CPA: Compositional Performance Analysis 23

Robin Hofmann, Leonie Ahrendts, and Rolf Ernst

Abstract

In this chapter we review the foundations Compositional Performance Analysis (CPA) and explain many extensions which support its application in design practice. CPA is widely used in automotive system design where it successfully complements or even replaces simulation-based approaches.

Acronyms

ACK	Acknowledgement
ARQ	Automatic Repeat Request
BCET	Best-Case Execution Time
BCRT	Best-Case Response Time
CAN	Controller Area Network
COTS	Commercial/Components Off-The-Shelf
CPA	Compositional Performance Analysis
DAG	Directed Acyclic Graph
DMA	Direct Memory Access
ECU	Electronic Control Unit
FIFO	First-In First-Out
MCR	Mode Change Request
SPNP	Static-Priority Non-Preemptive
SPP	Static Priority Preemptive
TWCA	Typical Worst-Case Analysis
TWCRT	Typical Worst-Case Response Time
WCET	Worst-Case Execution Time
WCRT	Worst-Case Response Time

R. Hofmann (✉) • L. Ahrendts • R. Ernst
Institute of Computer and Network Engineering, Technical University Braunschweig, Braunschweig, Germany
e-mail: rhofmann@ida.ing.tu-bs.de; ahrendts@ida.ing.tu-bs.de; ernst@ida.ing.tu-bs.de

S. Ha, J. Teich (eds.), *Handbook of Hardware/Software Codesign*,
DOI 10.1007/978-94-017-7267-9_24

Contents

23.1 Motivation

Despite the risk of overlooking critical corner cases, design verification is for the most part based on execution and test using simulation, prototyping, and the final system. Formal analysis and verification are typically used in cases where errors are particularly expensive or may have catastrophic consequences, such as in safety critical or high availability systems. Such formal methods have considerably improved in performance and usability and can be used on a broader scale to improve design quality, but they must cope with growing hardware and software architecture complexity.

The situation is similar when we consider system timing verification. Formal timing analysis methods have been around for decades, starting with early work by Liu and Layland in the 70s [24] which provided schedulability analysis and worst-case response time data for a limited set of task system classes and scheduling strategies for single processors. In the meantime, there were dramatic improvements in the scope of considered tasks systems, architectures, and timing models. One of the key analysis inputs is the maximum execution time of a task, the Worst-Case Execution Time (WCET), where there has been similar progress [51]. As in the case of function verification, progress in hardware and software architectures made analysis more challenging. In particular the dominant trend focusing Commercial/Components Off-The-Shelf (COTS) on average or typical system performance has impaired system predictability forcing analysis to resort to more conservative methods (i.e., methods that overestimate the real worst case). While architectures with higher predictability have been proposed [29,51], design practice currently has to live with the ongoing trend.

In some respect, efficient formal timing verification is even harder than function verification because of systems integration. Today, a vehicle, an aircraft, a medical device, and even a smartphone integrates many applications sharing the same network, processors, and run-time environment. This leads to potential timing interference of seemingly unrelated applications. The integrated modular architecture (IMA), standardized as ARINC 653 [45] for aircraft design, and even more the automotive AUTOSAR standard are perfect examples for such software architectures. They also stand for different philosophies. While ARINC 653 takes a constructive approach and uses scheduling to obtain application isolation at the cost of resource efficiency, AUTOSAR does not constructively prevent timing interference, but the related automotive safety standard ISO 26262 requires proof of "freedom from interference" for safety critical applications.

However, even with extensive runs on millions of cases, simulation and prototyping remain an investigation of collections of use cases with decreasing expressiveness for large integrated systems. Therefore, there is a strong incentive to use formal timing analysis methods at least on the network level. For example, there are formal methods for some protocols such as the automotive Controller Area Network (CAN) bus which is the dominant automotive bus standard today [10]. Unfortunately, current automotive systems are not only large but heterogeneous combining different protocols and scheduling and arbitration strategies. To make things worse, the component and network technology incrementally develops over time challenging flexibility and scalability of any formal timing analysis.

In this situation, the introduction of modular timing analysis methods which support composition of analyses for different scheduling and arbitration strategies as well as different architectures and application systems with a variety of models-of-computation was considered a breakthrough. Today, most automotive manufacturers and many suppliers use formal timing analysis as part of their network development. A corresponding tool, SymTA/S, has been commercialized and is widely used. The original ideas which led to that tool can be found in [37].

This chapter presents the general concept of the Compositional Performance Analysis (CPA) and extensions of the last couple of years. Since this is an overview chapter, it stays on the surface to keep readability. For more details, the reader is referred to the large body of related scientific papers covering compositional performance analysis and a related approach based on the Real-Time Calculus (RTC) [49].

23.2 Fundamentals

CPA is an analysis framework which serves to formally derive the timing behavior of embedded real-time systems.

From a hardware perspective, an embedded real-time system consists of a set of interconnected components. These components include communication and computation elements as well as sensors and actuators which act as the connection to the system environment. The interconnected components represent the platform on which software applications with real-time requirements are executed. A software application is composed of tasks, entities of computation, which are distributed over and executed on different components of the system.

The execution order of tasks belonging to one application is constrained, for instance, the read of a sensor must be performed before the computation of a control law and the control of an actuator. Moreover, if several tasks are executed on one component, the tasks have to share the processing service the component offers. This has obviously an impact on the timing behavior of each task. As a result, in order to determine the timing behavior of the system, it is not sufficient to focus on isolated tasks. Apart from the interaction of tasks which are caused by functional dependencies (imposed execution order), nonfunctional dependencies (share of component service) have also to be taken into consideration.

In the following, the system model used for CPA is described, and then the compositional analysis principle is deduced. Following the above argumentation, CPA structures its system model with respect to three aspects: (1) the individual tasks, (2) the individual components with intra-component (local) dependencies between mapped tasks, and (3) the system platform with inter-component (global) dependencies between mapped tasks. The analysis is structured according to the local and global aspects, and it is compositional in the sense that the timing properties of the system can be conclusively derived from its constituting components.

23.2.1 Timing Model

In the following the timing model of a real-time embedded system is described as it is used in CPA. The timing model is layered and includes the task timing model, the component timing model, and the system timing model. All three layers are explained in detail below and are illustrated in Fig. 23.1.

23.2.1.1 Task Timing Model

In this section the timing behavior of an individual task τ_i is presented which is characterized completely by its execution time C_i. The execution time C_i is the amount of service time which a component has to provide in order to process task τ_i. The actual execution time of a task τ_i does not only heavily depend on the task input and the task state but also on the execution platform. For instance, the processor architecture, the cache architecture, and the Direct Memory Access (DMA) policy impact the execution time. As a result, a task τ_i does not have a static but rather a varying execution time C_i as illustrated in Fig. 23.2. For the CPA, the lower and the upper bound on the task execution are of interest because they include every intermediate timing behavior. The lower bound on the task execution time is called Best-Case Execution Time (BCET) denoted as C_i^-, whereas the upper bound is called the WCET denoted as C_i^+.

Different methods exist to derive the BCET and WCET of a task τ_i. One method is to simulate the task execution under different scenarios and observe the required execution time. Since the number of test cases is naturally limited, the simulation is bound not to cover all corner cases thus underestimating the WCET and overestimating the BCET. Formal program analysis, on the other hand, evaluates the source or object code associated with a task τ_i and takes into account

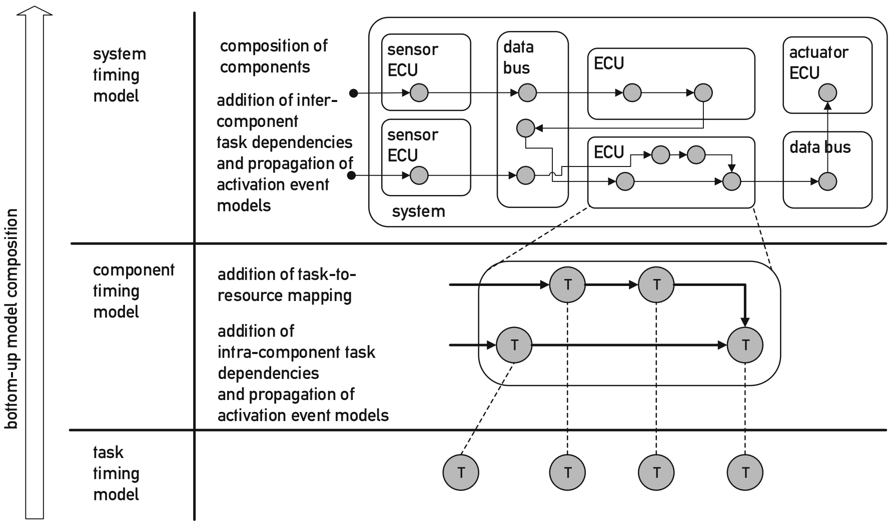
bottom-up model composition
system timing model
composition of components
addition of inter-component task dependencies and propagation of activation event models
sensor ECU
sensor ECU
data bus
ECU
ECU
actuator ECU
data bus
system
component timing model
addition of task-to-resource mapping
addition of intra-component task dependencies and propagation of activation event models
T
task timing model

Fig. 23.1 System timing model. The system timing model used in CPA comprises three aspects: (1) individual tasks, (2) individual components with mapped tasks and local task dependencies, and (3) the entire platform with mapped tasks and global task dependencies

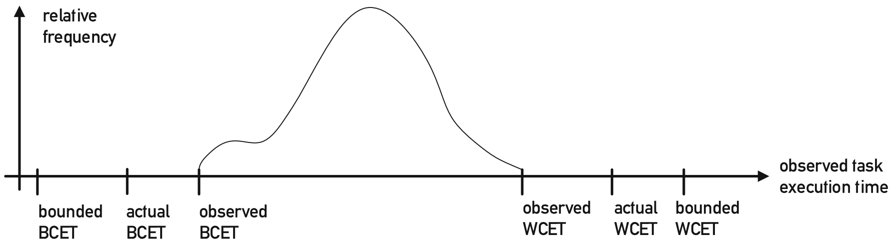

Fig. 23.2 Relative frequency of the observed execution time of a task. Observed execution times from simulations can in general not bound the actual best-case and worst-case behaviors. In contrast, a formal bound tends to overestimate the actual WCET and to underestimate the actual BCET

the architectural properties of the execution platform. Program analysis is capable of deriving a safe upper bound on the WCET of a task τ_i, relying on worst-case program configuration and worst-case program input in order to cover all behavioral corner cases. Apart from static WCET bounds, parametric WCET bounds have been proposed [9]. Tools for the derivation of WCET bounds like aiT [1] are available. As programs can be become arbitrarily complex, program analysis makes conservative assumptions leading to significant WCET overestimations and BCET underestimations. This is the reason why program analysis for BCET/WCET-estimation is often only used for tasks with high criticality, i.e., with high impact on system safety. While there is extensive research dedicated to the WCET analysis

[52], it is not part of CPA itself. Therefore, the BCETs and WCETs are assumed to be known as safe but possibly conservative values.

23.2.1.2 Local View: Component Timing Model

The component timing model represents a locally restricted view on the system. It focuses on the timing behavior in the scope of individual system components.

Once a set of tasks is mapped to a component, the timing behavior of each task can no longer be treated in isolation. On the one hand, tasks interact due to *nonfunctional dependencies*, i.e., the share of component service which is determined by the local scheduling policy. On the other hand, tasks interact due to *functional dependencies*. All tasks belonging to one application have activation patterns, and an execution order imposed by the application which per definition specifies the high-level functional context and the functional interaction of tasks.

The derivation of a task activation pattern and the realization of the required execution order of tasks is done by the propagation of activation events in the CPA component timing model. Assume a precedence-constrained execution order of a set of tasks which are mapped to a component as illustrated in Fig. 23.1 for the component timing model layer. The activation pattern of a task τ_i which represents the first element in the execution order of tasks (head of a task chain) is determined by the behavior of an event source outside of the component. Such an event source can either be located in the system environment or at another system component. A task τ_j which directly succeeds task τ_i in the task chain is activated by the termination events of task τ_i. This propagation of activation events applies for all elements in the task chains. Note that the event propagation in CPA timing model implies that the activation patterns of tasks are derived from high-level functional constraints, and do not represent a direct property of the individual task.

In this section, first the component abstraction and then the component scheduler is introduced which organizes the share of component service. Hereafter, in order to reason about service demand and event propagation among precedence-constrained tasks, the principles of activation/termination traces and activation/termination event models are explained.

Component Abstraction

System components are also called *resources* and are characterized by their property to provide processing service to tasks. A component can either be a *computation resource* or a *communication resource*, and depending on the kind of resource, a task can either represent an algorithm to be computed or a data frame to be transmitted. For instance, an Electronic Control Unit (ECU) is a computation resource which may provide processing service to tasks computing a control law. A data bus, in contrast, can be modeled as a communication resource transmitting data frames.

Scheduler

A resource can only serve one task at a time; hence, if more than one task is ready to execute, it has to be decided which task will be processed next. The decision process is performed by the *scheduler* of the resource. It specifies when to start and pause the execution of pending tasks. Commonly used scheduling policies for embedded real-time systems are Static Priority Preemptive (SPP) and Static-Priority Non-Preemptive (SPNP) *scheduling*. We will use these policies as important examples throughout the chapter, noting that CPA is not limited to static priority policies.

Activation Traces and Termination Traces

A task is activated by an *activation event*, where activation means that the task is moved from a sleeping state to a ready-to-execute state. Such an activation event can either be *time triggered* or *event triggered*. A time-triggered activation occurs according to a predefined time pattern, whereas an event-triggered activation is a reaction to a certain true condition in the system or the system environment. Additionally to being activated by an activation event, a task produces a *termination event* when it has finished.

An *activation trace* of a task τ_i describes the set of instants at which an activation event for a task τ_i takes place. A similar definition applies to the *termination trace*. An activation event for a task τ_i originates either from an *event source* which is triggered by the system environment or another external event source like a timer, or it is produced by a predecessor task τ_j if a precedence constraint with respect to the execution order exists between two tasks in the form of $\tau_j \rightarrow \tau_i$ (τ_j precedes τ_i). The termination event of the predecessor task τ_j, produced at the end of its execution, then represents the activation event for task τ_i. If the predecessor task is not local to the component, an external event source with a conservative activation pattern is initially assumed in the CPA component timing model, see Sect. 23.2.2.2.

Activation event traces from event sources and predecessor tasks can be combined to form a joint activation event trace. The *join* of two event traces can either follow an AND or an OR logic. An AND logic implies that only if an event from both incoming event traces is available at a given time, an event is produced on the joint event trace. An OR logic requires an event from only one of the two incoming event traces to produce an event on the joint trace. It is also possible for a single event trace to *fork*, i.e., to serve as an activation trace for multiple tasks. Figure 23.3 illustrates how activations events propagate through the system. Note that activation and termination events do not include any information on data, rather the event concept is agnostic of the concrete processed and transferred data and focuses on the timing behavior of tasks.

Event Models

An event trace of a task τ_i captures the sequence of event occurrences with respect to task τ_i for a given system trajectory in the system state space. Due to the countless number of possible system trajectories, it is not feasible to perform a timing analysis for all possible event traces. Rather, the timing analysis has to be restricted to corner cases which bound the timing behavior of the task τ_i in all other cases. CPA uses for this purpose *arbitrary activation [termination] event models* where arbitrary means that an arbitrary activation [termination] behavior of a task τ_i can be bounded by the event model [36].

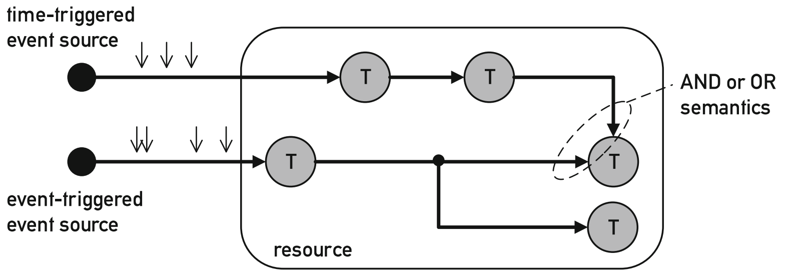

Fig. 23.3 **Event traces**. Tasks are activated by activation events from time-triggered or event-triggered event sources as well as by the termination events of predecessor tasks. The *arrows* connecting event sources and tasks as well as tasks among each other indicate the flow of events through the system. The *small arrows* $\downarrow$ indicate individual activation events which occur regularly in case of time-triggering and irregularly in case of event-triggering and propagate through the system

An event model of a task τ_i is defined by the set of two *distance functions* $\delta_i^-, \delta_i^+ : \mathbb{N}_0 \to \mathbb{R}_0^+$, namely, the minimum distance function δ_i^- and the maximum distance function δ_i^+. The minimum distance function $\delta_i^-(n)$ describes the minimum time distance between any n consecutive activation [termination] events of task τ_i. The maximum distance function $\delta_i^+(n)$ describes the maximum time distance between any n consecutive activation [termination] events of task τ_i. The distance between zero and one events is defined for mathematical convenience as zero so that $\delta_i^{+,-}(0) = \delta_i^{+,-}(1) := 0$.

Pseudo inverses of the distance functions are the *arrival functions* $\eta_i^+, \eta_i^- : \mathbb{R}_0^+ \to \mathbb{N}_0$. The maximum arrival function η_i^+ is the pseudo inverse of the minimum distance function δ_i^-, and the minimum arrival function η_i^- is the pseudo inverse of the maximum distance function δ_i^+. The function $\eta^+(\Delta t)$, resp. $\eta^-(\Delta t)$, returns the maximum, resp. minimum, number of activation [termination] events of task τ_i within any half-open time interval $[t, t + \Delta t)$. For a time interval $\Delta t = 0$, the event arrival functions η_i^- and η_i^+ are defined as zero. The pseudo inverses are introduced because they often allow a more elegant mathematical formulation of timing analysis problems.

The pair of minimum and maximum distance functions, resp. arrival functions, describe the best-case and worst-case event trace of a task τ_i with respect to event frequency. If distance functions, resp. arrival functions, cannot be formally derived, it is possible to extract them from measured event traces [17].

A commonly used event model is the *PJd event model* [36] shown in Fig. 23.4. A PJd event model is applicable to a task τ_i which is activated periodically but may experience bursts of activations once in a while. It can be characterized by the three parameters period T_i, jitter J_i, and a minimum event distance d_i. Bursts occur if the jitter of periodic activation events, i.e., the maximum relative deviation in time from the exactly periodic activation instant, is larger than the task activation period. In this case the task may receive multiple new activation events before the current task invocation has terminated. The PJd event model is also of historical

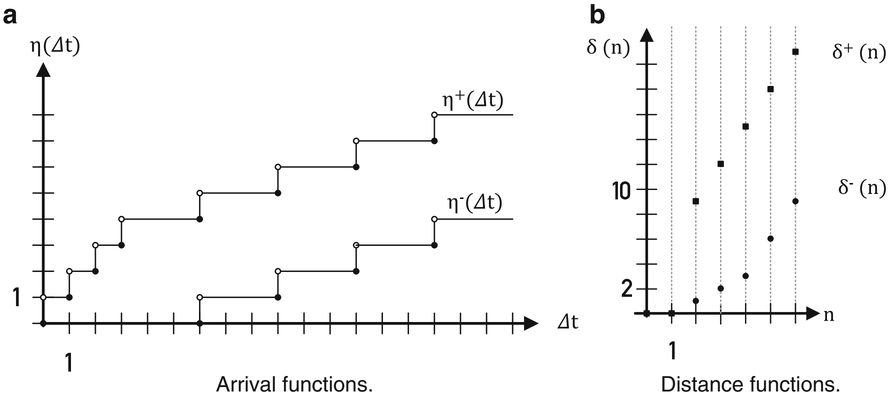

Fig. 23.4 PJd event model. (**a**) illustrates the maximum arrival function $\eta^{+}(\Delta t) = \min\left\{\left\lceil\frac{\Delta t}{d}\right\rceil, \left\lceil\frac{\Delta t+J}{T}\right\rceil\right\}$ and the minimum arrival function $\eta^{-}(\Delta t) = \max\left\{0, \left\lceil\frac{\Delta t-J}{T}\right\rceil\right\}$ for a PJd event model with $T = 3, J = 6, d = 1$. (**b**) illustrates for the same PJd model the minimum distance function $\delta^{-}(\Delta t) = \max\{(n-1)\cdot d, (n-1)\cdot T - J\}$ and the maximum distance function $\delta^{+}(\Delta t) = (n-1)\cdot T + J$ with $n \geq 2$. For small time intervals, the burst behavior dominates where the minimum event distance d bounds the maximum event frequency during the burst. For large time intervals, the periodic behavior with jitter dominates

importance because it was the basis for the description of activation patterns before the more general distance functions and arrival functions were introduced [18, 36]. It has the advantage that it can be described by a limited set of parameters and shows a periodic behavior for larger time intervals. In contrast, arbitrary event models may potentially extend indefinitely without showing a repetitive pattern. This complicating property of arbitrary event models has to be handled by an appropriate analysis approach which is able to extract relevant limited time windows for the investigation of the system timing behavior, see the busy period concept in Sect. 23.2.2.

23.2.1.3 Global View: System Timing Model

From a global perspective, the system is composed of a set of interconnected communication and computation resources which constitute the processing *platform*. Tasks are mapped to the resources and interact with each other forming larger functional entities, so-called applications.

In the global perspective, inter-component interactions between tasks as defined by applications become visible; see Fig. 23.1. Inter-component interactions between tasks are, like intra-component interactions, modeled by event propagation in CPA. To account for a dependency between two tasks τ_i, τ_j mapped on two different components where task τ_i precedes task τ_j, the termination event model of the predecessor task τ_i is propagated as activation event model to the successor task τ_j. The required termination event model of a task is determined iteratively during the system analysis procedure described in the following section.

23.2.2 Analysis

CPA is a systematic timing analysis method which serves to verify the timing properties of complex distributed real-time systems with heterogeneous components. The major challenge in analyzing such a system is to take into account the numerous interdependencies of task executions which result both from direct task interaction and indirect task interaction due to the share of resources.

CPA follows a compositional approach which first performs a local component-related timing verification step and then, in a global timing verification step, sets the local verification problems in a system context where inter-component dependencies are considered. The inter-component dependencies relate the local verification problems in such a manner that their inputs and outputs are linked. The relation of the local verification problems leads to a fixed point problem which converges if the propagation of outputs to inputs between related verification problems does not change the verification results any more. If the system is overloaded, the fixed point problem does not converge, and an abort criterion, e.g., the detected miss of a task deadline, is used to stop the iteration process.

In this section, first the local analysis is presented and then the superordinate global analysis is introduced.

23.2.2.1 Local Analysis

Local analysis refers to the analysis of timing properties of an individual system resource which processes tasks according to a given scheduling policy. The local analysis is based on the component timing model.

Resource Utilization

The *utilization* U of a resource is defined as the quotient of the execution request which the resource receives and the available service time which it can provide. It is computed by accumulating the utilization U_i that each task τ_i with $i = 1 \ldots N$ mapped to this resource imposes. The maximum utilization U_i^+ that an individual task τ_i can impose on a resource is given if the task requests its maximum execution time C_i^+ at its maximum activation frequency

$$U_i^+ = \lim_{n \to \infty} \frac{n \cdot C_i^+}{\delta_i^-(n)}. \tag{23.1}$$

The maximum utilization of a resource $U^+ = \sum_{i=1}^{N} U_i^+$ is an important variable to determine whether the resource is overloaded, and consequently the tasks are not schedulable. Apparently, it is impossible to schedule tasks sets with a resource utilization larger than one. In this case, the local analysis will be stopped. While being a necessary (under some conditions even sufficient) indicator for the schedulability of a task set, the utilization is not an appropriate means to describe the system timing behavior in detail. The utilization of a resource does not give any insight into the sequence of execution and suspension phases during the processing of a task which are determined by the applied scheduling policy.

Worst-Case Response Time

Since tasks which are allocated to the same resource have to share its service, the processing of a task τ_i is preempted if other tasks with higher priority are activated. This is illustrated in Fig. 23.5. The time interval between the activation and the termination of a task τ_i, including all suspension phases, is defined as the response time of task τ_i denoted as R_i. The minimum response time of task τ_i, denoted as R_i^-, and the maximum response time of task τ_i, denoted as R_i^+, serve to bound the response time behavior of task τ_i.

The decisive property of hard real-time systems is that no task τ_i is allowed to miss its deadline D_i. In other words, it is required that the deadline D_i is an upper bound on the worst-case response time R_i^+. A major purpose of timing analysis like CPA is to verify this system property which is crucial for safe operation of real-time systems.

Determining the Worst-Case Response Time

In order to verify whether the real-time requirement $R_i^+ \leq D_i$ for a task τ_i is fulfilled, the worst-case response time R_i^+ needs to be determined. Obviously, it is not possible to explore the entire system state space for this purpose. Rather a worst-case scenario has to be derived which allows to find the worst-case response time R_i^+ in a limited time window.

The limited time window of system behavior comprising the worst-case response time behavior of task τ_i is called the *longest level-i busy period* [23]. The longest level-i busy period is initiated by the so-called critical instant. The critical instant describes the alignment of task activations and the execution times which lead to the maximum interference with respect to task τ_i and consequently to the worst-case response time R_i^+. The longest level-i busy period closes if the investigation of a longer time interval is known not to contribute any new information to the worst-case response time analysis. In the following, first the concept of the level-i busy period is explained. Then the multiple activation scheduling horizon as well as the multiple activation processing time processing time are introduced. Both of those

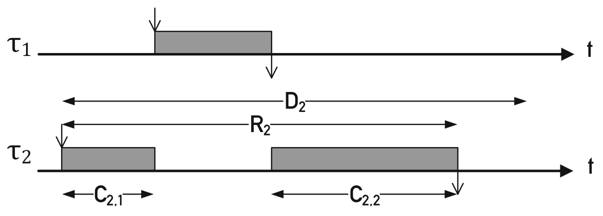

Fig. 23.5 **Response time and relative deadline.** Task τ_2 is preempted during execution by task τ_1 which is of higher priority. Therefore, the response time R_2 is larger than the execution time $C_2 = C_{2,1} + C_{2,2}$. The response time R_2 is still smaller than the deadline D_2

variables serve to formally describe the processing behavior of a local resource with respect to task τ_i within the level-i busy period.

Busy Period

A level-i busy period [23] is a time interval during which a resource R is busy processing a task τ_i or tasks of higher priority than task τ_i under a fixed priority scheduling policy. Directly before and after the level-i busy period, the resource R is idle with respect to task τ_i and tasks of higher priority.

The level-i busy period is an elegant means to perform a worst-case response time analysis by investigation of a limited time window. The idea is that the response time behavior of task τ_i in a level-i busy period is completely independent of events outside of this time interval. A level-i busy period is separated from the preceding and succeeding level-i busy periods by idle phases of the resource R. An idle phase implies that the resource R is virtually reset to an initial state being ignorant of previous execution requests of task τ_i or tasks of higher priority. It is thus sufficient to investigate in the timing analysis the level-i busy period which comprises the worst-case response time behavior of task τ_i. This so-called longest level-i busy period is initiated by the critical instant, which creates the maximum possible interference with respect to task τ_i so that the worst-case response time of task τ_i can be observed within the longest level-i busy period. It is a skillful task to derive this critical alignment of task activations and service requests for a given system configuration.

In the following, the longest level-i busy period and the initiating critical instant are derived for an SPP-scheduled task set on a single processing resource R with no restrictions on the task activation event models. Assume that at an instant $t - \epsilon$, the resource R is idle with respect to task τ_i and all tasks with higher priority. Shortly after at instant t, task τ_i is activated for the first time and requests its maximum execution time C_i^+. The activation causes the creation of the first task instance, also called job, which is denoted as $\tau_i(1)$. If all tasks with higher priority than task τ_i are activated simultaneously with $\tau_i(1)$ at t and request their maximum execution time at the highest possible frequency, then the maximum interference with respect to task $\tau_i(1)$ is evoked. This alignment of activations is the critical instant, the starting point of the longest level-i busy period. The level-i busy period generally comprises more than one job of task τ_i. The reason is that before job $\tau_i(1)$ terminates, a second activation of task τ_i may occur. Consequently the resource stays busy processing job $\tau_i(2)$ even if job $\tau_i(1)$ terminated, and incoming jobs of tasks with priority higher than task τ_i will preempt $\tau_i(2)$ from time to time. The same may of course happen before job $\tau_i(2)$ terminates etc., and only when a job of task τ_i finishes before a new activation for task τ_i comes in and no tasks with higher priority are processed, the level-i busy period closes.

The critical instant for a task τ_i scheduled under an SPNP policy occurs (1) if task τ_i is activated simultaneously with all tasks of higher priority, (2) if task τ_i and all tasks of higher priority request their maximum execution times at highest possible frequency, and (3) if a task τ_j of lower priority, which has the largest execution time among all tasks with a priority lower than task τ_i, started execution just previously to the first activation of task τ_i.

In the formal response time analysis, the closure of the level-i busy period is represented by the solution of a fixed point problem. In order to be able to formulate a formal response time analysis, the multiple activation scheduling horizon and the multiple activation event processing time have to be introduced as done in the following paragraphs. Both variables mathematically describe the timing behavior of task τ_i within the level-i busy window.

Multiple Activation Scheduling Horizon

The *q-activation scheduling horizon* $S_i(q)$ of task τ_i is defined as the maximum half-open time interval which starts with the arrival of the first job $\tau_i(1)$ of any sequence of q consecutive jobs $\tau_i(1), \tau_i(2), \ldots \tau_i(q)$. The scheduling horizon closes at the (not included) point in time when a theoretical activation of task τ_i with an infinitesimally short execution time ϵ could be immediately served by the resource R after the processing of the q consecutive jobs. This theoretical activation is independent from the actual activation model of task τ_i since it is never actually executed [11].

The q-activation scheduling horizon generalizes the idea of the level-i busy period for a number q of activations. During the q-activation scheduling horizon, the resource R is busy processing task q jobs of τ_i and tasks of higher priority. The condition, which a theoretical activation with infinitesimally short execution time ϵ could be potentially served at the end of the scheduling horizon, enforces an idle time with respect to q jobs of task τ_i and all tasks of higher priority at the end of the scheduling horizon. The scheduling horizon for $q = q_i^+$ corresponds to the longest level-i busy period, where q_i^+ is the maximum number of activations of task τ_i which fall into the scheduling horizon of their respective predecessor jobs

$$q_i^+ = \min\{q \geq 1 \mid S(q) < \delta_i^-(q+1)\}. \tag{23.2}$$

For the SPP scheduling policy, the q-activation scheduling horizon $S_i(q)$ is the solution to the following fixed point equation

$$S_i(q) = q \cdot C_i^+ + \sum_{j \in hp(i)} C_j^+ \cdot \eta_j^+(S_i(q)). \tag{23.3}$$

As can be seen in Eq. 23.3, the scheduling horizon $S_i(q)$ is composed of two parts. Firstly, it contains the maximum time interval which is required to service q jobs of task τ_i. And secondly, it comprises the maximum interference caused by tasks of higher priority than task τ_i ($hp(i)$). The maximum interference is evoked if every task τ_j with $j \in hp(i)$ is activated according to its maximum arrival curve η_j^+ and every job requests the worst-case execution time C_j^+ during $S_i(q)$. At the end of $S_i(q)$, a hypothetical $q + 1$st activation of task τ_i with ϵ execution time could immediately be served because all q jobs of task τ_i are processed and no jobs of higher priority are pending. In case of the SPNP scheduling policy, the q-activation scheduling horizon $S_i(q)$ has to take into account the worst-case one-time blocking caused by a task of lower priority than task τ_i

$$S_i(q) = q \cdot C_i^+ + \max_{j \in lp(i)} \left\{C_j^+\right\} + \sum_{k \in hp(i)} \eta_k^+(S_i(q)) \cdot C_k^+. \tag{23.4}$$

Therefore, $S_i(q)$ is composed of (1) the maximum processing time of q jobs of task τ_i, (2) the maximum one-time blocking of task τ_i by a task of lower priority than task τ_i due to non-preemption, and (3) the maximum interference of tasks with a higher priority than task τ_i.

Multiple Activation Processing Time

The *q-activation processing time* $B_i(q)$ is defined as the time interval starting with the arrival of the first job $\tau_i(1)$ and ending at the termination of job $\tau_i(q)$ for any q consecutive activations of task τ_i which fall into the scheduling horizon of their respective predecessors.

The maximum q-activation processing time $B_i^+(q)$ serves as basis for the worst-case response computation. By definition, the maximum response time of the qth task instance, denoted as $R_i^+(q)$, is the difference of the maximum q-activation processing time of and its earliest possible time of activation

$$R_i^+(q) = B_i^+(q) - \delta_i^-(q). \tag{23.5}$$

The worst-case response time of a task τ_i, denoted as R_i^+, is the maximum response time of task τ_i within the longest level-i busy period, respectively, the q_i^+-activation scheduling horizon, thus

$$R_i^+ = \max_{1 \le q \le q_i^+} R_i^+(q). \tag{23.6}$$

For the SPP policy, the maximum q-activation processing time $B_i^+(q)$ is identical to the q-activation scheduling horizon $S_i(q)$ so that

$$B_i^+(q) = q \cdot C_i^+ + \sum_{j \in hp(i)} C_j^+ \cdot \eta_j^+(B_i^+(q)). \tag{23.7}$$

The identity of the q-activation processing time and the q-activation scheduling horizon is due to the sub-additive behavior of the SPP scheduling policy with respect to the processing times [11, 39]: $B_i^+(q + p) \le B_i^+(q) + B_i^+(p)$. This property is, however, not fulfilled for the SPNP scheduling policy. The maximum processing time $B_i^+(q)$ under the SPNP policy is the sum of the maximum queuing delay with respect to job $\tau_i(q)$, denoted as $Q_i^+(q)$, and the maximum execution time of job $\tau_i(q)$

$$B_i^+(q) = Q_i^+(q) + C_i^+. \tag{23.8}$$

The maximum queuing delay $Q_i^+(q)$ is the time a job $\tau_i(q)$ has to wait before it is selected for execution by the SPNP scheduler. Activations of tasks with a higher priority than task τ_i which occur during the execution of the job $\tau_i(q)$ do not prolong its processing time as by definition of the scheduling policy, it cannot be preempted once it has started executing. The queuing delay can be bounded from above by [10]

$$Q_i^+(q) = (q-1)\cdot C_i^+ + \max_{j\,\in\,lp(i)}\left\{C_j^+\right\} + \sum_{k\,\in\,hp(i)} C_k^+ \cdot \eta_k^+\left(Q_i(q)+\epsilon\right). \tag{23.9}$$

The maximum queuing delay accounts for (1) the maximum execution demand of all jobs of task τ_i activated prior to job $\tau_i(q)$, (2) the longest one-time lower priority blocking due to non-preemption, and (3) the longest higher priority blocking during queuing. The infinitesimally long time interval ϵ added to the queuing delay serves to check whether another job interfering with task $\tau_i(q)$ could start exactly at the end of the iteratively computed queuing delay, thus it extends the investigated time window. Note that Eq. 23.8–23.9 and Eq. 23.4 are not identical since the fixed point iteration for $B_i^+(q)$ accumulates higher priority interference only during the queuing delay plus ϵ, whereas $S_i(q)$ takes also into account interference of higher priority during the execution of the qth job.

Example

Consider the timing diagram in Fig. 23.6 which illustrates the concept of the multiple activation scheduling horizon and the multiple activation processing time. The timing diagram shows three tasks τ_1, τ_2, and τ_3 which are scheduled on a common resource under an SPNP policy, where task τ_1 is of higher priority than task τ_2 and task τ_2 is of higher priority than task τ_3. The scenario represents the worst case with respect to the response time of task τ_2 since (1) task τ_3 is activated just prior to tasks τ_1 and τ_2 with maximum execution demand, (2) task τ_1 causes maximum interference of higher priority, and (3) task τ_2 always requests its maximum execution time at highest possible frequency.

In the timing diagram, the multiple activation scheduling horizons and the multiple activation processing times are indicated. All scheduling horizons and processing times start at time 0. The termination of job $\tau_2(1)$ marks the end of $B_2^+(1)$. The scheduling horizon $S_2(1)$ is longer than $B_2^+(1)$ due to the higher priority interference which prevents that a hypothetical activation of task τ_2 with an infinitesimally short execution time ϵ could be immediately served after the first job $\tau_2(1)$. Note that the scheduling horizon is defined as a half-open interval and thus does not take interference of higher priority into account which arrives exactly at the interval boundary of $S_2(1)$. Since the activation of $\tau_2(2)$ falls into the scheduling horizon $S_2(1)$ of job $\tau_2(1)$, $B_2^+(2)$ exists. Again, the scheduling horizon $S_2(2)$ is longer than the processing time $B_2^+(2)$, but the activation of job $\tau_2(3)$ does not fall into the scheduling horizon $S_2(2)$. Thus, $S_2(2)$ is the longest scheduling horizon and

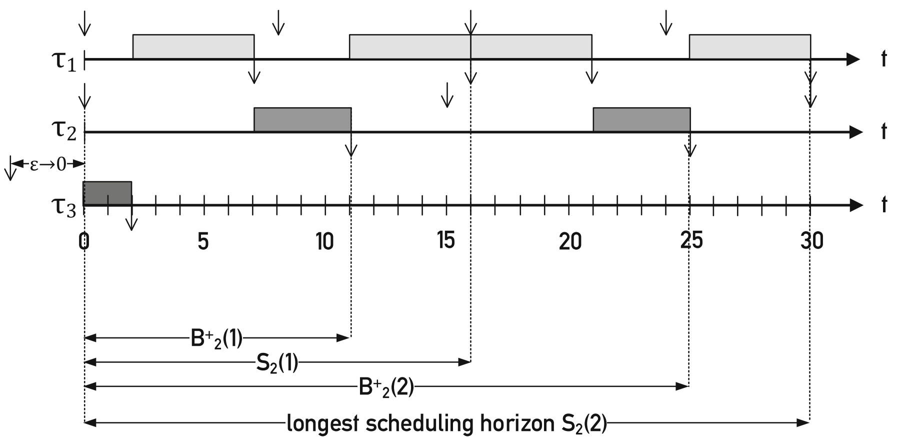

Fig. 23.6 **Scheduling horizons and processing times for SPNP scheduling**

represents the longest level-2 busy period. Note that the depicted scenario illustrates the non-subadditive behavior of the processing times: $B_2^+(2) > B_2^+(1) + B_2^+(1)$. The maximum processing time of job $\tau_2(1)$ is $B_2^+(1) = 11$, and its worst-case response time equals $R_2^+(1) = B_2^+(1) - \delta^-(1) = 11 - 0 = 11$. The maximum processing time of job $\tau_2(2)$ is $B_2^+(1) = 25$, and its worst-case response time equals $R_2^+(2) = B_2^+(2) - \delta^-(2) = 25 - 15 = 10$. Thus, a worst-case response time analysis yields the result $R_2^+ = \max\limits_{1 \le q \le q_i^+} R_2^+(q) = 11$.

Best-Case Response Time

The best-case response time R_i^- and the worst-case response time R_i^+ serve to bound the response time behavior of task τ_i. A simple approximation of the best-case response time R_i^- relies on the following assumptions: (1) the absence of interference by tasks with equal or higher priority, and (2) the request of the minimum execution time C_i^-.

$$R_i^- = C_i^-. \tag{23.10}$$

Even though this approximation does not necessarily represent a tight bound, it is usually acceptable as timing analysis aims to provide real-time guarantees and thus focuses particularly on worst-case behavior.

Jitter

Jitter represents the maximum time interval by which the occurrence of a given event may deviate from the expected occurrence of the event. The response time jitter of a task τ_i can hence be calculated as the difference between the best-case response time R_i^+ and the worst-case response time R_i^-

$$J_{i,resp} = R_i^+ - R_i^-. \tag{23.11}$$

Backlog

It is possible that an activation event for a task τ_i arrives before the previously activated task instance has been processed, e.g., due to high interference or jitter. In this case, a *backlog* of activation events with respect to task τ_i arises. To prevent any loss of information, all activation events are queued until they are processed. The queue semantics in CPA is characterized by a First-In First-Out (FIFO) organization and a nondestructive write to and a destructive read from a queue storing activation events. The determination of an appropriate queue size for pending activation events of task τ_i is important both to avoid dropping of events and over-dimensioning. The maximum activation backlog for task τ_i, denoted as b_i^+, is bounded by

$$b_i^+ = \max_{1 \le q \le q_i^+} \left\{0,\ \eta_i^+(B_i^+(q) + o_{i,out}) - q + 1\right\}. \tag{23.12}$$

The expression $\eta^+(\Delta t)$ represents the maximum possible number of task instances which can be activated in Δt, here with $\Delta t = B_i^+(q) + o_{i,out}$. The first term $B_i^+(q)$ is the maximum processing time for the task instance q of τ_i, the second term $o_{i,out}$ represents the maximum overhead required to remove a finished task instance from the activation queue [13]. The term $-q + 1$ accounts for the fact that previously activated task instances have already been finished. In other words, Eq. 23.12 computes the difference between the number of occurred activation events and processed ones, hence returning the number of pending activation events.

23.2.2.2 Global Analysis

In the previous section on the principle of the local analysis, we have shown how to compute the worst-case and best-case response times, the output jitter and the maximum activation backlog for a task with a given activation model. The local analysis is resource-related and does not consider any interaction between tasks on different resources. However, in reality many real-time applications are composed of multiple tasks which are distributed over several resources. For instance, a typical real-time application performs a control function which evaluates and processes sensor data in order to control an actuator according to a given control law. An exemplary mapping of such a real-time application to a platform with communication and computation elements is illustrated in Fig. 23.7.

In this section, it is shown how the global analysis integrates the dependencies of tasks on different resources into the analysis.

Consideration of Global Precedence Constraints

Tasks in an application can generally not be executed in an arbitrary order but have to be executed in a function-related order. The functional restrictions on the possible execution orders of tasks are expressed in form of *precedence constraints*. Precedence constraints can be described by a directed graph, the nodes representing the tasks, and the directed edges representing the directed execution dependencies. Paths in a precedence graph describe linear, branched, or even cyclic structures of dependencies.

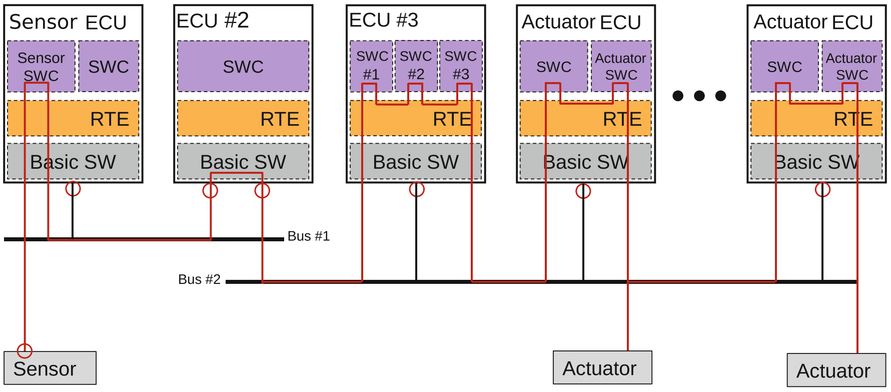

Fig. 23.7 Real-time application distributed over multiple computation and communication resources. ECU Electronic Control Unit, SWC Software Component, RTE Run-Time Environment

If two dependent tasks are mapped to two different resources, the timing behavior of those tasks is no longer exclusively determined by the local parameters. Consequently, the local view of a resource is no longer sufficient. To appropriately consider the precedence constraints of a pair of tasks where τ_i precedes τ_j ($\tau_i \rightarrow \tau_j$) in CPA, the termination event model of the predecessor task τ_i is used as the input event model of its successor task τ_j, i.e., the completion of one task triggers the activation of another. The termination behavior of task τ_i is bounded by the minimum output distance function, denoted as $\delta^-_{i,out}$, and the maximum output distance function, denoted as $\delta^+_{i,out}$. The distance functions naturally depend on the input (activation) event model ($\delta^-_{i,in}, \delta^+_{i,in}$) of task τ_i and the processing behavior of the resource [11, 36]

$$\delta^-_{i,out}(n) = \max\left\{\delta^-_{i,in} - J_{i,resp}, (n-1)\cdot d_{i,min}\right\} \tag{23.13}$$

$$\delta^+_{i,out}(n) = \delta^+_{i,in}(n) + J_{i,resp} \tag{23.14}$$

where $d_{i,min}$ is the minimum distance between any two terminations of task τ_i. A refined computation of the output models can be found in [43].

With the definition of inter-resource precedence constraints and the derivation of the best-case and worst-case output event models, the global analysis of the system can be performed.

Analysis Strategy

The global analysis is a timing verification step which sets the local verification problems in a system context where inter-resource precedence constraints are taken into account. As explained above, these inter-resource precedence constraints relate the local verification problems in such a manner that their input and output event models are linked. The relation of the local verification problems leads to a fixed point problem which converges if the propagation of outputs to inputs between related verification problems does not change the verification results any more. Therefore, the CPA procedure consists of two parts: First, the local analysis of the individual resources is performed in order to generate the initial output event models. Then in a global analysis step, the output event models are propagated through the system to the tasks which utilize them as input event models due to global precedence constraints. The local analysis is repeated under updated input parameters computing best-case and worst-case response times, jitter and required queue sizes. Due to possible circular dependencies between activation models, it might be necessary to repeat this process multiple times. This propagation of output event models and update of input event models is continued until the analysis results converge.

In the following, the detailed procedure of the global analysis is presented which is also illustrated in Fig. 23.8:

1. For every task τ_i which is activated by events in the system environment, the input event model $(\delta^-_{i,in}, \delta^+_{i,in})$ is initialized with the input event model of the respective external event source.
2. For every task τ_j which is part of a precedence path, the input event model $(\delta^-_{j,in}, \delta^+_{j,in})$ corresponds to the output event model of the predecessor task. If in the initial analysis run no output event model of the predecessor task is available, then the input event model $(\delta^-_{j,in}, \delta^+_{j,in})$ is initialized with the input event model of the predecessor task.
3. A local analysis is performed for each resource with the objective of deriving the task output event models. Additionally, it is checked if the local analysis results violate any constraints, for instance, the required absence of system overload or the guarantee of all task deadlines.
4. The computed task output event models are propagated through the system to the tasks which utilize them as input event models due to respective precedence constraints.
5. If the propagated output event models are identical to the input event models used in the previous local analysis, a global fixed point has been reached and the analysis terminates [46]. All timing constraints, particularly task deadlines and end-to-end path latencies, are checked. The classical approach to compute the (worst case) end-to-end path latencies, is to accumulate the individual (worst case) response times for each task along the path [21, 39, 47]. If any constraint is violated, the system is not schedulable.

   Otherwise, if no fixed point has been reached yet, the local analysis is repeated with the updated input event models.

If the CPA has successfully terminated, the Best-Case Response Times (BCRTs) and Worst-Case Response Times (WCRTs) of each task are known such that the response time behavior of every task can be safely bounded. Moreover, maximum required queue sizes are derived. Further system performance results can be derived using the supplementary analysis modules of CPA presented in Sect. 23.3.

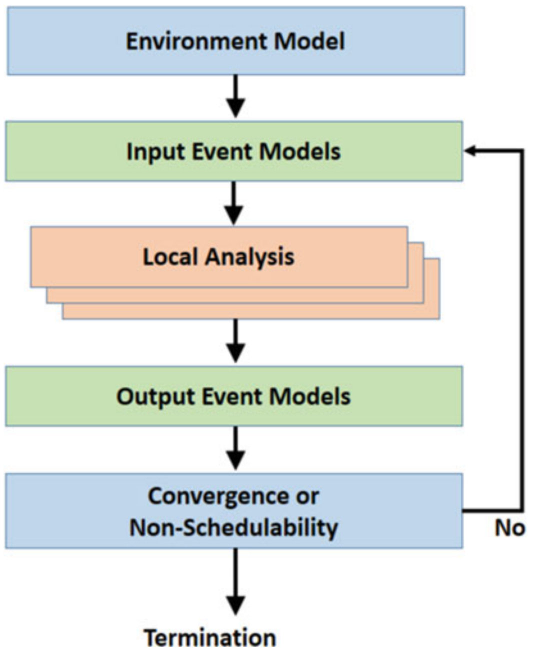

Fig. 23.8 In the system-level analysis, the local analyses are combined with a step to propagate updated output event models. This is repeated until the analysis converges or terminates due to abort criteria, e.g., constraint violation

23.3 Extensions

In the previous section, the basic CPA approach has been presented. It can be extended to cover more complex system models or to improve the analysis to compute tighter bounds for task response times or path latencies. In this section, CPA extensions are introduced which are able to deal with the usage of shared resources, the change of operation modi and the use of error handling protocols. Moreover an improved analysis for chained tasks is presented, and it is shown how CPA can be used to give formal timing guarantees for weakly-hard real-time systems.

23.3.1 Analysis of Systems with Shared Resources

In multi-core architectures several computational units have to share resources, such as the memory or data buses. For many of these shared resources concurrent or interrupted accesses are problematic, e.g., parallel write accesses to the same memory location can leave the accessed data corrupted or in an undefined state. Accesses to shared resources for which unconstrained accesses can be problematic are called *critical section*. Therefore, accesses to critical sections have to be isolated, i.e., locked by mutexes or semaphores. To ensure that only one process can have access to a critical section at a time, the accesses have to be modeled as

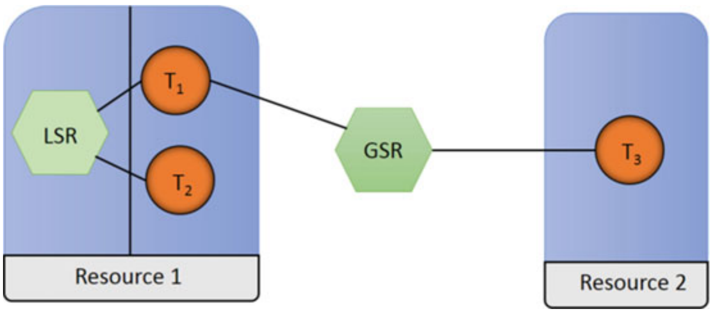

Fig. 23.9 Locally shared resources (LSR) can be only accessed by tasks from the same resource, globally shared resources (GSR) are available for any resource. As multiple tasks from different resources can attempt to access a GSR simultaneously, GSR require a different protection mechanism compared to LSR

non-preemptive sections [42]. If a task attempts to access a critical section, which has already been entered by another task, it has to wait until the current access has finished. This blocking time is further delaying the waiting task's response time as it cannot continue executing. Simply locking resources can lead to deadlocks or the priority inversion problem [22]. The automotive standard AUTOSAR, for instance, specifies different protocols for accessing locally shared resources (LSR) and globally shared resources (GSR) [3]. LSRs can only be accessed by tasks mapped on one computational resource, while GRSs can be accessed by any task in the system; see Fig. 23.9. Taking the different interference scenarios into account, AUTOSAR specifies to use the priority ceiling protocol for LSR and a spinlock mechanism for GSR [3].

When computing the WCRT of a task, the worst-case delay has to be assumed when accessing a shared resource. The scenario in which the worst-case delay occurs depends on the protocol used to protect the critical section. Generally, the worst-case delay occurs if τ_i tries to access a shared resource which is currently locked, and other tasks with a higher priority have accesses pending and continue to issue new ones. If the duration and number of accesses from all tasks is known, the maximum blocking a single access on a shared resource can be calculated [39]. Tasks often need to access a shared resource more than once. Calculating the independent worst-case delay for each individual access and accumulating the delay can provide an upper bound for the total blocking time a task can experience.

However, doing so can largely overestimate the actual possible interference, as the individual worst cases for each access are often mutually exclusive, i.e., blocking that can happen once is accounted for each single access.

To avoid this overestimation, the authors in [39] present a response time analysis combining arbitrarily activated tasks in multiprocessor systems with shared resources. Instead of the accumulation of the individual worst-case delays, they bounded the maximum possible interference that could occur during a given time interval, the task's processing time. The authors in [42] provided an analysis framework to calculate task timing behavior under the multi-core priority ceiling protocol. In [28] the authors improved on this by taking into account local scheduling dependencies allowing to analyze sets of functional dependent tasks. The analysis has been extended to allow non-preemptive scheduling for tasks in multi-core architectures with shared resources in [25].

23.3.2 Analysis of Systems Undergoing Mode Changes

Some real-time systems have to execute in multiple different modi, being able to adapt to the environment or the mission being executed in multiple stages. A plane, for instance, has to operate differently during start, landing, or flight, while in a car the engine control might turn off certain analysis features, depending on the engine speed [26]. With the capability to change its configuration, the system is able to run more efficiently as it only needs to execute tasks when necessary and disables functions when no longer needed. Being able to deactivate unrequired functions can prevent or reduce expensive hardware over-dimensionation. Switching between different configurations is called a *mode change*. This comes with the requirement to analyze and verify the safety not only under one static configuration, but of the different operational modes and also of the transition phases. For real-time systems, this includes ensuring that the system satisfies all its deadlines during each possible configuration.

A system is running in a *steady state* if it is executing in one mode without any residing influence from a previous mode change, only executing the tasks from the corresponding task set. If the system receives a Mode Change Request (MCR) the set of running tasks has to be changed from the current mode to the new one. In order to evaluate the transition phase, each task is classified according to the following categories:

1. *Old tasks* are tasks which were present in the previous mode but not in the new one. They are immediately terminated when the MCR occurs, i.e., any active or pending task instances are removed from the system.
2. *Finished or completed tasks* are tasks which were present in the previous mode but not in the new one. During the transition phase, these tasks are allowed to finish their active and pending task instances, but no new instances will be started.
3. *New or added tasks* are tasks which are present in the new mode, but not in the old one. They can represent updated tasks from the old mode, e.g., with changed execution time or activation pattern, or new functionalities.
4. *Unchanged tasks* are present during the old and new modes with identical properties, only in systems with periodicity.

If the system's mode change protocol allows *unchanged tasks*, it is referred to as *with periodicity*, and *without periodicity* if all task sets are disjunct.

When a MCR occurs, the system has to change from the current mode to the new one, remove *old* and *finished tasks* and add *new tasks*. If the system waits until all *finished tasks* have completed their execution before starting to schedule *new tasks*, the mode change protocol is called *synchronous*. Respectively, it is called *asynchronous*, if it starts scheduling *new tasks* right after the MCR has occurred, simultaneously with the last instances of *finished tasks*. Synchronous protocols ensure isolation between the modes and therefore do not require specific schedulability analyses for the transition phase. However, due to the delay introduced by the separation of the modes, synchronous protocols are not always feasible, if the transition has to be performed as fast as possible [27]. Asynchronous protocols, on the other hand, overcome this limitation and allow the simultaneous scheduling of tasks from the old and new mode. With an asynchronous protocol, the *new tasks* are added to the set of scheduled tasks, hence possibly increasing the resource utilization. As simply adding new tasks to the current set can lead to temporal overload on the resource, asynchronous protocols require specific schedulability analyses [27, 34, 35, 50]. For the remainder of this section, we will focus be on asynchronous protocols.

The CPA approach provides functionality to analyze the WCRTs of tasks and path latencies for a system in a certain mode. The transition periods can be modeled conservatively, by assuming that all tasks from the two (previous and new) modes are active simultaneously. In order to be able to model a transition phase, rules have to be defined regarding which transition phases can occur. The CPA extension relies on the assumption that the system executes in a steady state when the MCR occurs, i.e., new mode changes are not allowed to arise, while an older MCR is still exerting influence on the task activations. With this restriction, only tasks from exactly two mode sets need to be considered for a transition phase. The outcome from this is that not only the response times of tasks during steady states and the mode change phases are relevant but also the duration of these transition times. These *transient latencies* determine the distance to the next possible mode change [27].

The authors in [27] have shown that due to complex task dependencies the effects of a MCR are propagated delayed through the system, possibly causing feedback to the source of the MCR. They have shown how to bound the transition latency, by dividing it into local task transition latencies and global system transition latencies. In [26] the authors evaluate options for the design of multi-core real-time systems to minimize the impact of overly pessimistic measures taken in current practice.

23.3.3 Analysis of the Timing Impact of Errors and Error Handling

Safety-critical computing systems are usually designed to be fault tolerant toward communication and/or computation errors. However, each fault tolerance mechanisms incurs some time penalty because errors need first to be detected, and then an error correction or error masking measure has to be taken. To guarantee the correct timing behavior of a fault-tolerant safety-critical computing system, a formal performance analysis has to take these error-related, additional timing effects into account.

The consideration of timing overhead of fault tolerance mechanisms requires the adaptation of the local CPA. The stochastic nature of errors involves the introduction of a stochastic busy period; this is described in Sect. 23.3.3.2. Apart from timing overhead fault-tolerant systems have specific precedence constraints which result, for instance, from redundant task executions. In Sect. 23.3.3.1, an adapted worst-case response time analysis is briefly outlined for such fault-tolerant systems.

23.3.3.1 Computation Errors and Error Handling

A basic principle of detecting computational errors is to perform the same computation several times and to compare the results. A discrepancy in the computed results is an evidence for an occurred fault. In a multi- or many-core processing system, it is possible to parallelize the redundant computations or, respectively, the execution of the redundant tasks. Such a fault tolerance approach leads to fork-join task graphs, where *forking* means the parallel execution of replicated tasks and *joining* means synchronization and the comparison of results. In [5] a strategy is presented how to derive worst-case response times for tasks in a task set with fork-join precedence constraints, so that the timing impact of replicated and parallelized computations can be evaluated.

23.3.3.2 Communication Errors and Error Handling

Unreliable communication links in data buses or packet-switched networks introduce bit errors in the digitally transmitted information. The occurrence of bit errors can be modeled by stochastic processes which often use the average bit error rate or packet error rate as an important parameter. If the assumption of independent bit errors is justified, the Bernoulli process or its approximation as a Poisson process is a classical modeling choice. If the probability of a bit error depends on past events, a state-aware Markov process is more appropriate [44].

For multi-master data buses and point-to-point communication, the detection of bit errors in transmitted frames at the receiver is typically based on error detecting codes. If a an error has been detected and signaled, a retransmission of the the corrupted or lost frame is initiated and the system is set back to a consistent state. Since bit errors can occur arbitrarily often albeit with a very low probability, the computation of a worst-case response time which includes an excessive detection, signaling, and correction time overhead is meaningless. A probabilistic scheduling guarantee, however, in the form of an exceedance function which specifies an upper bound on the probability that a task instance exceeds a reference response time value, is far more expressive. In [6], a probabilistic scheduling analysis is presented for a fault-tolerant multi-master/point-to-point communication system with non-preemptive fixed priority arbitration which is, for instance, applicable to CAN. The analysis computes first the worst-case response time of a frame under $1 \ldots K$ errors and the corresponding probabilities, and then derives an exceedance function by summing up the probabilities for all error scenarios which have a worst-case response time smaller or equal than the reference response time. In [4], an improved approach is presented which relies on stochastic frame transmission times. A stochastic frame transmission time is composed of the error-free frame transmission time and the stochastic overhead for error signaling and correction.

Stochastic frame transmission times give rise to stochastic busy periods from which stochastic response times and a less pessimistic exceedance function can be derived.

The performance analysis of switched real-time networks, both on-chip and off-chip networks, is treated in [7]. The network switches are assumed to employ a fixed priority-based arbitration scheme, and an end-to-end error protocol in form of an Automatic Repeat Request (ARQ) scheme is investigated. In the ARQ scheme, the sender buffers a sent packet until an Acknowledgement (ACK) message is received. If no ACK arrives at the sender in a given time interval, a timeout occurs and the buffered packet is retransmitted. The detection of corrupted packets at the receiver is typically based on error detecting codes. Both corrupted and lost packets are signaled to the sender by an omitted acknowledgement so that a retransmission is implicitly triggered. Variants of this type of ARQ error handling protocol are selective ARQ, stop-and-wait ARQ, and Go-back-N ARQ.

23.3.4 Refined Analysis of Task Chains

Real-time applications are usually not implemented as single tasks, but rather as a set of logical dependent tasks, as shown in Fig. 23.7. The tasks within an application are typically ordered and presented as a Directed Acyclic Graph (DAG), representing the logical order of execution. Within such a graph any logical dependent tasks form a *task path* or *task chain*. Sensor-actuator chains in automotive or avionic systems, for example, are distributed within the system as the components are physically apart, or information needs to be gathered in a central instance to perform decision-making. Multimedia applications on the other hand are often pipelined in order to process media streams more efficiently. To be able to take advantage of the parallelization of applications, the analyses methods need to support task paths and provide mechanisms to efficiently analyze the dependencies between tasks.

The basic CPA approach supports the latency analysis of task paths, as described in Sect. 23.2.2. The conservative approach is to compute the WCRT of each task which is an element of the considered task path and to derive the path latency by accumulating the individual WCRTs [21, 47]. While this simple accumulation provides an upper bound for the path latency, it is pessimistic if local worst-case scenarios within the same path are mutually exclusive.

In [38] the authors consider the communication between application threads and the corresponding precedence constraints in the resulting task graphs in order to improve the local WCRTs. They exclude infeasible worst-case scenarios for logically dependent tasks on the same SPP-scheduled resource by extending the scope of the busy period approach. By leveraging the particular semantics – including the distinction of synchronous and asynchronous communication – they were able to significantly reduce pessimism and the analysis complexity, resulting in a faster execution of the local analysis.

Another situation, which can lead to especially large local WCRTs, occurs when a task is activated with a burst. Bursts can potentially occur anywhere within the system, but the same burst cannot occur on all resources at once. Accumulating the local WCRTs from a path with a bursty activation event model can therefore lead to a significant overestimation. Pessimistically bounded WCRTs can translate into over-dimensioning of the required hardware components and hence increased costs [39, 41]. This issue is captured in the 'pay burst only once' problem [15].

In order to reduce the impact of the pay burst only once problem, the authors in [40] proposed a method to identify relevant combinations of local response times to derive a tighter worst-case path latency. They provided a methodology for computing path latencies, considering pipelined chains of tasks with transient overload along the path. This approach was extended in [41] by enabling the analysis of a wider variety of system topologies and including functional cycles and nonfunctional dependencies.

A similar method can be used to improve the analysis of Ethernet networks. In Ethernet networks, different data streams often need to be transferred with the same priority, as Ethernet switches only have a limited range of priorities. Streams with the same priority are queued and transferred according to FIFO scheduling. Therefore, for the individual worst-case analyses, each other stream with the same priority has be assumed to arrive simultaneously with the analyzed one but to be served first [13].

In [48] the authors have shown how the analysis of Ethernet networks can be improved by limiting the interference of tasks with the same priority that share more than one consecutive switch; see Fig. 23.10. Streams with the same priority that arrive simultaneously on one switch cannot arrive simultaneously on the following switch, as only one stream can be transfer at a time. This dependency can be exploited to provide tighter bounds for end-to-end path delays.

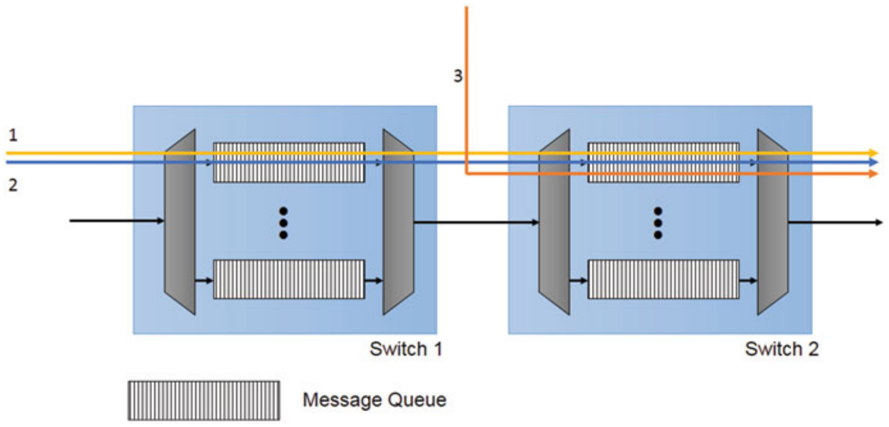

Fig. 23.10 The analysis of Ethernet networks can be improved, if dependencies between streams with the same priority – here stream 1 and 2 – are exploited. This can be done, if these streams are lead through the same two consecutive switches

23.3.5 Timing Verification of Weakly-Hard Real-Time Systems

Hard real-time computing systems require per definition that each instance of a task meets its deadline, whereas weakly-hard real-time systems tolerate occasional but in number and distribution precisely bounded deadline misses of tasks [8]. For instance, a weakly-hard system may require that a given task misses not more than m deadlines in any sequence of k consecutive task activations. The tolerance toward occasional deadline misses of tasks is usually based on the characteristics of the implemented real-time applications. Prevalent real-time applications like control functions, monitoring functions, and multimedia functions have shown to be robust against occasional but bounded sample or frame losses which can be interpreted as consequences of missed task deadlines. This robustness allows real-time applications to continue in safe operation even in the presence of limited transient overload [8, 16]. The analysis of weakly-hard real-time systems which is implemented as an extension of CPA, called Typical Worst-Case Analysis (TWCA) [2,31,32,53], provides formal guarantees for the compliance with weakly-hard real-time requirements for a wide range of system configurations. It scales to real-sized systems and provides a good computational efficiency [30].

TWCA assumes that each task has a typical behavior, e.g., a periodic activation pattern, which is captured in a typical activation model. In rare circumstances, a task may additionally experience nontypical activations, e.g., sporadic activations [31] or sporadic bursts [32], and then can be described by its worst-case activation model. The distance between the typical and the worst-case activation model of a task is captured by the so-called overload model. In the *typical worst case*, which occurs if all tasks show their densest pattern of typical activations and demand their maximum execution time, no deadlines are missed. In the *worst case*, which occurs if all tasks show their densest pattern of typical and nontypical activations and demand their maximum execution time, deadlines will be missed due to overload.

In two classical CPAs, the worst-case response times for both the typical worst case and the worst-case behavior of the system are computed: *Typical Worst-Case Response Time (TWCRT)* and *WCRT* for all tasks. If the WCRT of a task τ_i exceeds its deadline, a *deadline miss model* is computed which indicates the maximum number of observable deadline misses m in any sequence of k of consecutive instances of task τ_i. The computation of the deadline miss model relies on three main impact factors which need to be derived for each task interfering with task τ_i. Firstly, the overload model which is an indicator for how often nontypical activations can be encountered in a given time interval. Secondly, the longest time interval during which overload activations can impact the behavior of a sequence of k of consecutive instances of task τ_i. And thirdly, the maximum number of deadline misses of task τ_i that can be traced back to one overload activation. The computed deadline miss model for a task τ_i can be tightened if the number and distribution of overload activations, which induce the maximum number of deadline misses of task τ_i in any sequence of k consecutive instances, are bounded as precisely as possible [53].

23.3.6 Further Contributions

This chapter could only introduce the main functions of CPA. There are many more contributions that exceed the available space and should only be mentioned.

The robustness of a system, for instance, determines how sensitive the system reacts to changes in, e.g., execution and transmission delays, input data rates, or CPU clock cycles. A sensitivity analysis determines the influence of input data, or system configurations on the system robustness. The authors of [19, 20, 33] have shown how to identify critical components for the system robustness and how to optimize the platform. In many embedded systems, such as automotive systems, sensors are measuring the system behavior with a set period. If data is accessed periodically, but the communication path, e.g., a FlexRay bus, is transmitting the data with a different period, additional delay can occur due to the period mismatch. In [14] the authors discuss different end-to-end timing scenarios with a focus on register-based communication, taking different aspects of end-to-end delays into account.

23.4 Conclusion

In this chapter the compositional performance analysis approach has been presented. CPA provides a scalable framework to perform timing analysis of distributed embedded systems. It is widely used in the industrial development processes of real-time systems, especially in the automotive field where it is extensively proven in practice but also in avionics and even in networks-on-chip [12]. Numerous extensions exist to cover more complex applications, different applications of timing analysis in sensitivity, and robustness as well as error analysis.

Acknowledgments The project leading to this overview has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 644080 as well as from the German Research Foundation (DFG) under the contract number TWCA ER168/30-1.

References

1. AbsInt. aiT. http://www.absint.com/ait/. Accessed 24 Feb 2016
2. Ahrendts L, Hammadeh ZAH, Ernst R (2016) Guarantees for runnable entities with heterogeneous real-time requirements (to appear). In: Design, automation & test in Europe conference & exhibition (DATE 2016)
3. Autosar (2011) Specification of operating system, 5.0.0 edn. http://autosar.org/download/R4.0/AUTOSAR_SWS_OS.pdf
4. Axer P, Ernst R (2013) Stochastic response-time guarantee for non-preemptive, fixed-priority scheduling under errors. In: 50th ACM/EDAC/IEEE design automation conference (DAC 2013), pp 1–7. doi:10.1145/2463209.2488946
5. Axer P, Quinton S, Neukirchner M, Ernst R, Dobel B, Hartig H (2013) Response-time analysis of parallel fork-join workloads with real-time constraints. In: 25th Euromicro conference on real-time systems (ECRTS 2013), pp 215–224. doi:10.1109/ECRTS.2013.31

6. Axer P, Sebastian M, Ernst R (2012) Probabilistic response time bound for CAN messages with arbitrary deadlines. In: Design, automation test in Europe conference exhibition (DATE 2012), pp 1114–1117. doi:10.1109/DATE.2012.6176662
7. Axer P, Thiele D, Ernst R (2014) Formal timing analysis of automatic repeat request for switched real-time networks. In: 9th IEEE international symposium on industrial embedded systems (SIES 2014), pp 78–87. doi:10.1109/SIES.2014.6871191
8. Bernat G, Burns A, Liamosi A (2001) Weakly hard real-time systems. IEEE Trans Comput 50(4):308–321. doi:10.1109/12.919277
9. Bygde S (2010) Static WCET analysis based on abstract interpretation and counting of elements. Mälardalen University, Västerås
10. Davis RI, Burns A, Bril RJ, Lukkien JJ (2007) Controller area network (CAN) schedulability analysis: refuted, revisited and revised. Real-Time Syst 35(3):239–272. doi:10.1007/s11241-007-9012-7
11. Diemer J (2016) Predictable architecture and performance analysis for general-purpose networks-on-chip. Technische Universität Braunschweig, Braunschweig
12. Diemer J, Ernst R (2010) Back suction: service guarantees for latency-sensitive on-chip networks. In: Proceedings of the 2010 fourth ACM/IEEE international symposium on networks-on-chip (NOCS 2010). IEEE Computer Society, Washington, DC, pp 155–162. doi:10.1109/NOCS.2010.38
13. Diemer J, Rox J, Ernst R, Chen F, Kremer KT, Richter K (2012) Exploring the worst-case timing of ethernet AVB for industrial applications. In: Proceedings of the 38th annual conference of the IEEE industrial electronics society, Montreal. http://dx.doi.org/10.1109/IECON.2012.6389389
14. Feiertag N, Richter K, Nordlander J, Jonsson J (2008) A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics. In: Proceedings of the IEEE real-time system symposium – workshop on compositional theory and technology for real-time embedded systems, Barcelona, 30 Nov 2008
15. Fidler M (2003) Extending the network Calculus Pay bursts only once principle to aggregate scheduling. In: Proceedings of the quality of service in multiservice IP networks: second international workshop, QoS-IP 2003 Milano, 24–26 Feb 2003. Springer, Berlin/Heidelberg, pp 19–34. doi:10.1007/3-540-36480-3-2
16. Frehse G, Hamann A, Quinton S, Wöhrle M (2014) Formal analysis of timing effects on closed-loop properties of control software. In: 35th IEEE real-time systems symposium 2014 (RTSS), Rome. https://hal.inria.fr/hal-01097622
17. GmbH S. SymTA/S and traceanalyzer. https://www.symtavision.com/products/symtas-traceanalyzer/. Accessed 29 Jan 2016
18. Gresser K (1993) An event model for deadline verification of hard real-time systems. In: Proceedings of the fifth Euromicro workshop on real-time systems, 1993, pp 118–123. doi:10.1109/EMWRT.1993.639067
19. Hamann A, Racu R, Ernst R (2006) A formal approach to robustness maximization of complex heterogeneous embedded systems. In: Proceedings of the international conference on hardware – software codesign and system synthesis (CODES), Seoul
20. Hamann A, Racu R, Ernst R (2007) Multidimensional robustness optimization in heterogeneous distributed embedded systems. In: Proceedings of the 13th IEEE real-time and embedded technology and applications symposium
21. Henia R, Hamann A, Jersak M, Racu R, Richter K, Ernst R (2005) System level performance analysis – the symTA/S approach. IEE Proc Comput Digit Tech 152(2):148–166. doi:10.1049/ip-cdt:20045088
22. Lampson BW, Redell DD (1980) Experience with processes and monitors in mesa. Commun ACM 23(2):105–117. doi:10.1145/358818.358824
23. Lehoczky JP (1990) Fixed priority scheduling of periodic task sets with arbitrary deadlines. In: Proceedings of the 11th real-time systems symposium, pp 201–209. doi:10.1109/REAL.1990.128748
24. Liu JW (2000) Real-time systems. Prentice Hall, Englewood Cliffs

25. Negrean M, Ernst R (2012) Response-time analysis for non-preemptive scheduling in multi-core systems with shared resources. In: Proceedings of the 7th IEEE international symposium on industrial embedded systems (SIES), Karlsruhe
26. Negrean M, Ernst R, Schliecker S (2012) Mastering timing challenges for the design of multi-mode applications on multi-core real-time embedded systems. In: 6th international congress on embedded real-time software and systems (ERTS), Toulouse
27. Negrean M, Neukirchner M, Stein S, Schliecker S, Ernst R (2011) Bounding mode change transition latencies for multi-mode real-time distributed applications. In: 16th IEEE international conference on emerging technologies and factory automation (ETFA 2011), Toulouse. http://dx.doi.org/10.1109/ETFA.2011.6059009
28. Negrean M, Schliecker S, Ernst R (2009) Response-time analysis of arbitrarily activated tasks in multiprocessor systems with shared resources. In: Proceedings of the Design, Automation, and Test in Europe (DATE), Nice. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5090720
29. Pellizzoni R, Schranzhofer A, Chen JJ, Caccamo M, Thiele L (2010) Worst case delay analysis for memory interference in multicore systems. In: Design, automation test in Europe conference exhibition (DATE 2010), pp 741–746. doi:10.1109/DATE.2010.5456952
30. Quinton S, Ernst R, Bertrand D, Yomsi PM (2012) Challenges and new trends in probabilistic timing analysis. In: Design, automation & test in Europe conference & exhibition (DATE 2012), pp 810–815. doi:10.1109/DATE.2012.6176605
31. Quinton S, Hanke M, Ernst R (2012) Formal analysis of sporadic overload in real-time systems. In: Design, automation test in Europe conference exhibition (DATE), pp 515–520. doi:10.1109/DATE.2012.6176523
32. Quinton S, Negrean M, Ernst R (2013) Formal analysis of sporadic bursts in real-time systems. In: Design, automation test in Europe conference exhibition (DATE), pp 767–772. doi:10.7873/DATE.2013.163
33. Racu R, Hamann A, Ernst R (2008) Sensitivity analysis of complex embedded real-time systems. Real-Time Syst 39:31–72
34. Rafik Henia RE (2007) Scenario aware analysis for complex event models and distributed systems. In: Proceedings real-time systems symposium
35. Real J, Crespo A (2004) Mode change protocols for real-time systems: a survey and a new proposal. Real-Time Syst 26(2):161–197. doi:10.1023/B:TIME.0000016129.97430.c6
36. Richter K (2005) Compositional scheduling analysis using standard event models. Ph.D. thesis, TU Braunschweig, IDA
37. Richter K, Jersak M, Ernst R (2003) A formal approach to MpSoC performance verification. Computer 36(4):60–67
38. Schlatow J, Ernst R (2016) Response-time analysis for task chains in communicating threads. In: 22nd IEEE real-time embedded technology and applications symposium (RTAS 2016), Vienna
39. Schliecker S (2011) Performance analysis of multiprocessor real-time systems with shared resources. Ph.D. thesis, Technische Universität Braunschweig, Braunschweig. http://www.cuvillier.de/flycms/de/html/30/-UickI3zKPS76fkY=/Buchdetails.html
40. Schliecker S, Ernst R (2008) Compositional path latency computation with local busy times. Technical report IDA-08-01, Technical University Braunschweig, Braunschweig
41. Schliecker S, Ernst R (2009) A recursive approach to end-to-end path latency computation in heterogeneous multiprocessor systems. In: Proceedings of the 7th international conference on hardware software codesign and system synthesis (CODES-ISSS). ACM, Grenoble. http://doi.acm.org/10.1145/1629435.1629494
42. Schliecker S, Negrean M, Ernst R (2009) Response time analysis in multicore ECUs with shared resources. IEEE Trans Ind Inf 5(4):402–413. http://ieee-ies.org/tii/issues/iit09_4.shtml
43. Schliecker S, Rox J, Ivers M, Ernst R (2008) Providing accurate event models for the analysis of heterogeneous multiprocessor systems. In: Proceedings of the 6th IEEE/ACM/IFIP international conference on Hardware/Software codesign and system synthesis. ACM, pp 185–190

44. Sebastian M, Axer P, Ernst R (2011) Utilizing hidden Markov models for formal reliability analysis of real-time communication systems with errors. In: IEEE 17th Pacific Rim international symposium on dependable computing (PRDC 2011), pp 79–88. doi:10.1109/PRDC.2011.19
45. Service ASC. Arinc 600 series. http://store.aviation-ia.com/cf/store/catalog.cfm?prod_group_id=1&category_group_id=3. Accessed 16 Mar 2016
46. Stein S, Diemer J, Ivers M, Schliecker S, Ernst R (2008) On the convergence of the symta/s analysis. Technical report, TU Braunschweig, Braunschweig
47. Sun J, Liu JWS (1995) Bounding the end-to-end response time in multiprocessor real-time systems. In: Proceedings of the third workshop on parallel and distributed real-time systems, 1995, pp 91–98. doi:10.1109/WPDRTS.1995.470502
48. Thiele D, Axer P, Ernst R (2015) Improving formal timing analysis of switched ethernet by exploiting fifo scheduling. In: Design automation conference (DAC), San Francisco
49. Thiele L, Chakraborty S, Naedele M (2000) Real-time calculus for scheduling hard real-time systems. In: Proceedings of the IEEE international symposium on circuits and systems. Emerging technologies for the 21st century, 2000, pp 101–104. doi:10.1109/ISCAS.2000.858698
50. Tindell KW, Burns A, Wellings AJ (1992) Mode changes in priority preemptively scheduled systems. In: Real-time systems symposium, 1992, pp 100–109. doi:10.1109/REAL.1992.242672
51. Wilhelm R, Engblom J, Ermedahl A, Holsti N, Thesing S, Whalley D, Bernat G, Ferdinand C, Heckmann R, Mitra T, Mueller F, Puaut I, Puschner P, Staschulat J, Stenstrom P (2008) The worst-case execution time problem – overview of methods and survey of tools. ACM Trans Embed Comput Syst 7(3):Art. 36
52. Wilhelm R, Engblom J, Ermedahl A, Holsti N, Thesing S, Whalley D, Bernat G, Ferdinand C, Heckmann R, Mitra T, Mueller F, Puaut I, Puschner P, Staschulat J, Stenström, P (2008) The worst-case execution-time problem&mdash;overview of methods and survey of tools. ACM Trans Embed Comput Syst 7(3):36:1–36:53. doi:10.1145/1347375.1347389
53. Xu W, Hammadeh Z, Quinton S, Kröller A, Ernst R (2015) Improved deadline miss models for real-time systems using typical worst-case analysis. In: 27th Euromicro conference on real-time systems (ECRTS), Lund