servers. These reservation parameters can be provided by the system architects and integrators. In such a case, these parameters serve as requirements for the application. The developer of the application can also be asked by the system integrators to provide optimized reservation parameters. The resources are reserved on uplink and downlink in each link model separately and independently. Intuitively, an uplink may have a different reservation than the downlink on the same link within one application.

6.4 Augmenting the node model with resource reservations

The existing node model in RCM is augmented with new properties to support the specification of resource reservations for each application as shown in Fig. 16. The "Usage Name" property is already discussed in Section 6.1. In order to specify reservation parameters for the server that schedules the run-time entities (corresponding to the software circuits) on the node, three new properties are augmented with the node model. These properties include "Resource Budget", "Resource Period" and "Resource Priority". The last property in Fig. 16, denoted by "Switch Usage Name", corresponds to the usage name of the switch that is connected to the node in the case when one of the switched Ethernet protocols is selected in the network model. In the case of CAN and its higher-level protocols, this property is disabled by the development environment because these protocols do not use switches. It is important to note that all these properties including resource reservations are specified on each node separately within each application that uses it.



Property	Value
Name	BrakeControl ECU
Unique Identifier	1
Usage Name	BC_ECU
Resource Budget	4 ms
Resource Period	10 ms
Resource Priority	1
Switch Usage Name	SW1

Fig. 16 Newly added properties to the node model in RCM.

7 Extracting Execution Model from the Software Architecture

In this section, we describe a technique to extract the execution model from the software architecture with resource reservations. In this regard, first we model software architecture of a vehicular distributed embedded system using the extensions in RCM. In the second step, we extract the execution model from the software architecture.

7.1 Modeling of a vehicle system with resource reservations

In order to focus on the model extraction and for the sake of simplicity, we model partial architecture of the infotainment system. The system consists of four nodes and one network that implements the HaRTES protocol. For the sake of a proof of concept, we have selected HaRTES instead of AVB because HaRTES has higher flexibility and better support for resource reservations [53]. In HaRTES, resources in each link of the architecture can be reserved for a set of messages using a hierarchical scheduling framework, that is implemented for each link in the HaRTES switch. This allows better adoption of the presented model compared to other switched Ethernet-based technologies. However, AVB can also be used as it provides ability to reserve resources for each class of messages.

The reason for not selecting CAN is that it does not inherently support resource reservations unless traffic shapers are explicitly used. There are two applications that share the node and network resources in the system. The models of the system, applications and network are shown in Fig. 17. There are four messages in the network model; three of them belong to Application1, whereas the forth message (Msg4) belongs to Applciation2. The network contains two HaRTES switches and five links as shown in Fig. 18. The reservation parameters specified on the network model are as follows: size of Elementary Cycle is 1 ms; size of Synchronous Window is 700 μs; and size of Asynchronous Window is zero (since all messages are periodic).

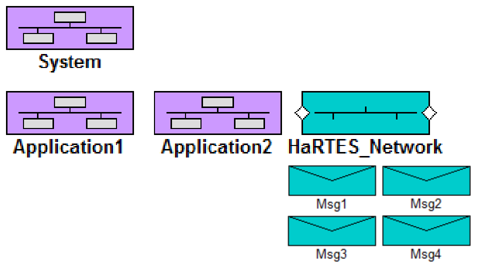


Fig. 17 Models of the system with two applications.

The software architectures of the two applications are shown in Fig. 19 and Fig. 20. Each application gets the copy of the network model. However, the application can access only its own messages. Hence, Msg1, Msg2 and Msg3 are shown in Fig. 19. Whereas, only Msg4 is shown in Fig. 20. Application1 is modeled with three nodes; whereas, application2 is modeled with two nodes. The resource reservation information is assumed to be provided by the system integrator. The two applications share Sensor_ECU node, Link1 and Switch1. Therefore, the usage name for the Sensor_ECU node is the same in both applications. This means, the software architecture of the two node models in these applications will be deployed on one ECU. As a result, the two node models can be developed independently due to resource reservations in the applications. The resources used by the two applications and the reservation parameters for each resource are tabulated in Fig. 21. Note that only Link3 uses full duplex communication in Application1 as depicted in Fig. 18. Hence, reservation parameters for uplink and downlink for Link3 are reserved separately. The software architectures of all nodes in the two applications are shown in Fig. 22 and Fig. 23.

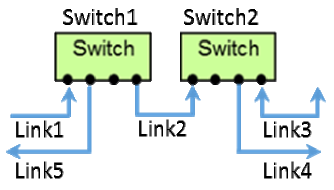


Fig. 18 Models of switches and links in the network.

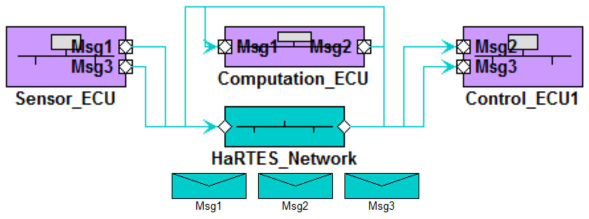


Fig. 19 Software architecture of Application1.

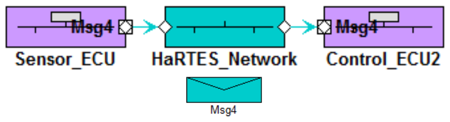
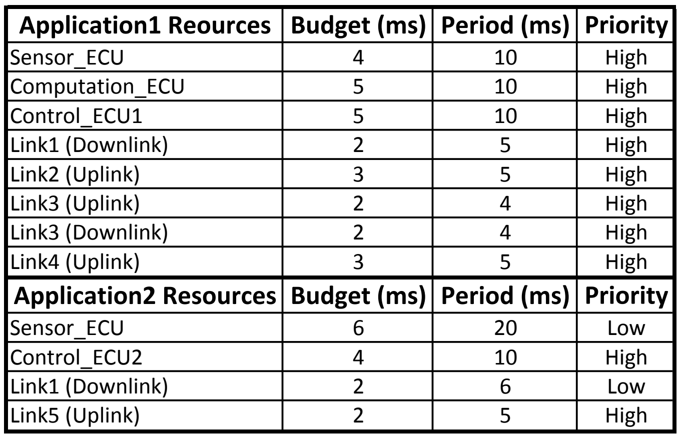


Fig. 20 Software architecture of Application2.



Application1 Reources	Budget (ms)	Period (ms)	Priority
Sensor_ECU	4	10	High
Computation_ECU	5	10	High
Control_ECU1	5	10	High
Link1 (Downlink)	2	5	High
Link2 (Uplink)	3	5	High
Link3 (Uplink)	2	4	High
Link3 (Downlink)	2	4	High
Link4 (Uplink)	3	5	High
Application2 Resources	**Budget (ms)**	**Period (ms)**	**Priority**
Sensor_ECU	6	20	Low
Control_ECU2	4	10	High
Link1 (Downlink)	2	6	Low
Link5 (Uplink)	2	5	High

Fig. 21 Resource reservation parameters for Application1 and Aplication2.

7.2 Extraction of execution model

The execution model extracted from the software architecture of the infotainment system is shown in Fig. 7. The applications in the execution model, presented in previous subsection, are centered around the concept of transactions. The transactions are extracted from the component chains in the software architectures shown in Fig. 22 and Fig. 23. Two transactions, denoted by $\Gamma_1^{(1)}$ and $\Gamma_2^{(1)}$ in Fig. 7, are extracted from Application1. Whereas, one transaction, denoted by $\Gamma_1^{(2)}$ in Fig. 7, is extracted from Apllication2. The translation considers a one-to-one mapping between a software circuit and a task, e.g., $\tau_{1,1,1}$ in Fig. 7 maps to SWC1 in the software architecture of Sensor_ECU in Fig. 22. The translation assigns identifier 1 to Sensor_ECU; identifier 2 to Control_ECU2; identifier 3 to Control_ECU1; and identifier 4 to Computation_ECU. The extracted transactions are as follows.

$$\Gamma_1^{(1)} = \{\tau_{1,1,1}, \tau_{1,1,2}, m_{1,1}, \tau_{4,1,1}, \tau_{4,1,2}, \tau_{4,1,3}, m_{1,2}, \tau_{3,1,1}\}$$

$$\Gamma_2^{(1)} = \{\tau_{1,2,3}, \tau_{1,2,4}, m_{2,3}, \tau_{3,2,2}, \tau_{3,2,3}\}$$

$$\Gamma_1^{(2)} = \{\tau_{1,1,5}, m_{1,4}, \tau_{2,1,1}, \tau_{2,1,2}\}$$

In Fig. 7, the node resources are identified by r_1, r_2, r_3, r_4 which represent Sensor_ECU, Control_ECU2, Control_ECU1 and Computation_ECU respectively. The communication resources are identified by r_5, r_6, r_7, r_8, r_9 and r_{10} which represent Link1 (downlink), Link2 (uplink), Link3 (uplink), Link3 (downlink), Link4 (uplink) and Link5 (uplink) respectively. $\mathcal{A}^{(1)}$ and $\mathcal{A}^{(2)}$ share the resources r_1 and r_5. The resources used by the two applications are represented as follows.

$$R^{(1)} = \{r_1, r_3, r_4, r_5, r_6, r_7, r_8, r_9\}$$

$$R^{(2)} = \{r_1, r_2, r_5, r_{10}\}$$

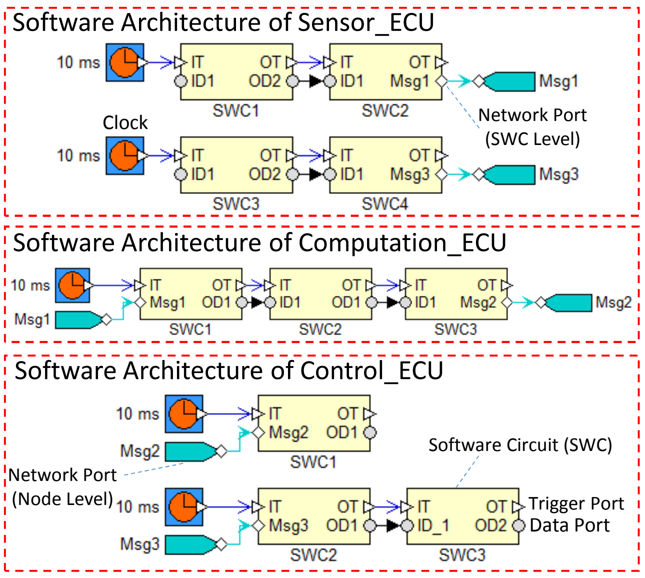


Fig. 22 Software architectures of all nodes in Application1.

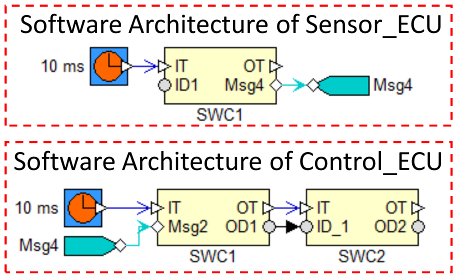

Software Architecture of Sensor_ECU
10 ms
IT
OT
ID1
Msg4
SWC1
Msg4
Software Architecture of Control_ECU
10 ms
Msg4
IT
OT
Msg2
OD1
SWC1
IT
OT
ID_1
OD2
SWC2

Fig. 23 Software architectures of all nodes in Application2.

Using the above-mentioned mapping and the table in Fig. 21, a deferable server is assigned to schedule an application in each resource. For instance, the reservation for resource r_4, represented by $\{\Pi_4^{(1)}, \Theta_4^{(1)}, \Psi_4^{(1)}\}$, is equal to $\{10, 5, High\}$. This means that a deferrable server should be implemented in the Computation_ECU with a period of 10 ms, a budget of 5 ms and high priority. This server schedules three tasks, namely $\tau_{4,1,1}$, $\tau_{4,1,2}$ and $\tau_{4,1,3}$ as shown in Fig. 7. Since r_1 is shared between the two applications, two servers should be implemented in the Sensor_ECU. The first server, with parameters $\{\Pi_1^{(1)}, \Theta_1^{(1)}, \Psi_1^{(1)}\}$ equal to $\{10, 4, High\}$, is responsible for scheduling the four tasks in $\mathcal{A}^{(1)}$, namely $\tau_{1,1,1}$, $\tau_{1,1,2}$, $\tau_{1,1,3}$ and $\tau_{1,1,4}$, that belong to the Sensor_ECU. Whereas, the second server, with parameters $\{\Pi_1^{(2)}, \Theta_1^{(2)}, \Psi_1^{(2)}\}$ equal to $\{20, 6, Low\}$, schedules the only task, namely $\tau_{1,1,5}$, in $\mathcal{A}^{(2)}$ that belongs to the Sensor_ECU.

8 Case Study and Evaluation

In order to show the applicability of the proposed model, resource reservation design method and end-to-end timing analysis we present a case study from the vehicular systems domain. We consider an Autonomous Steering Control (ASC) system, that provides electronic steer control to a vehicle using mechanical and electronic components. The ASC system uses an Ethernet network for communication as the backbone network in the vehicle. The architecture of the ASC system is depicted in Fig. 24. The ASC system is distributed over six Electronic Control Units (ECUs) and two HaRTES switches. There are four Wheel Control (WC) ECUs; one Steer Control (SC) ECU; and one Collision Avoidance Control (CAC) ECU. Moreover, there is a camera (CAM) that is mounted in front of the vehicle to gather the video frames for collision avoidance purpose.

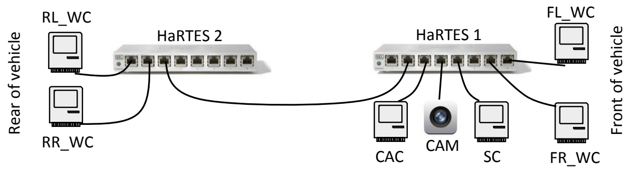


Fig. 24 The network architecture for the ASC system.

The ASC system is divided into two applications: (i) the Collision Avoidance (CA) application, and (ii) a conventional steering control, known as Steer-By-Wire (SBW) application. Each application together with its transactions is described in the next subsection. It should be noted that the parameters for both applications are inspired by an industrial use case from our industrial partners. All experiments related to the reservation designs are performed on an Intel Core i7-5500U CPU clocked at 2.4GHz with two cores and 8GB memory.

8.1 Collision avoidance application

The main function of this application is to detect obstacles by means of information from a radar and video frames from a camera to calculate a new steering angle accordingly. This information is collected in the CAC ECU. The CAC ECU sends this information in an Ethernet message to the SC ECU which is responsible for computing the steer actuator signals accordingly. The transactions belonging to this application are illustrated in Fig. 25. In the above transactions, the WCETs of the tasks are selected among $100\mu s$, $200\mu s$ and $400\mu s$ depending on the task function. The notation below each task and message shows its activation pattern, which is independent (denoted by I) and periodic (denoted by P) with period of 40ms in all tasks and messages. The execution time of the tasks and the size of messages are also denoted below each task. The message payload size for the control and radar signals is 64 bytes, whereas the message payload size for the video frame is 1500 bytes. For these transactions, the end-to-end deadline is 100 ms, while the age and reaction constraints are 110 ms and 150 ms, respectively.

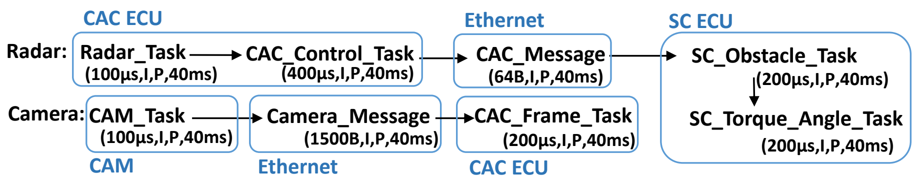


Fig. 25 Transactions in the CA application.

8.1.1 Designing reservations

In the following we design different end-to-end reservations for the collision avoidance application corresponding to different optimization cases presented in Section 5.

Minimum Footprint. The search for the minimum footprint design for the collision avoidance application completes in 1.5 s. This design is presented in Table 1. This design uses 7% of the system resources since each resource uses 1%. The tool sets the minimum allowed periods (1 ms) to all reservation periods. The performance of this design is presented in Table 2 which shows that all the timing constraints, specified on the transactions, are satisfied.

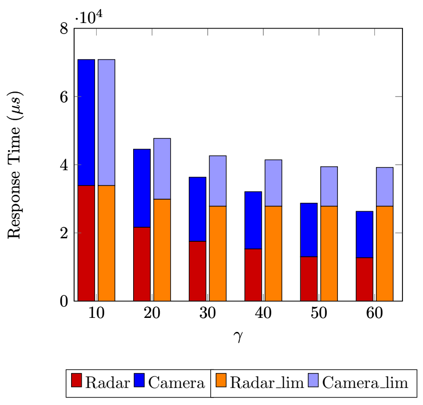
	CAC ECU	SC ECU	CAM	CAC ECU	l_j
α_j	1%	1%	1%	1%	1%
Θ_j	10 μs	10 μs	10 μs	10 μs	10 μs
Π_j	1 ms	1 ms	1 ms	1 ms	1 ms

Table 1 Minimum footprint design for the collision avoidance application.

	RT	Age Delay	Reaction Delay
Radar	53940 μs	90990 μs	130990 μs
Camera	47960 μs	50990 μs	90990 μs

Table 2 Performance in minimum footprint design for the collision avoidance application.

Best Performance. We limit the overall footprint of the application to different values from 10% to 60% and run the tool six times to obtain the designs with the best response times. Note that the optimization criterion in this case is to minimize the sum of the response times of the two transactions within the application. Fig. 26 shows the sum of the response times against the maximum allowed footprint. The figure shows that the reduction in response times is more significant when increasing γ from 10% to 20% than increasing it to the values that are above 20%. Also, we have observed that by providing 8.5 times more resources (from 7% to 60%) we can achieve around 74% better performance in terms of response times. The age delays and reaction delays also follow a similar trend. In another experiment, we have modified the minimum allowed period for the reservations ($\Pi^{\mathtt{min}}$), which sets the time resolution of the reservation periods, and searches for the best performance design. The maximum allowed footprint is set to 20% in these cases. The results are depicted in Fig. 26. The figure shows that larger $\Pi^{\mathtt{min}}$ results in worse performance. In all of the above experiments we have limited the search time to 30 minutes. Note that in all of the cases, the tool finds the first design in less than two seconds. A better design is found a few seconds after the first design in some cases, while the attempt to improve the first two designs in the rest of the exploration time fails in most cases.



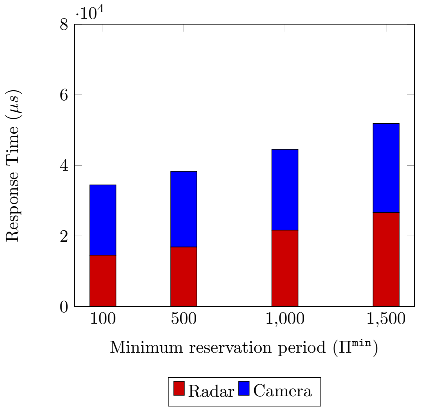


Fig. 26 Sum of the transaction response times against (i) the overall footprint where $\Pi^{\mathtt{min}} = 1$ ms (left figure) and (ii) the minimum reservation periods where $\gamma = 20\%$ (right figure).

So far we have only imposed a limit on the total application footprint. In the case where an application is being integrated to a system which already hosts other applications, we may have limited availability of particular resources. We perform a new experiment to evaluate the reservation design tool. In this experiment we assume that there are only 2% of the resources that are available to CAC ECU and SC ECU. We repeat the search for best response times assuming different total footprints and considering the above constraints on CAC ECU and SC ECU. The results are depicted in Fig. 26 using the captions Radar_lim and Camera_lim. Increasing the total allowed footprint from 30% to 60% does not have significant effect on the response times as availability of the two resources is limited.

Minimum Overhead. The search for the minimum overhead design with $\gamma = 7$ and 30 minutes timeout results in the following reservation periods: $\Pi_{\mathtt{CAC\ ECU}} = 10$ms, $\Pi_{\mathtt{SC\ ECU}} = 15$ms and $\Pi_{\mathtt{CAM}} = 20$ms. This set of end-to-end reservations imposes 15 times less overhead on the processor resources than the above designs. On the other hand, the performance of this design is around 44% worse than the performance of the minimum footprint design that is reported in Table 1.

Combined Footprint and Response Times. In another experiment, we search for the optimal design with respect to the combined footprint and response times. In this experiment, we assume $w_\gamma = 1000$ and $w_i = 1$. This weight assignment indicates that we consider 1% footprint improvement to be equivalent to 1 second improvement in response time. Similar to the previous cases we use 30 minutes timeout. Fig. 27 shows the quality of obtained design against the amount of time spent on searching. The figure shows that the improvements in the first few seconds are more significant than the rest of the search duration. Note that the x-axis is in the logarithmic scale. The final design is found after around 9 minutes and it uses 20% of the system resources, i.e. $\gamma = 20\%$.

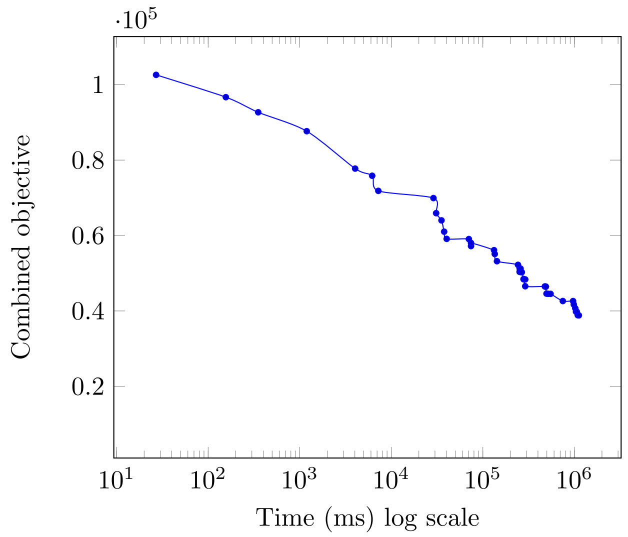


Fig. 27 Evolution of the design's objective value against search time.

8.2 Steer-by-Wire application

Each WC ECU gathers the wheel information including the angle and torque of the wheel. This information is sent to the SC ECU via an Ethernet message. Apart from this message, the SC ECU receives several sensor values including steering angle, steering torque and vehicle speed. Based on this information, the SC ECU provides feedback on steering torque to the feedback torque actuator. This actuator provides the feeling of turning effect for the driver. Moreover, the SC ECU sends an Ethernet message to each WC ECU. This message includes the steer angle and torque signals which are used by the WC ECUs to control the wheels actuators. For the sake of clarity and better readability, we only show the transactions for one wheel, i.e., rear-left (RL) wheel. Two transactions from the RL_WC ECU to the SC ECU are depicted in Fig. 28. Note that the transactions share the RL_WC_Controller_Task (in the RL_WC ECU), RL_WC_Message (in the network) and SC_Steer_Control_Task (in the SC_ECU). Similarly to the previous application, the notation below each task and message shows its activation pattern, execution time of tasks and size of messages, e.g., (100μs, I, P, 40ms) shows that the task is periodically activated by an independent source with a period of 40ms and execution time of 100μs.

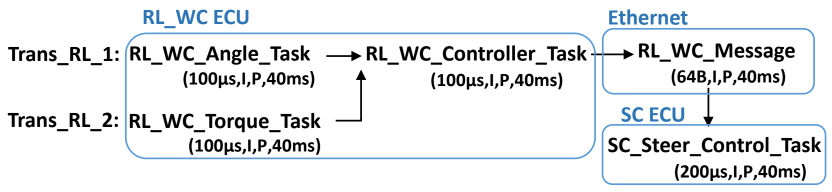


Fig. 28 Transactions from the rear-left WC ECU to the SC ECU.

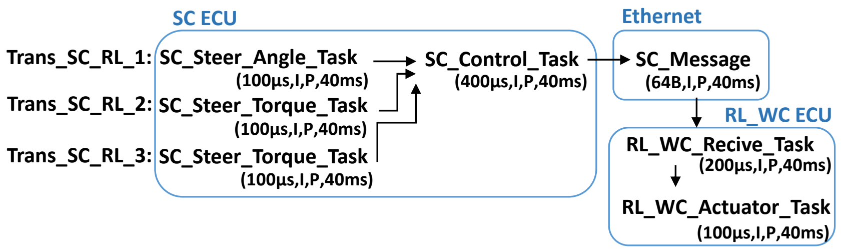


Fig. 29 Transactions from the SC ECU to the rear-left WC ECU.

In addition, there are three transactions to send information from the SC ECU to WC ECU. These transactions also share tasks and messages which are illustrated in Fig. 29. Since there are five transactions for each wheel, the total number of transactions in the SBW application is equal to 20. The constraints specified on the transactions include the deadline (100 ms), age constraint (100 ms) and reaction constraint (140 ms).

8.2.1 Designing reservations

The search for minimum footprint reservations for the steer-by-wire application ends after 126 s. The search time for the steer-by-wire application is significantly larger as compared to the collision avoidance application because of large size of the former application (20 transactions). The result of the minimum footprint design is presented in Table 3, while Table 4 presents the performance figures for this design. We also run the tool for best performance design with different imposed maximum footprints. We run the search for 30 minutes for each given γ. The result is presented in Fig. 30. Note that the y-axis, i.e. response time, starts from 9ms in this figure. In these cases, the first design is usually found under 300 seconds. The difference between the performance of design with $\gamma = 100\%$ and design with $\gamma = 49\%$ is larger than the other cases. Finally, we run the tool for combined footprint and response times objective with the same weight values as for the collision avoidance application and 30 minutes timeout. The results are shown in Fig. 31. In comparison to the collision avoidance application, the quality of the designs evolve slower in this case as the application consists of more transactions.

	WC ECUs	SC ECU	l_4	Other Links
α_i	4%	20%	2%	1%
Θ_i	40 μs	200 μs	20 μs	10 μs
Π_i	1 ms	1 ms	1 ms	1 ms

Table 3 Minimum footprint design for the collision avoidance application.

	RT	Age Delay	Reaction Delay
Trans_RL_1	20570 μs	81800 μs	121800 μs
Trans_RL_2	20570 μs	81800 μs	121800 μs
Trans_SC_RL_1	23250 μs	83940 μs	121800 μs
Trans_SC_RL_2	23250 μs	83940 μs	121800 μs
Trans_SC_RL_3	23250 μs	83940 μs	121800 μs

Table 4 Performance in minimum footprint design for the collision avoidance application.

8.3 Discussion

In this case study we have demonstrated the usability of the proposed end-to-end resource reservation model. The case study, inspired from a real industrial application, is of reasonable size as it consists of two applications each with several transactions. The resource reservation parameters computed by the proposed reservation design method render the system schedulable. We have also demonstrated an important feature that is offered by the proposed model, method and analysis which allows the applications to be designed and analyzed in isolation without being affected by each other. This feature can assist the developers and integrators at the design phase by supporting the design and implementation of each application by a separate team. Moreover, to improve the model and to guide the designers for assigning the best reservation parameters, a reservation design method is presented. We have also shown that this method can find the reservation design for heavier applications, consisting of more than 20 transactions, in reasonable amount of time. We showed

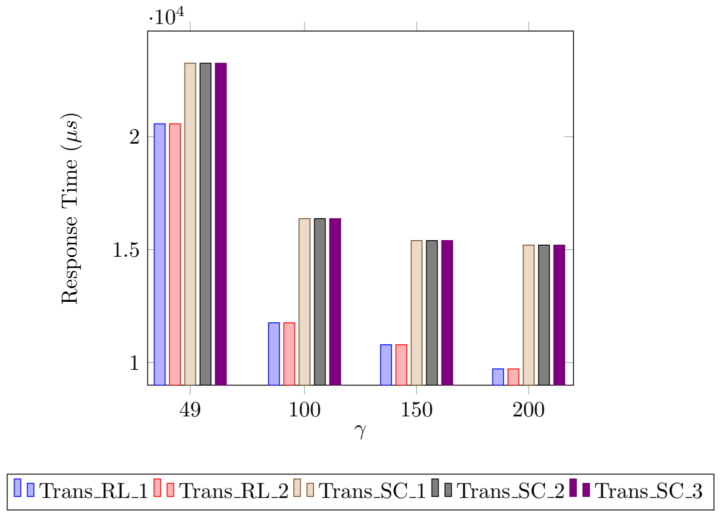


Fig. 30 Response times against footprint γ for the steer-by-wire application.

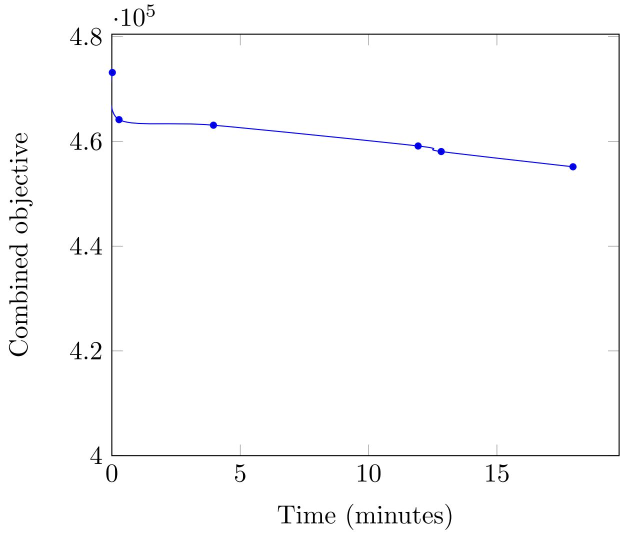


Fig. 31 Evolution of the design's objective value against search time.

that our reservation design method always finds a feasible solution after a few seconds. Finding the optimal solution, however, depends on the selected optimization criterion. Minimum footprint designs are also found quickly, while the best performance designs usually require search over large portions of the search space, hence take longer time. However, since we guide the solver in such a way that it searches promising parts of the search space first, we quickly achieve good designs.

9 Conclusion and Future Work

This chapter presented a new model for end-to-end resource reservation in distributed embedded systems. The model supports reservation on both computational nodes and communication among nodes. The proposed model considers a general transactional model such that the transactions can have several tasks in each node and several messages in the network. In addition, the tasks and messages in the transaction can have various activation patterns including trigger, data and mixed chains. Such activations patterns are common in several industrial domains. Therefore, the presented model responds to a vast number of industrial domains with different requirements with respect to design choices. Furthermore, the chapter presented end-to-end timing analysis for the reservation model in order to compute the end-to-end response times as well as end-to-end delays. The design method exploits several optimization criteria in computing the reservation parameters. The design method can play a key role in helping the system integrator making efficient design decisions and improving the schedulability of the system. From the development perspective of such systems, the chapter identified requirements for software development component models to support the proposed resource reservation model. Then, in order to provide a proof of concept, the chapter presented implementation extensions to the Rubus Component Model (RCM), which is an existing industrial component model for vehicular embedded systems. A technique to extract end-to-end models from the software architectures of the systems with resource reservation is also presented. Finally, the chapter presented the usability of the model, the reservation method and timing analysis with the help of a vehicular application case study.

References

1. Timing Augmented Description Language (TADL2) syntax, semantics, metamodel Ver. 2, Deliverable 11, Aug. 2012.
2. AUTOSAR Techincal Overview, Release 4.1, Rev. 2, Ver. 1.1.0., The AUTOSAR Consortium, Oct., 2013. http://autosar.org.
3. Catalog of Specialized CORBA Specifications. OMG Group. http://www.omg.org/technology/documents/.
4. EAST-ADL Domain Model Specification, V2.1.12,. http://www.east-adl.-info/Specification/V2.1.12/EAST-ADL-Specification_V2.1.12.pdf.
5. Luis Almeida and Paulo Pedreiras. Scheduling within temporal partitions: Response-time analysis and server design. In *Proceedings of the 4th ACM International Conference on Embedded Software*, October 2004.
6. A. Aminifar, E. Bini, P. Eles, and Z. Peng. Analysis and design of real-time servers for control applications. *IEEE Transactions on Computers*, March 2016.
7. Mohammad Ashjaei, Moris Behnam, Paulo Pedreiras, Reinder J. Bril, Luis Almeida, and Thomas Nolte. Reduced buffering solution for multi-hop HaRTES switched Ethernet networks. In *The 20th IEEE Int. Conference on embedded and Real-Time Computing Systems and Applications*, August 2014.
8. Mohammad Ashjaei, Nima Khalilzad, Saad Mubeen, Moris Behnam, Ingo Sander, Luis Almeida, and Thomas Nolte. Designing end-to-end resource reservations in predictable distributed embedded systems. *Real-Time Systems*, June 2017.

9. N. Audsley, A. Burns, M. Richardson, K. Tindell, and A.J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284–292, September 1993.
10. N. C. Audsley, A. Burns, R. I. Davis, K. W. Tindell, and A. J. Wellings. Fixed priority preemptive scheduling: An historical perspective. *Real-Time Systems*, 8(2):173–198, March/May 1995.
11. Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. *ACM symposium on operating systems principles*, October 2003.
12. Unmesh D. Bordoloi, Amir Aminifar, Petru Eles, and Zebo Peng. Schedulability analysis of ethernet AVB switches. In *The 20th IEEE International Conference on embedded and Real-Time Computing Systems and Applications*, August 2014.
13. M. Broy, I.H. Kruger, A. Pretschner, and C. Salzmann. Engineering automotive software. *Proceedings of the IEEE*, 95(2):356 –373, feb. 2007.
14. S. Chatterjee and J. Strosnider. Distributed pipeline scheduling: end-to-end analysis of heterogeneous, multi-resource real-time systems. In *The 15th Int. Conf. on Distributed Computing Systems*, May 1995.
15. Ivica Crnkovic and Magnus Larsson. *Building Reliable Component-Based Software Systems*. Artech House, Inc., Norwood, MA, USA, 2002.
16. T. Cucinotta and L. Palopoli. QoS control for pipelines of tasks using multiple resources. *IEEE Transactions on Computers*, March 2010.
17. R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien. Controller area network (CAN) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*, 35:239–272, 2007.
18. Rob Davis and Alan Burns. An investigation into server parameter selection for hierarchical fixed priority pre-emptive systems. In *16th International Conference on Real-Time and Network Systems*, October 2008.
19. Z. Deng and J. W. S. Liu. Scheduling real-time applications in an open environment. In *The 18th IEEE Real-Time Systems Symposium*, December 1997.
20. G. Dermler, W. Fiederer, I. Barth, and K. Rothermel. A negotiation and resource reservation protocol (NRP) for configurable multimedia applications. In *The Third IEEE International Conference on Multimedia Computing and Systems*, June 1996.
21. A. Easwaran, M. Anand, and I. Lee. Compositional analysis framework using EDP resource models. In *28th IEEE International Real-Time Systems Symposium*, December 2007.
22. N. Feiertag, K. Richter, J. Nordlander, and J. Jonsson. A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics. In *Workshop on Compositional Theory and Technology for Real-Time Embedded Systems*, December 2008.
23. Xiang (Alex) Feng. Towards real-time enabled microsoft windows. In *The 5th ACM International Conference on Embedded Software*, 2005.
24. N. Fisher and F. Dewan. Approximate bandwidth allocation for compositional real-time systems. In *21st Euromicro Conference on Real-Time Systems*, July 2009.
25. Sourav Ghosh, Jeffery Hansen, Ragunathan (Raj) Rajkumar, and John Lehoczky. Integrated resource management and scheduling with multi-resource constraints. In *The 25th IEEE International Real-Time Systems Symposium*, 2004.
26. K. Hänninen et.al. The Rubus Component Model for Resource Constrained Real-Time Systems. In *3rd IEEE International Symposium on Industrial Embedded Systems*, June 2008.
27. M. Gonzalez Harbour and J.C. Palencia. Response time analysis for tasks scheduled under EDF within fixed priorities. In *24th IEEE Real-Time Systems Symposium*, pages 200–209, December 2003.
28. R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst. System level performance analysis - the symta/s approach. *Computers and Digital Techniques*, 152(2):148–166, March 2005.
29. Thomas A. Henzinger and Joseph Sifakis. The Embedded Systems Design Challenge. In *14th International Symposium on Formal Methods (FM), Lecture Notes in Computer Science*, pages 1–15. Springer, 2006.

30. IEEE. IEEE Std. 802.1Q, IEEE standard for local and metropolitan area networks, bridges and bridged networks. 2014.
31. Z. Iqbal, L. Almeida, R. Marau, M. Behnam, and T. Nolte. Implementing hierarchical scheduling on COTS Ethernet switches using a master/slave approach. In *7th IEEE International Symposium on Industrial Embedded Systems*, June 2012.
32. Zahid Iqbal, Luis Almeida, and Moris Behnam. Designing network servers within a hierarchical scheduling framework. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, 2015.
33. ISO 11898-1. Road Vehicles interchange of digital information controller area network (CAN) for high-speed communication, ISO Standard-11898, Nov. 1993.
34. M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, March 1986.
35. Xu Ke, K. Sierszecki, and C. Angelov. COMDES-II: A Component-Based Framework for Generative Development of Distributed Real-Time Control Systems. In *13th International Conference on Embedded and Real-Time Computing Systems and Applications*, Aug. 2007.
36. N. Khalilzad, M. Ashjaei, L. Almeida, M. Behnam, and T. Nolte. Adaptive multi-resource end-to-end reservations for component-based distributed real-time systems. In *13th IEEE Symposium on Embedded Systems For Real-time Multimedia*, October 2015.
37. R. Apt Krzysztof. *Principles of Constraint Programming*. Cambridge University Press, 2003.
38. K. Lakshmanan and R. Rajkumar. Distributed resource kernels: OS support for end-to-end resource isolation. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, April 2008.
39. G. Lipari and E. Bini. Resource partitioning among real-time applications. In *Proceedings of 15th Euromicro Conference on Real-Time Systems*, July 2003.
40. Giuseppe Lipari and Enrico Bini. A methodology for designing hierarchical scheduling systems. *Journal of Embedded Computing - Real-Time Systems*, April 2005.
41. C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of ACM*, 20(1):46–61, 1973.
42. J. Loeser and H. Haertig. Low-latency hard real-time communication over switched ethernet. In *16th Euromicro Conference on Real-Time Systems*, June 2004.
43. R. Marau, M. Behnam, Z. Iqbal, P. Silva, L. Almeida, and P. Portugal. Controlling multi-switch networks for prompt reconfiguration. In *9th International Workshop on Factory Communication Systems*, May 2012.
44. S. Mubeen, M. Ashjaei, T. Nolte, J. Lundbck, M. Glnander, and K. L. Lundbck. Modeling of end-to-end resource reservations in component-based vehicular embedded systems. In *2016 42th Euromicro Conference on Software Engineering and Advanced Applications*, August 2016.
45. S. Mubeen, H. Lawson, J. Lundbäck, M. Gålnander, and K. L. Lundbäck. Provisioning of predictable embedded software in the vehicle industry: The rubus approach. In *2017 IEEE/ACM 4th International Workshop on Software Engineering Research and Industrial Practice (SER IP)*, pages 3–9, May 2017.
46. S. Mubeen, J. Mäki-Turja, and M. Sjödin. Support for end-to-end response-time and delay analysis in the industrial tool suite: Issues, experiences and a case study. *Computer Science and Information Systems*, 10(1), 2013.
47. Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Communications-Oriented Development of Component- Based Vehicular Distributed Real-Time Embedded Systems. *Journal of Systems Architecture*, 60(2):207–220, 2014.
48. T. Nolte, M. Nolin, and H. A. Hansson. Real-time server-based communication with can. *IEEE Transactions on Industrial Informatics*, August 2005.
49. A. B. Oliveira, A. Azim, S. Fischmeister, R. Marau, and L. Almeida. D-RES: Correct transitive distributed service sharing. In *IEEE Emerging Technology and Factory Automation*, September 2014.
50. R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek. A resource allocation model for QoS management. In *The 18th IEEE Real-Time Systems Symposium*, December 1997.

51. Saowanee Saewong, Ragunathan Rajkumar, John P. Lehoczky, and Mark H. Klein. Analysis of hierarchical fixed-priority scheduling. In *Proceedings of the 14th Euromicro Conference on Real-Time Systems*, July 2002.
52. Rui Santos, Moris Behnam, Thomas Nolte, Paulo Pedreiras, and Luis Almeida. Multi-level hierarchical scheduling in Ethernet switches. In *Proc. of the Int. Conference on Embedded Software*, October 2011.
53. Rui Santos, Moris Behnam, Thomas Nolte, Paulo Pedreiras, and Luís Almeida. Multi-level hierarchical scheduling in ethernet switches. In *9th International Conference on Embedded Software*, 2011.
54. Christian Schulte, Guido Tack, and Mikael Z. Lagerkvist. Modeling and programming with Gecode. Technical report, March 2015.
55. Sverine Sentilles, Aneta Vulgarakis, Tomas Bures, Jan Carlson, and Ivica Crnkovic. A Component Model for Control-Intensive Distributed Embedded Systems. In *CBSE 2008)*, pages 310–317.
56. L. Sha, T. Abdelzaher, K.-E. Årzén, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. K. Mok. Real time scheduling theory: A historical perspective. *Real-Time Systems*, 28(2):101–155, February 2004.
57. Insik Shin and Insup Lee. Periodic resource model for compositional real-time guarantees. In *24th IEEE Real-Time Systems Symp.*, December 2003.
58. Michal Sojka, Pavel Píša, Dario Faggioli, Tommaso Cucinotta, Fabio Checconi, Zdenk Hanzálek, and Giuseppe Lipari. Modular software architecture for flexible reservation mechanisms on heterogeneous resources. *Journal of System Architecture*, April 2011.
59. Brinkley Sprunt, Lui Sha, and John Lehoczky. Aperiodic task scheduling for hard-real-time systems. *Real-Time Systems Journal*, 1(1):27–60, June 1989.
60. J. K. Strosnider, J. P. Lehoczky, and Lui Sha. The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments. *IEEE Transactions on Computers*, 44(1):73–91, January 1995.
61. T. Scott, Chief Technology Officer, Information Systems and Services Division, General Motors Coorporation. Keynote Talk. In *CeBIT America Conference*, May 2004.
62. Ernesto Wandeler and Lothar Thiele. Real-time interfaces for interface-based design of real-time systems with fixed priority scheduling. In *the 5th ACM International Conference on Embedded Software*, EMSOFT '05, 2005.

Acknowledgements The research leading to this chapter has been supported by the Swedish Foundation for Strategic Research (SSF) within the project FIC and the Swedish Knowledge Foundation (KKS) within the project PreView. We would like to thank all our industrial partners, specially Arcticus Systems, Volvo Construction Equipment and BAE Systems Hägglunds, Sweden.