Int J Softw Tools Technol Transfer (2006) 8:649–667
DOI 10.1007/s10009-006-0019-5

SPECIAL SECTION ON QUANTITATIVE ANALYSIS OF REAL-TIME EMBEDDED SYSTEMS

System architecture evaluation using modular performance analysis: a case study

Ernesto Wandeler · Lothar Thiele · Marcel Verhoef · Paul Lieverse

Published online: 20 July 2006

Abstract Performance analysis plays an increasingly important role in the design of embedded real-time systems. Time-to-market pressure in this domain is high while the available implementation technology is often pushed to its limit to minimize cost. This requires analysis of performance as early as possible in the life cycle. Simulation-based techniques are often not sufficiently productive. We present an alternative, analytical, approach based on Real-Time Calculus. Modular performance analysis is presented through a case study in which several candidate architectures are evaluated for a distributed in-car radio navigation system. The analysis is efficient due to the high abstraction level of the model, which makes the technique suitable for early design exploration.

This work has been carried out as part of the BODERC project under the responsibility of the Embedded Systems Institute.

E. Wandeler (✉) · L. Thiele
ETH Zürich, Information Technology and Electrical Engineering Department, Computer Engineering and Networks Laboratory (TIK), Gloriastrasse 35, 8092 Zürich, Switzerland
e-mail: wandeler@tik.ee.ethz.ch

L. Thiele
e-mail: thiele@tik.ee.ethz.ch

M. H. G. Verhoef
Chess Information Technology B.V., Nieuwe Gracht 13, 2011 NB, Haarlem, The Netherlands

M. H. G. Verhoef
Radboud University Nijmegen, Institute for Computing and Information Sciences, Nijmegen, The Netherlands
e-mail: marcel.verhoef@chess.nl

P. Lieverse
Siemens VDO Trading B.V., Luchthavenweg 48, 5657 EB, Eindhoven, The Netherlands
e-mail: paul.lieverse@siemens.com

1 Introduction

Today's embedded systems industry is mainly characterized by the extremely high time-to-market pressure. In particular, in the areas of consumer and mass-market electronics, such as mobile telephones, product life cycles are measured in weeks and months rather than years. This tremendous pressure has caused a shift in the way these embedded systems are designed. Recently, industrial focus was mainly on improving the efficiency of the design process and raising the quality of the engineered product. Introduction of advanced tools for both hardware and software design, the adoption of UML, and the implementation of quality improvement programs to reach higher levels of the capability maturity model (CMM) are just a few examples of these efforts.

Industry has now realized that these investments are insufficient to become and stay competitive because the gains achieved still do not meet the required time-to-market targets. The main problem is that product development times often exceed the technology innovation cycle. At the time the product is ready, it may be outdated altogether or it may be able to be produced at lower cost using some other technology that has just become available. Both scenarios are equally undermining for the business case of the product. Therefore, industrial focus is shifting from improving the system implementation phase towards improving the system design phase. The capability to assess new ideas quickly, either inspired by novel technology or by changed market conditions, is essential. This evaluation process shall be light-weight, fast, and reliable such that new products

will meet customer requirements can be produced faster and at minimum cost.

The main question to solve in the early stages of the design is whether or not a particular distribution of functionality over a proposed system decomposition (the so-called *system architecture*) will meet the overall requirements. This is a hard problem because, paradoxically, there are still many unknowns that might have great impact on the system performance. *Design space exploration* and *system level performance analysis* are proposed as solutions to this problem and several techniques have been developed to implement these concepts. Note that *performance analysis* is not necessarily restricted to the timing aspects of the system, although it will be the main focus of this paper.

For example, SystemC [7] is such a system-level design technique. It consists of a modeling language (a C++ class library), a simulation-based kernel, and a verification library. These components are used to build and exercise executable models at the system level. The development process is improved mainly by raising the level of abstraction from traditional VHDL or Verilog-based hardware design and by providing a platform for hardware/software co-design. Hence, it takes less time to construct a model and analyze its fitness for purpose.

While this is already a big step forward, it still has some major drawbacks, as is the case with other simulation-based techniques as well, for example Matlab/Simulink. First of all, constructing a simulation model is, in general, not a trivial task. Quite some effort is needed to compose a model especially if it needs to cover a wide range of architectural derivatives. Furthermore, these models need to be sufficiently detailed to be of value to support the major design decisions; often this has repercussions on the amount of time needed to execute the models and analyze the simulation results. Of course, libraries of building blocks can be developed to construct new simulation models more efficiently. But the initial cost of creating these building blocks remains and this investment is only worthwhile if the library is actually used sufficiently often. Application of simulation-based techniques is therefore often po stponed to later stages of the design process because they are considered too expensive due to these long lead times.

This paper proposes an approach to the problem of performance analysis in the very early phases of the system life cycle that does not have the disadvantages described above. As mentioned before, one of the key problems is that simulation based techniques require a detailed description of the actual computation that is performed. Our approach is to characterize functionality merely by describing incoming and outgoing event rates, message sizes, and execution times. Similarly, capacity of computing and communication resources can be described. Real-Time Calculus [17] is then used to compute hard upper and lower bounds of the system performance. This calculation can be done very efficiently, because the model is at a much higher level of abstraction than a typical simulation-based model.

First, we introduce modular performance analysis (MPA) with Real-Time Calculus. This technique is then applied to a case study of a distributed in-car radio navigation system. The case study is described using sequence diagrams which we annotate to feed the analysis method with data. Then, several architectural derivatives are proposed and the analysis results are discussed. Finally, we draw some conclusions from the case study and suggest directions for future research.

Related work. Two recent publications give an extensive overview of various performance evaluation methodologies that are currently available. Gries [6] lists methodologies from the area of system-on-chip design, where the focus is on evaluating the performance of combined hardware–software systems. However, the techniques used in this domain are often more focused on hardware than on software.

Balsamo et al. [2] compares methodologies that are coming from the area of software engineering. There, the focus is on evaluating the performance of software architectures on a given hardware platform. The design or optimization of the hardware is typically not considered.

The case study presented in this paper is based on the Real-Time Calculus presented in [3], which is also mentioned in the overview of Gries [6]. While the Real-Time Calculus has until now mainly been applied for data-dominated systems, such as network processors, we have applied it to a more control-oriented and software-intensive system. Real-Time Calculus is based on the well-known Network Calculus [10] which is in turn based on max-plus algebra [1]. Note that max-plus algebra has been applied to a wide variety of problems, including scheduling and performance analysis of dynamic discrete event systems, see *http://www.maxplus.org*. The method proposed in this paper uses a notation that is close to current engineering practice, while maintaining the sound mathematical basis provided by the Real-Time Calculus.

Some of the techniques used in this paper are inspired by other software-oriented performance analysis methodologies mentioned in [2]. For example, software performance evaluation (SPE [16]), also annotates sequence diagrams with information on resource utilization. However, the analysis is done differently: probabilistic queuing networks are used as the underlying

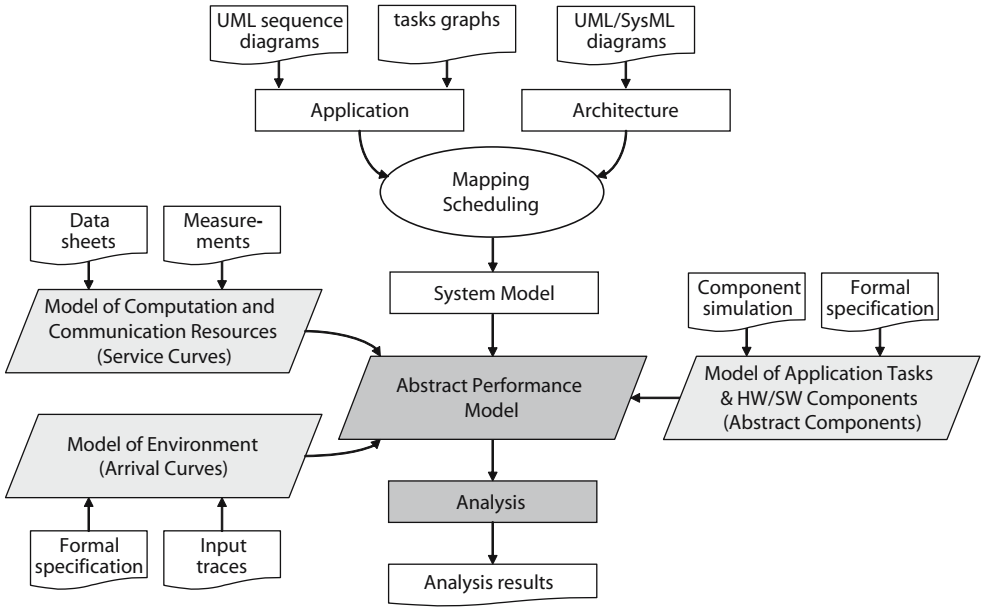

Fig. 1 Elements of modular performance analysis

analysis technique, whereas we use Real-Time Calculus, which comes from the family of deterministic queuing theories. Cortelessa et al. [4] describes a methodology in which a probabilistic queuing network-based performance model is automatically derived from UML use case diagrams, sequence diagrams, and deployment diagrams. In this paper, the models are still constructed by hand but this could certainly be automated as shown by [16] and [4] and therefore we do not emphasize the formalization of the informal requirements more than necessary here.

Contribution of this paper. We show that Real-Time Calculus can be used to evaluate system architectures effectively. The technique provides a high level of abstraction which allows for fast model evaluation, while still producing results that are sufficiently accurate. Furthermore, the typical set-up costs associated with simulation-based models are avoided. It is possible to find weak spots in the design even with little available data – circumstances that are typical for the early stages of the system life cycle. For example, we show that sensitivity analysis can be performed on the models to determine robustness of the design. Modular performance analysis supports the system architect in his design activities by providing timely feedback at little investment. Confidence in the design can be increased and risks are reduced if this technique is applied. We believe that this has a positive effect on both the development time and quality of the product.

2 MPA and Real-Time Calculus

In the domain of communication networks, powerful abstractions have been developed to model flow of data through a network. In particular Network Calculus [10] provides the means to deterministically reason about timing properties of data flows in queuing networks, and can be viewed as a deterministic queuing theory. Real-Time Calculus [17] extends the concepts of Network Calculus to the domain of real-time embedded systems, and in [3] a unifying approach to MPA with Real-Time Calculus has been proposed. It is based on a general event and resource model, allows for hierarchical scheduling and arbitration, and can take computation as well as communication resources into account.

With Real-Time Calculus, hard upper and lower bounds can be computed to various performance criteria in a real-time system, such as end-to-end delays of event streams, or buffer requirements. Real-Time Calculus hence qualifies to analyze hard real-time systems. This clearly distinguishes Real-Time Calculus from any probabilistic performance estimation methods, or from performance estimation through simulation.

Figure 1 presents an overview of the basic elements of MPA, and the relations between them. We basically follow the well-known *Y-chart* scheme as proposed in [9]. The central idea of MPA is to first build an abstract performance model of the concrete system that bundles all information needed for performance analysis with Real-Time Calculus. The abstract performance

model unifies essential information about the environment, about the available computation and communication resources, about the application tasks (or dedicated HW/SW components), as well as about the system architecture itself.

The environment models describe how a system is being used by the environment: how often will system functions be called, how much data is provided as input to the system, and how much data is generated by the system back to its environment. Environment models can be derived from formal behavior specifications or for example from measured input traces. We will show how UML sequence diagrams can be used to formalize these aspects.

The resource models provide information about the properties of the computing and communication resources that are available within a system, such as processor speed and communication bus bandwidth. This information is typically found in data sheets or benchmarks, or can be obtained from measurements on existing systems.

The application task (or dedicated HW/SW component) models provide information about the processing semantics that is used to execute the various application tasks or to run the dedicated HW/SW components.

Finally, the system model captures information about the applications and the available hardware architecture, and it also defines the mapping of tasks to computation or communication resources of the hardware architecture and specifies the scheduling and arbitration schemes used on these resources. In Sect. 3, we will elaborate in more detail on how to specify this information using UML and other methods, and we will show how to construct abstract performance models using this information. We will present the model of the environment in Sect. 2.1 and the model of computation and communication resources in Sect. 2.2. The model of application tasks and dedicated HW/SW components and construction of the abstract performance model is presented in Sect. 2.3 and 2.4. Finally, we explore the analysis of the abstract performance models in Sect. 2.5.

2.1 Arrival curves: a general event stream model

A trace of an event stream can conveniently be described by means of a cumulative function $R(t)$, defined as the number of events seen on the event stream in the time interval $[0,t)$. While any R always describes *one* concrete trace of an event stream, a tuple $\overline{\alpha}(\Delta) = \left[\overline{\alpha}^u(\Delta), \overline{\alpha}^l(\Delta)\right]$ of upper and lower *arrival curves* [5] provides an abstract event stream model, representing *all* possible traces of an event stream.

For this, the upper arrival curve $\overline{\alpha}^u(\Delta)$ provides an upper bound on the number of events that are seen on the event stream in *any* time interval of length Δ, and analogously, the lower arrival curve $\overline{\alpha}^l(\Delta)$ provides a lower bound on the number of events in a time interval Δ. In other words, in any time interval of length Δ there will always arrive at least $\overline{\alpha}^l(\Delta)$ and at most $\overline{\alpha}^u(\Delta)$ events on an event stream that is modeled by $\overline{\alpha}(\Delta)$.

Arrival curves were first introduced in [5], and are defined as follows:

Definition 1 (arrival curves) Let $R(t)$ denote the number of events that arrive on an event stream in the time interval $[0,t)$. Then, R, $\overline{\alpha}^u$ and $\overline{\alpha}^l$ are related to each other by the following inequality

$$\overline{\alpha}^l(t-s) \leq R(t) - R(s) \leq \overline{\alpha}^u(t-s), \quad \forall s < t \tag{1}$$

with $\overline{\alpha}^l(0) = \overline{\alpha}^u(0) = 0$. □

Arrival curves substantially generalize the classical representation of standard event arrival patterns such as sporadic, periodic, periodic with jitter, or others. Besides being able to represent any event stream with known deterministic timing behavior that is obtained from a system specification, it is also possible to determine arrival curves corresponding to any finite length event stream trace, obtained for example from observation or simulation. For this, a sliding window approach can be used.

Example 1 In literature, standard event arrival patterns are often specified by a parameter triple (p,j,d), where p denotes the period, j the jitter, and d the minimum inter-arrival distance of events in the modeled stream. Event streams that are specified using these parameters can directly be modeled by the following arrival curves:

$$\overline{\alpha}^l(\Delta) = \left\lfloor \frac{\Delta - j}{p} \right\rfloor \tag{2}$$

$$\overline{\alpha}^u(\Delta) = \min\left\{ \left\lceil \frac{\Delta + j}{p} \right\rceil, \left\lceil \frac{\Delta}{d} \right\rceil \right\} \tag{3}$$

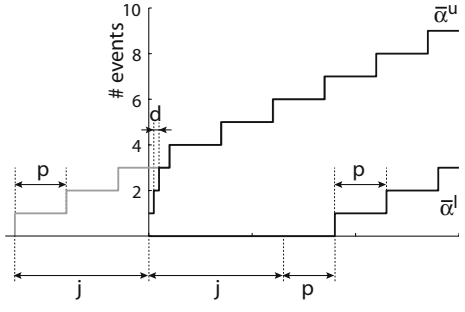

Fig. 2 Arrival curves from (p, j, d)

In Fig. 2, the relation between these parameters and the corresponding arrival curves is graphically depicted. Note that in this particular example the jitter is much greater than the period which is typical for a so-called event streams with bursts. This also explains the steep ascend at the beginning of the upper arrival curve.

Figure 3 shows some typical examples of arrival curves. The arrival curves in Fig. 3a model a strictly periodic event stream, while the arrival curves in Fig. 3b model a periodic event stream with jitter, and the arrival curves in Fig. 3c model a periodic event stream with bursts.

The arrival curves in Fig. 3d model an event stream with more complex timing behavior. This event stream may have short steep bursts, longer lasting less steep bursts, and the maximum long-term period does not equal the minimum long-term period. An event stream with such a complex timing behavior can not be represented accurately by any of the classical event arrival patterns. □

As defined above, the arrival curves $\overline{\alpha}^u$ and $\overline{\alpha}^l$ denote the number of events that arrive on an event stream in any given time interval. For performance analysis we are however not so much interested in the number of events that arrive, but rather in the resource demand that these arriving events produce on a HW/SW component. We therefore introduce *resource-based arrival curves* that are denoted as $\alpha(\Delta) = \left[\alpha^u(\Delta), \alpha^l(\Delta)\right]$. While event-based arrival curves represent the number of arriving events per unit of time interval, the resource-based arrival curves represent the generated resource demand per unit of time interval.

In the most basic scenario, every arriving event generates the same resource demand on a HW/SW component, i.e., the worst-case execution demand equals the best-case execution demand. Resource-based arrival curves can then be obtained directly by multiplying the event-based arrival curves with a constant that represents the resource demand of a single event. And analogously, event-based arrival curves are obtained by dividing resource-based arrival curves by the same constant.

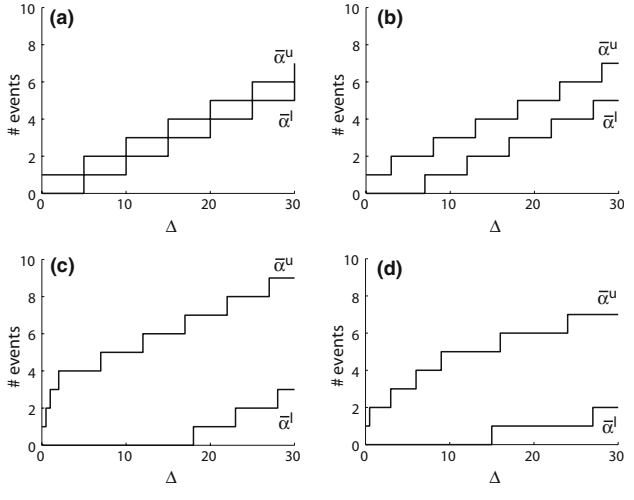

Fig. 3 Examples of arrival curves

In more complex systems, the events arriving on an event stream may be of one of several different event types, each having a different resource demand, or it may be known that not all events lead to the worst-case execution demand. In such systems, automata may be used to represent possible arrival patterns of the different event types, and the information captured in these automata may then be used to transform event- to resource-based arrival curves. A similar approach may also be used to model system state-dependent workload demands, as introduced for example by caches. For more details see [12], [18], and [19].

2.2 Service curves: a general resource model

Analogously to the cumulative function $R(t)$, the concrete availability of a computation or communication resource can be described by a cumulative function $C(t)$, that is defined as the number of available resources, e.g., processor cycles or bus capacity, in the time interval $[0, t)$.

Analogous to arrival curves that provide an abstract event stream model, a tuple $\beta(\Delta) = \left[\beta^u(\Delta), \beta^l(\Delta)\right]$ of upper and lower *service curves* then provides an abstract resource model. The upper service curve $\beta^u(\Delta)$ provides an upper bound on the available resources in *any* time interval of length Δ, and the lower service curve $\beta^l(\Delta)$ provides a lower bound on the available resources in a time interval Δ. And in other words again, in any time interval of length Δ there will always be at least $\beta^l(\Delta)$ and at most $\beta^u(\Delta)$ capacity available on a resource that is modeled by $\beta(\Delta)$.

In Real-Time Calculus, service curves are defined as follows:

Definition 2 (service curves) Let $C(t)$ denote the number of processing or communication cycles available from a resource over the time interval $[0, t)$. Then C, β^u, and β^l are related by the following inequality

$$\beta^l(t-s) \leq C(t) - C(s) \leq \beta^u(t-s), \quad \forall s < t \tag{4}$$

with $\beta^l(0) = \beta^u(0) = 0$. □

Note that the above definition of lower service curves corresponds to the definition of strict service curves in Network Calculus [10], while the definition of upper service curves as given above is not used in [10].

The service curves of a resource can be determined using data sheets, using analytically derived properties, or by measurement. For example, in the simplest case of an unloaded processor, whose capacity we measure in available processing cycles per time unit, both the upper and the lower resource curves are equal and are represented by straight lines $\beta^u(\Delta) = \beta^l(\Delta) = f \cdot \Delta$, where f equals the processor speed, i.e., the number of available processing cycles per time unit. With service curves, we may also model communication resources, where the service curves are bounded by the minimum and maximum number of transmittable bits in a given time interval.

Example 2 Figure 4 shows some examples of service curves that model the resource availability on processors or communication channels. The service curves in Fig. 4a model a resource with full availability, while the service curves in Figure 4(b) model a bounded delay resource. The service curves in Fig. 4c model the resource availability of one slot on a time division multiple access (TDMA) resource, and finally the service curves in Fig. 4d model a periodic resource as defined in [15]. □

2.3 From components to abstract components

In a real-time system, an incoming event stream is typically processed on a sequence of HW/SW components, that we will interpret as tasks on a task chain that are executed on possibly different hardware resources.

Figure 5 shows such a component. A trace of an event stream, described by $R(t)$, enters the component and is processed using a hardware resource whose availability is described by $C(t)$. After being processed, the events are emitted on the output of the component, resulting in an outgoing event stream trace, described by $R'(t)$, and the remaining resources that were not consumed to process the event trace $R(t)$ are made available to other components and are described by an outgoing resource availability $C'(t)$.

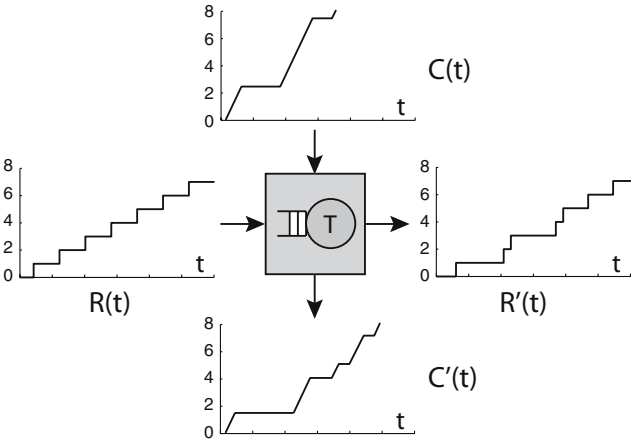

Fig. 5 A concrete component, processing an event stream on a resource

The relations between $R(t)$, $C(t)$, $R'(t)$, and $C'(t)$ depend on the processing semantics of the component. The outgoing event stream $R'(t)$ will typically not equal the incoming event stream $R(t)$, as it may, for example, exhibit more (or less) jitter. Analogously, $C'(t)$ will differ from $C(t)$.

For modular performance analysis with Real-Time Calculus, we model such a HW/SW component as an abstract component as shown in Fig. 6. Here, an abstract event stream $\alpha(\Delta)$ enters the abstract component and is processed using an abstract resource $\beta(\Delta)$. The output is then again an abstract event stream $\alpha'(\Delta)$, and the remaining resources are expressed again as an abstract resource $\beta'(\Delta)$.

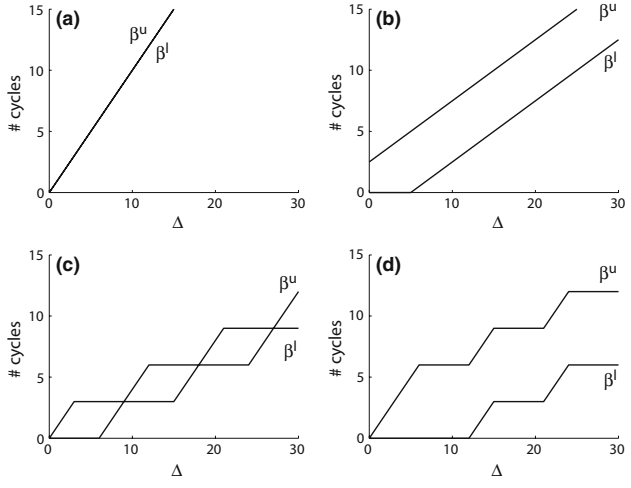

Fig. 4 Examples of service curves

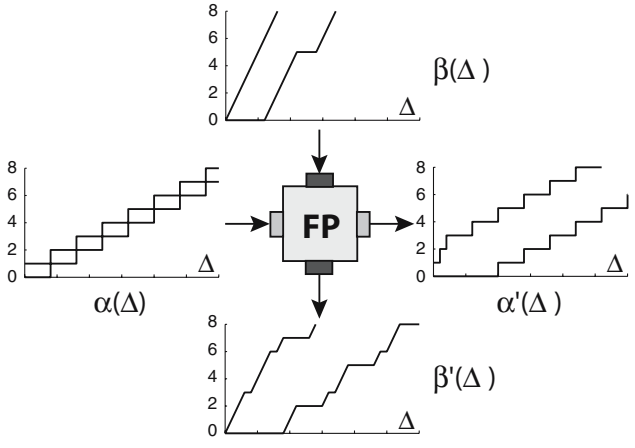

Fig. 6 An abstract component, processing an abstract event stream on an abstract resource

Internally, such an abstract component is specified by a set of functions that relate the incoming arrival and service curves to the outgoing arrival and service curves:

$$\alpha' = f_\alpha(\alpha, \beta) \tag{5}$$

$$\beta' = f_\beta(\alpha, \beta) \tag{6}$$

For a given abstract component, these relations f_α and f_β depend on the processing semantics of the modeled concrete component, and must be determined such that $\alpha'(\Delta)$ correctly models the event stream with event trace $R'(t)$ and that $\beta'(\Delta)$ correctly models the resource availability $C'(t)$.

As an example of an abstract component, consider a concrete component that is triggered by the events of an incoming event stream. A fully preemptable task is instantiated at every event arrival to process the incoming event, and active tasks are processed in a greedy fashion in FIFO order, while being restricted by the availability of resources. Such a component can be modeled as an abstract component with following internal relations[1] [3]:

$$\alpha'^{u}_{FP} = \min\left\{\left(\alpha^u \otimes \beta^u\right) \oslash \beta^l, \beta^u\right\} \tag{7}$$

$$\alpha'^{l}_{FP} = \min\left\{\left(\alpha^l \oslash \beta^u\right) \otimes \beta^l, \beta^l\right\} \tag{8}$$

$$\beta'^{u}_{FP} = \left(\beta^u - \alpha^l\right) \mathbin{\overline{\oslash}} 0 \tag{9}$$

$$\beta'^{l}_{FP} = \left(\beta^l - \alpha^u\right) \mathbin{\overline{\otimes}} 0 \tag{10}$$

Components with the above described processing semantics are very common in the area of real-time embedded systems, and we will refer to them as a *fixed priority* (FP) components.

To model a component with different processing semantics, one has to determine the appropriate internal relations f_α and f_β to obtain a corresponding abstract component.

2.4 Abstract system performance models

At this point, we know how to model event streams, computation and communication resources, as well as single HW/SW components (tasks). But in order to analyze performance criteria of a system, we need to build an abstract model of the complete system architecture. We will call such a model the *abstract performance model* of a system.

To obtain the abstract performance model of a system, we first need to abstractly model all event streams that trigger the system, all computation and communication resources that are available to the system, as well as all components (tasks) in the system, using the corresponding abstract representations, as described in the preceding sections. Then, by correctly interconnecting all arrival and service inputs and outputs of all these abstract models, we obtain the abstract performance model of the system. An example of an abstract performance model is depicted in Fig. 14.

The arrival inputs and outputs in the abstract performance model are interconnected to reflect the flow of data in the system horizontally, while the interconnections of service inputs and outputs model the resource sharing policies in the system vertically.

To elaborate on the service interconnections, suppose that several components of a system are allocated to the same resource. In the concrete system, these components share this resource according to a scheduling policy. In the abstract performance model, this scheduling policy on a resource can then be modeled by the way the abstract resources β are distributed among the different abstract components.

For example, consider preemptive fixed priority scheduling: an abstract component A with the highest priority may use all available resources of a CPU, whereas an abstract component B with the second highest priority only gets the resources that were not consumed by A. This resource sharing policy is modeled in the abstract performance model by using the service curves β'_A that exit the abstract FP component A as input to the abstract FP component B.

For some other scheduling policies, such as GPS (generalized processor sharing) or TDMA (time division multiple access), the available resources must be distributed differently, while for some scheduling policies, such as EDF (earliest deadline first) or non-preemptive scheduling, different abstract components, with tailored internal relations, must be used. Some examples of how to model different scheduling policies are depicted in Fig. 7.

2.5 Analysis

After interconnecting all abstract models of a system to the system's abstract performance model as described in the previous section, this abstract performance model captures all the information that builds the basis for performance analysis with Real-Time Calculus. Various performance criteria, such as end-to-end delay guarantees or buffer requirements can be computed analytically in this abstract performance model. The exact analysis methods may thereby slightly vary for different abstract components but remains deterministic at all times. Following, we present the performance analysis methods for FP components. The analysis methods

[1] See the Sect. 6 for a definition of $\otimes$, $\oslash$, $\overline{\otimes}$, and $\overline{\oslash}$.

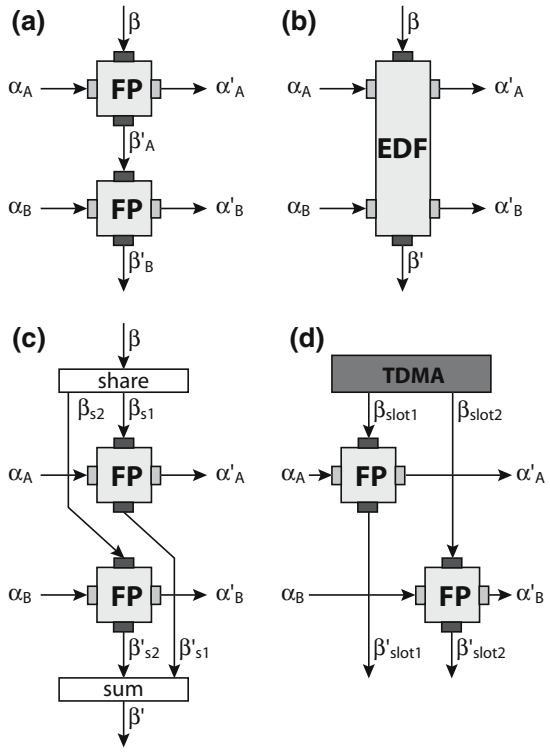

Fig. 7 Modeling of preemptive fixed priority scheduling (**a**), EDF scheduling (**b**), GPS scheduling (**c**) and TDMA scheduling (**d**) in MPA

for other abstract components are mostly very similar or even equal to these.

When an event stream with arrival curves α is processed by an FP component on a resource with service curve β, then the *maximum delay* $d_{\max}$ experienced by any event on the event stream is bounded by [3,10]:

$$d_{\max} \leq \sup_{\lambda \geq 0} \left\{ \inf \left\{ \tau \geq 0 : \alpha^u(\lambda) \leq \beta^l(\lambda + \tau) \right\} \right\} \stackrel{def}{=} Del(\alpha^u, \beta^l) \tag{11}$$

When an event stream is processed by a sequence of several components, we could simply add the different maximum delays of each individual component together to obtain an end-to-end delay guarantee. However, in this case we can exploit the phenomenon known as "Pay Bursts Only Once" [10], and the end-to-end delay guarantee can be tightened to [10]:

$$d_{\max} \leq Del\left(\alpha^u, \beta_1^l \otimes \beta_2^l \otimes \cdots \otimes \beta_n^l\right) \tag{12}$$

On the other hand, the maximum buffer space $b_{\max}$ that is required to buffer an event stream with arrival curve α in the input queue of an FP component on a resource with service curve β is bounded by [10]:

$$b_{\max} \leq \sup_{\lambda \geq 0} \left\{ \alpha^u(\lambda) - \beta^l(\lambda) \right\} \stackrel{def}{=} Buf\left(\alpha^u, \beta^l\right) \tag{13}$$

and when the buffers of several consecutive components use the same shared memory, the total required buffer space can even be tightened to:

$$b_{\max} \leq Buf\left(\alpha^u, \beta_1^l \otimes \beta_2^l \otimes \ldots \otimes \beta_n^l\right) \tag{14}$$

In Fig. 8, the relations between α, β, $d_{\max}$, and $b_{\max}$ are depicted graphically. From this figure, we see that $d_{\max}$ and $b_{\max}$ are bounded by the maximum horizontal and maximum vertical distance between the upper arrival curve and the lower service curve, respectively. This corresponds to the intuition that $d_{\max}$ and $b_{\max}$ occur when the maximum load arrives at the same time as the minimum resources are available.

Besides enabling the computation of the various end-to-end delay guarantees and buffer requirements in a system, the abstract performance model of a system may also provide other interesting insights to a system that may for example be obtained by analyzing the characteristics of the outgoing service curves. This analysis may for example expose the utilization of the various computation or communication resources in the system.

2.6 The relation between network and Real-Time Calculus

Real-Time Calculus is based on Network Calculus [10] and extends the basic concepts of it to the domain of real-time embedded systems, as we already mentioned at the start of this chapter. The foundations of Network Calculus itself are Max-Plus and Min-Plus algebra. A good introduction to these algebras can be found in [1] and [10].

Real-Time Calculus is a direct extension of Network Calculus, and therefore there are many fundamental results that exist in both the calculi. However, there are also differences between Real-Time Calculus and

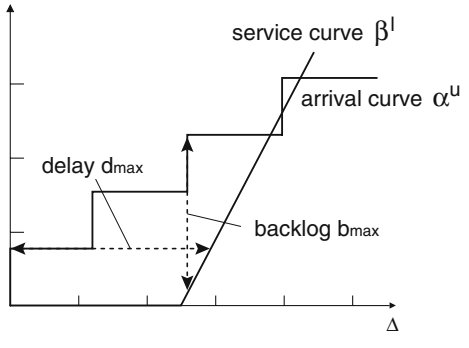

Fig. 8 Delay and backlog obtained from arrival and service curves

Network Calculus, that are sometimes only subtle but nevertheless important. In the following, we comment on the most important differences.

First of all, in Network Calculus the basic quantity that interconnects everything in an analyzed network system is the cumulative flow $R(t)$, although (upper) arrival curves are also of some importance. Most results in [10] are derived to work directly on cumulative flows, and also the definition of service curves in [10] is based directly on the relation between $R(t)$ and $R'(t)$.

In Real-Time Calculus, on the other hand, the basic quantities that connect everything are arrival curves α as well as service curves β. In Real-Time Calculus, the same regard is paid to the resource model as to the event stream model, and both models use consequently upper as well as lower curves.

In Network Calculus, lower arrival curves do not exist, and upper service curves play an insignificant role. In Real-Time Calculus, these curves, however, play a significant role and lead to tighter results for delay guarantees and buffer requirements in distributed systems with multiple input event streams. Without upper service curves that limit the maximum available service to a component, this maximum available service must be assumed to be infinite. This, however, would mean that any number of backlogged events could in the best-case be processed in zero time, possibly leading to a large burst on the output event stream that consequently would lead to a big worst-case resource demand burst on the succeeding component. With upper service curves, such bursts are flattened. Lower arrival curves, on the other hand, are similarly needed to tighten outgoing upper service curves of a component, and sometimes their existence is even essential, for example, to analyze real-time systems with play-out buffers, see [11]. As a consequence of the importance of upper service curves and lower arrival curves, the relations that define abstract component, as for example (7)–(10), are different from the ones known from Network Calculus.

Another big difference is the treatment of remaining resources. In Network Calculus, remaining resources are not explicitly considered, and consequently no results are presented in [10] to compute them. In Real-Time Calculus, on the other hand, remaining resources and therefore the computation of β' plays a crucial role for the ability of modular composability, and lays also the foundation to the modeling of various scheduling policies.

2.7 Deficiencies of MPA with Real-Time Calculus

The key enabling factor for the modularity in MPA and the easy analyzability of abstract performance models with Real-Time Calculus is the consequent representation of all time-varying quantities (event streams and resources) in a time interval domain. This abstraction from the time domain to the time interval domain does, however, not come for free.

Consider a system that processes two event streams described by $R_1(t)$ and $R_2(t)$, and assume that both event streams are periodic with periods $p_1 = p_2$. Depending on the application area, it may be known that the event streams are implicitly synchronized and that a fixed time shift exists between events arriving on $R_1(t)$ and $R_2(t)$. Due to this time shift, it may be that events of these two event streams may never arrive at the same time in the real system. If we model these event streams in the time interval domain, using arrival curves, then the knowledge of this time shift that was implicitly represented in $R_1(t)$ and $R_2(t)$ gets lost. The Real-Time Calculus analysis results would reflect the worst-case scenario, where the events of the two event streams would always arrive at the same time, even though this worst-case scenario may never occur in the real system, due to the implicit dependency of the two event streams.

The same deficiency on representing concrete time also limits the capability of Real-Time Calculus to analyze systems with synchronized resources. Research is, however, going on to, at least partly, eliminate these deficiencies [20].

2.8 Areas of application

The framework of MPA with Real-Time Calculus is tailored towards performance analysis of distributed real-time systems, where independent applications share a common execution platform to process event streams.

In such systems, the framework can be used not only to compute hard upper and lower bounds on maximum end-to-end delays and buffer requirements, but also other performance criteria such as individual resource utilizations may be analyzed.

The obtained analysis results are deterministic and provide hard upper and lower bounds for any analyzed quantity. This enables the framework to be used for the analysis of hard real-time systems. However, as a consequence it is obviously not possible to obtain average case results for any performance criteria in a system.

The framework can analyze distributed systems consisting of any number of interconnected computation and communication resources that are shared among different applications using preemptive fixed priority scheduling (FP), rate monotonic scheduling (RM), generalized processor sharing (GPS), or time division multiple access (TDMA). The framework can also analyze

resources that are shared with any hierarchical composition of these scheduling policies.

Research is currently going on to analyze systems with shapers, as well as resources that are shared using earliest deadline first scheduling (EDF) and non-preemptive scheduling policies.

So far, the framework was only used to analyze systems without cyclic dependencies. For systems with cyclic dependencies, it is, however, possible to perform a fixed-point analysis. While this area still requires further research, first promising results are already available [14].

In the framework, tasks are specified only by their execution demand. This sometimes limits the level of detail that can be modeled and analyzed, especially when it comes to including functional properties of a system. The framework may, however, be used to analyze systems with deterministic execution demand variability as introduced for example by deterministic caches, data dependencies or different occurring event types.

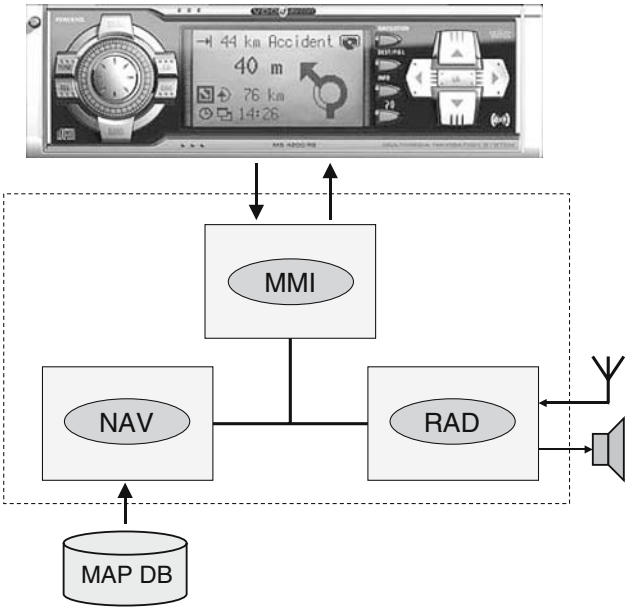

Fig. 9 High-level overview of a distributed radio navigation system

3 Case study: distributed in-car radio navigation system

The case study presented in this section is inspired by a system architecture definition study for a distributed in-car radio navigation system. Such a system typically executes a number of concurrent applications that share a common platform. Nevertheless, each application might have hard individual performance requirements that need to be met by the platform. During the system definition phase, several candidate platform architectures might be proposed by the engineers and the system architect needs to evaluate each one. Typical questions that need to be answered are: (1) does this platform meet the performance requirements of all applications (2) how robust is the platform with respect to changes in application or architecture parameters and (3) can I replace components in the architecture by cheaper (but less powerful) components to save cost but still meet the performance criteria of all applications? We present the applications and the architecture candidates in Sect. 3. We briefly show how these are modeled and how a MPA model is composed. In Sect. 4 we will show how typical design questions, as the ones mentioned before, can be analyzed using Real-Time Calculus.

An overview of the system is presented in Fig. 9, it is composed of three main clusters of functionality:

– The man–machine interface (MMI) which takes care of all interaction with the user, such as handling key inputs and graphical display output.
– The navigation functionality (NAV) which is responsible for destination entry, route planning, and turn-by-turn route guidance giving the driver both audible and visual advices. The navigation functionality relies on the availability of a map database, typically stored on a CD or DVD, and positioning information, e.g., speed and GPS. The latter is not shown here.
– The radio functionality (RAD) which is responsible for basic tuner and volume control as well as handling of traffic information services such as RDS/TMC (radio data system/traffic message channel). RDS/TMC is broadcast along with the audio signal of radio channels.

The key question that is investigated in this paper is how to distribute the functionality over the available resources, such that we meet our global timing requirements. To achieve this goal, the following steps were taken:

1. identify key usage scenarios and system functions
2. quantify event rates, message sizes, and execution times
3. identify resources and their communication structure
4. quantify resource and communication capacities
5. compose a MPA model, calculate, and evaluate

A general description of a new product is typically made during the initial phase of an industrial product creation process. For example, an *Operational Concept Description* from the IEEE 12207 system life cycle standard [8] may be produced. Such a document does not only list functional and non-functional requirements, boundary conditions and other restrictions for the design, it should also contain high-level use-cases. These use-cases are the starting point for the design of the system architecture. The use-cases and associated sequence diagrams are analyzed and annotated in such a way that they are useful for MPA analysis. This is step 1 and 2 of the recipe described above. Although there is no principle limit to the amount of scenarios that can be analyzed, it is not uncommon to first concentrate on those scenarios that are expected to have the highest impact on the set of requirements to be met. It is the system architect who makes this decision, often based on previous experience. The order of magnitude of the numbers shown in the sequence diagrams in this paper is realistic. During the design, the system architect tries to improve the accuracy of the numbers by using for example better estimation techniques on details of the design, such as worst-case execution time analysis (WCET) or by performing measurements on existing and comparable systems. In our case study, we have selected three distinctive scenarios:

1. "Change volume" – The user turns the rotary button and expects instantaneous audible feedback from the system. Furthermore, the visual feedback (the volume setting on the screen) should be timely and synchronized with the audible feedback. This seemingly trivial use-case is actually quite complex because many components are affected. Changing volume might involve commanding a digital signal processor (DSP) and an amplifier in such a way that the quality of the audio signal is maintained while changing the volume. This scenario is shown in detail in Fig. 10. Note that three operations are identified, *HandleKeyPress*, *AdjustVolume*, and *UpdateScreen*. Execution times, event rates, and message sizes are estimated and annotated in the sequence diagram together with the principle timing requirements applicable to this scenario.
2. "Address look-up" – Destination entry is supported by a smart "typewriter" style interface. By turning a knob the user can move from letter to letter; by pressing it the user will select the currently highlighted letter. The map database is searched for each letter that is selected and only those letters in the on-screen alphabet are enabled that are potential next letters in the list. This scenario is shown in detail in Fig. 11. Note that the *DatabaseLookup* operation is expensive compared to the other operations and that the size of the output value of the operation is 16 times larger than the input message.
3. "TMC message handling" – Digital traffic information is very important for in-car radio navigation systems. It enables features such as automatic re-planning of the planned route in case a traffic jam occurs ahead. It is also increasingly important to enhance road safety by warning the driver, for example when a ghost driver is spotted on the planned route. RDS TMC is such a digital traffic information service. TMC messages are broadcast by radio stations together with stereo audio sound. RDS TMC messages are encoded: only problem location identifiers and message types are transmitted. The map database is accessed to translate these identifiers and to construct human readable text. The TMC message handling scenario is shown in Fig. 12.

The scenarios sketched above have an interesting property: they can occur in parallel. RDS TMC messages must be processed while the user changes the volume or enters a destination. However, "Change Volume" and "Address Look-up" cannot occur at the same time because they share a common resource; the rotary button is used for both. The architecture shown in Fig. 9 suggests to assign the three clusters of functionality each to its own processing unit. The computation resources are interconnected by a single communication bus. Does this architecture meet our requirements and is it the best architecture for our applications?

Figure 13 shows that there are many more potential architectures that might be applicable. Note that the capacity of the resource units and communication infrastructure is quantified, completing steps 3 and 4 of our approach. Again, the order of magnitude of the numbers shown in the diagram is correct – they are taken from the data sheets of several commercially available automotive CPUs. Observe that architecture (b) can only be evaluated if we introduce an additional operation on the MMI resource that transfers the data from one communication link to another, in the case that NAV wants to communicate to RAD or vice versa.

Sufficient information is now available to construct the MPA models for each architecture. The model for architecture (a) is shown in Fig. 14. Note that the resources occur as column headings in the model. Observe that all outgoing horizontal arrows from the performance components NAV, RAD, and MMI, representing their respective output message flows, connect to an input of a BUS performance component, which

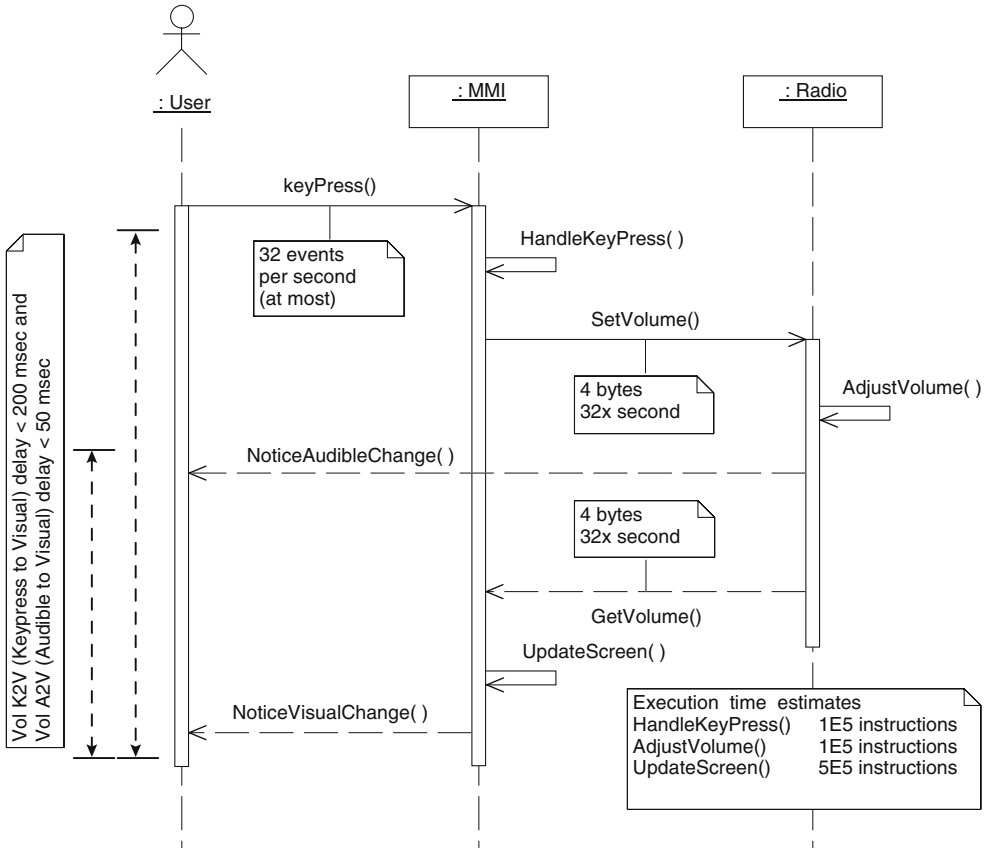

Fig. 10 Annotated sequence diagram for "change volume"

encodes the notion of the *shared* communication medium. In the case of architecture (b), two BUS resources (columns) would exist in the model instead of one.

The scenarios, that were defined by the sequence diagrams, are depicted as the rows of the model. Each row starts with a load scenario symbol, which is connected to the input of a performance component. The flow of the sequence diagrams can be followed, in horizontal direction, in the MPA model. Take for example the "change volume" scenario from Fig. 10. Events arrive at the MMI where *HandleKeyPress* is executed and the result is forwarded, via the communication bus, to RAD. *AdjustVolume* is executed and the result is sent back to MMI via the same communication bus. Finally, *UpdateScreen* is executed and the scenario is completed. The load scenario data, α is extracted from the annotations in the sequence diagram. The resource model data, β, is extracted from the informal deployment diagrams shown in Fig. 13.

As described in Sect. 2.4, the order of the rows in the MPA model determines the priority. In this case, the "change volume" scenario is assigned a higher priority than "Handle TMC". The system architect decides the initial priority setting again based on experience. This does not hinder the evaluation in any way, since the priorities are easily changed by rearranging the vertical order of the scenarios. A MPA model must be constructed for each proposed architecture. This is normally a simple tasks because it is merely reconnecting event flows in the horizontal direction.

4 System analysis

In this section, we will look at some typical design problems that occur during the early phases of a system design cycle, and we will address them using the approach to MPA presented in Sect. 2.

For a correct interpretation of the results in this section, we need to remember that in order to be applicable for the analysis of hard real-time systems, the modular performance analysis presented in Sect. 2 is designed to compute hard upper and lower bounds for all system characteristics. While these upper and lower bounds are always hard, they are in general not tight (exact). So the analysis performed is conservative and

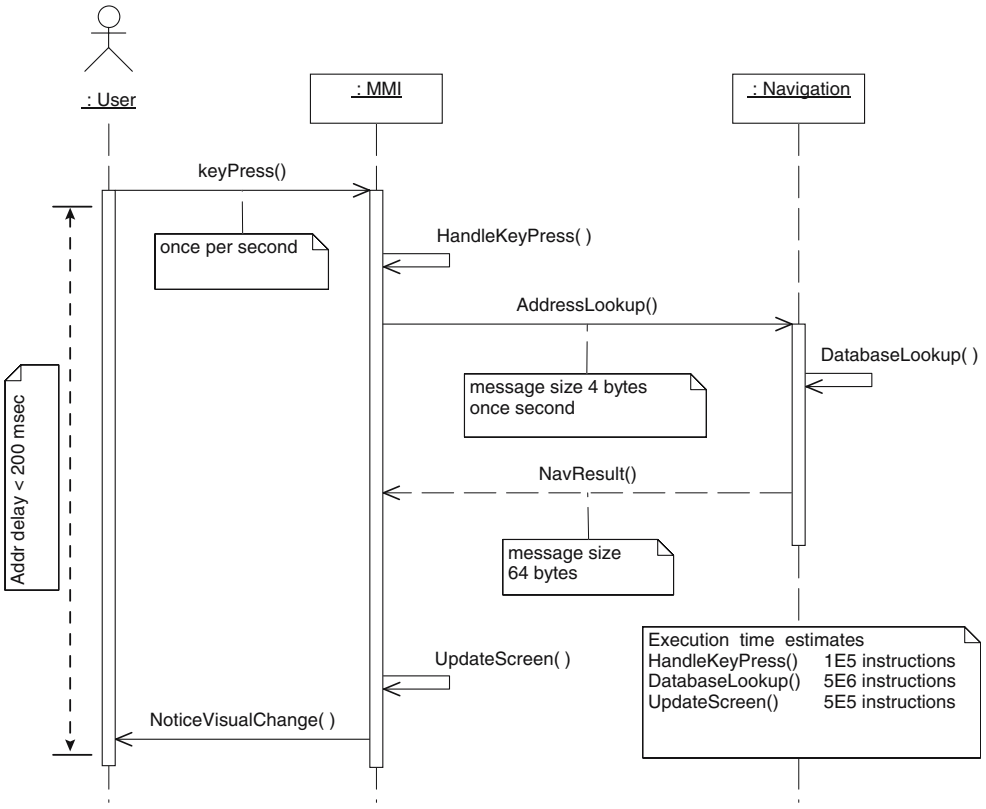

Fig. 11 Annotated Sequence Diagram for "Address Look-up"

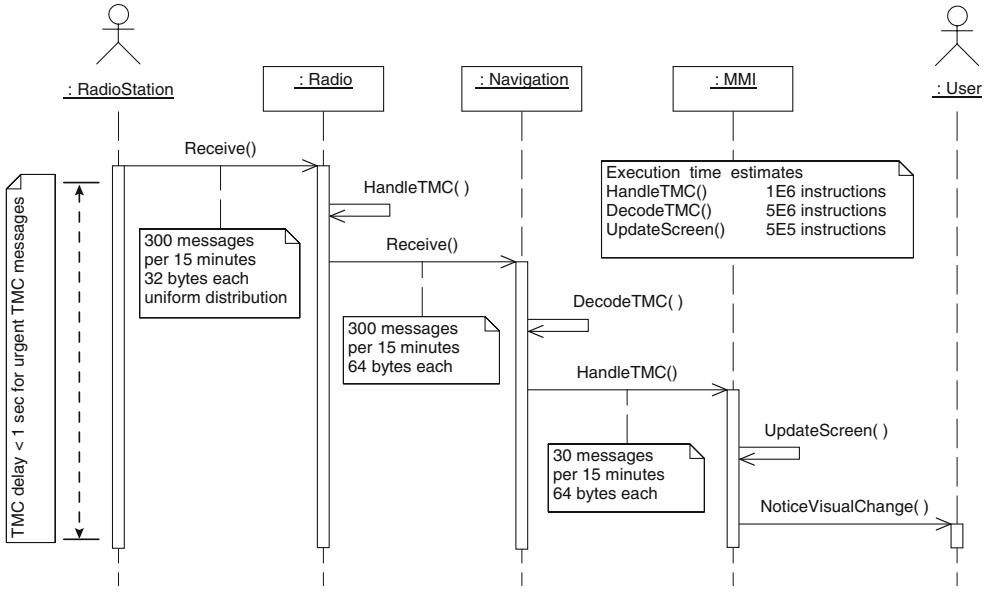

Fig. 12 Annotated sequence diagram for "TMC message handling"

the computed maximum delays in this section are therefore hard upper bounds to the real maximum delays in the real system.

Due to this conservative approach, it may be that we reject a system architecture that would fulfill all system requirements in reality, but for which our analysis cannot guarantee the fulfillment of all system requirements. The other way around, we can guarantee that any system architecture accepted by our analysis fulfills all system requirements in reality.

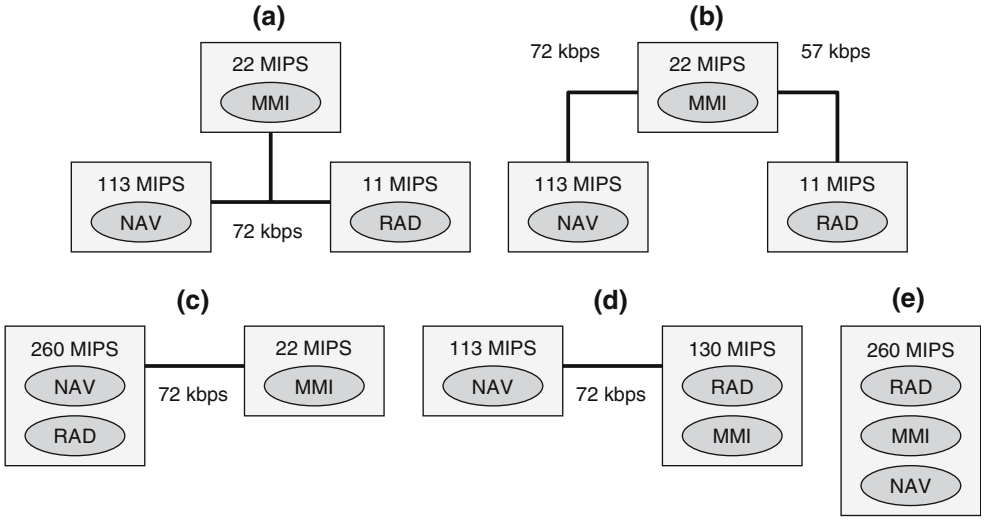

Fig. 13 Alternative system architectures to explore

Without any attempts to optimization, analyzing one system architecture in the design space took around 1 s on a Pentium Mobile 1.6 GHz using Matlab 7. Therefore, computing the four mesh plots in Fig. 19 took for example around 5 min.

To compute the end-to-end delay of an event stream in a given system architecture, we first construct the performance model of the given system architecture. By applying the transformations given in formulas 7–10 at every performance component of the performance model, we eventually end up with the output event streams and the remaining system resources. We then use formula 11 to compute the upper bound of the maximum delay of an event stream at every performance component it passes in the performance model. Finally, we sum up all these delays, using formula 12 to obtain a hard upper bound on the maximum end-to-end delay of the event stream in the given system architecture.

4.1 Design problems and analysis results

We will now present three typical design problems and show how they are analyzed.

Problem 1 In Fig. 13, five different system architectures are proposed for the in-car radio navigation system. How do these system architectures compare with respect to the different end-to-end delay requirements of the three use-cases? Consider that the "change volume" and the "address look-up" scenarios may not occur at the same time, because the same rotary button is used for these functions. However, "TMC message handling" does occur in parallel with either scenario.

We build the performance model for the "change volume" & "TMC message handling" situation (depicted in Fig. 14), as well as the performance model for the "address lookup" and "TMC message handling" situation. For both models, we compute the upper bounds to the end-to-end delay of every event stream as described in the last section, and we then take the end-to-end delays obtained from the two analysis runs (for the TMC delay, we take the bigger value of the two runs). From the results presented in Fig. 15, we see that all proposed system architectures fulfill the requirements (as mentioned in Figs. 10, 11, 12) on the different maximum end-to-end delays. Furthermore, the results suggest that architectures (d) and (e) process the input data to the system particularly fast. This may be explained partly by the reduced communication overhead in these architectures, but most probably, these architectures are also over-dimensioned.

Problem 2 Suppose that the in-car radio navigation system is implemented using architecture (a). How robust is this architecture? Where is the bottleneck of this architecture?

To investigate the robustness of architecture (a), we first compute its sensitivity towards changes in the input data rates. These sensitivity results are shown in Fig. 16. The height of the columns in this figure depict the increase of end-to-end delays relative to the respective specified maximum end-to-end delays, in dependence to increasing input data rates. For example, the tallest column in Fig. 16 shows us that if we increase the data rate of the "change volume" scenario slightly (i.e., by 3%, to 33.3 events/s), the end-to-end delay of the TMC message handling increases by 1.14% of its specified

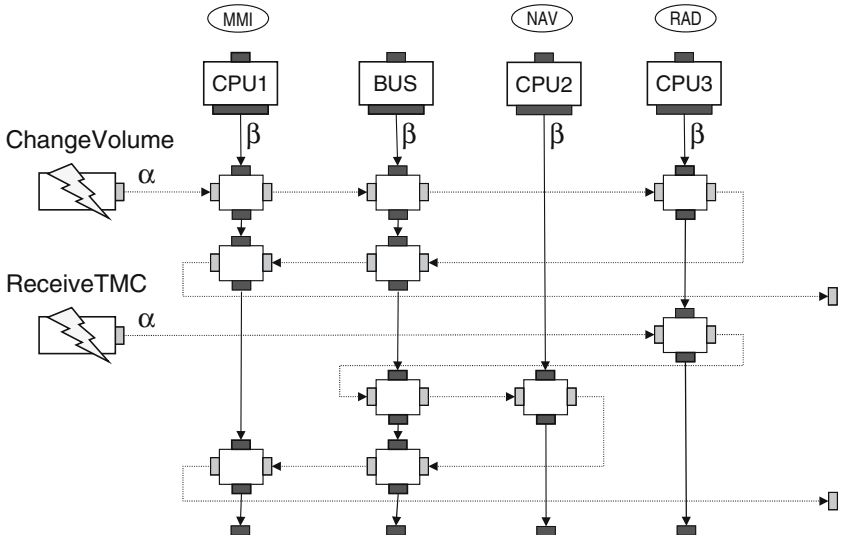

Fig. 14 MPA model for system architecture (a) of Fig. 13

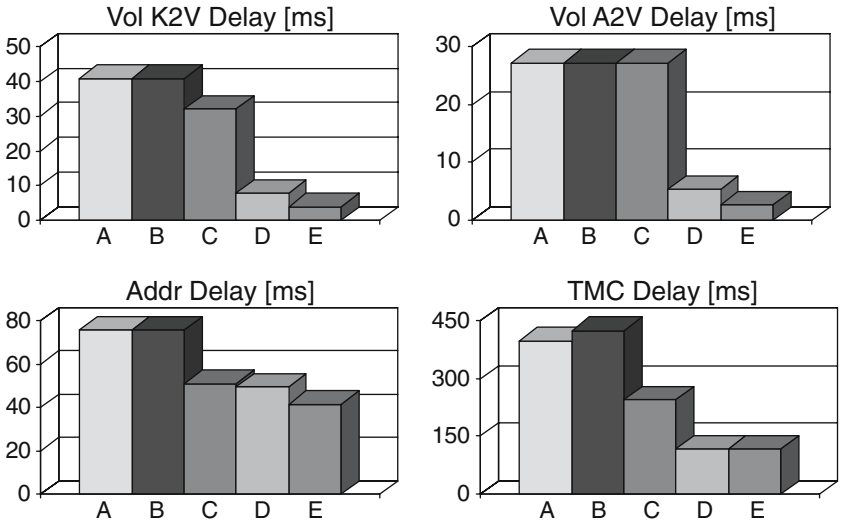

Fig. 15 Maximum end-to-end delays for each system architecture

maximum end-to-end delay (i.e., 1.14% of 1,000 ms or 11.4 ms).

From the results shown in Fig. 16, we see that architecture (a) is very sensitive towards increasing the input data rate of the "change volume" scenario, while increasing the input data rate of the "address look-up" and the "TMC message handling" scenarios do not really affect the response times. And in fact, further analysis reveals that in order to still guarantee all system requirements, we must not increase the input data rate of the "change volume" scenario by more than 7%, while we could increase the input data rate of the other two scenarios by a factor of more than 20.

After investigating the system sensitivity towards changes in the input data rates, we investigate the system sensitivity towards changes in the resource capacities. These sensitivity results are shown in Fig. 17. The height of the columns in this figure depicts the increase of end-to-end delays relative to the respective specified maximum end-to-end delays, in dependence to decreasing resource capacities. For example, from the tallest column in Fig. 17 we know that if we decrease capacity of the MMI processor by 1% (i.e., to 21.78 MIPS), the end-to-end delay of the TMC message handling increases by 3.22% of its specified maximum end-to-end delay (i.e., 3.22% of 1,000 or 32.2 ms).

From the results shown in Fig. 17, we see that architecture (a) is most sensitive towards the capacity of the MMI processor. This suggests that the MMI processor is a potential bottleneck of architecture (a). To investigate this further, we compute the end-to-end delay of the TMC message handling for different MMI processor capacities. The results of these computations are shown in Fig. 18.

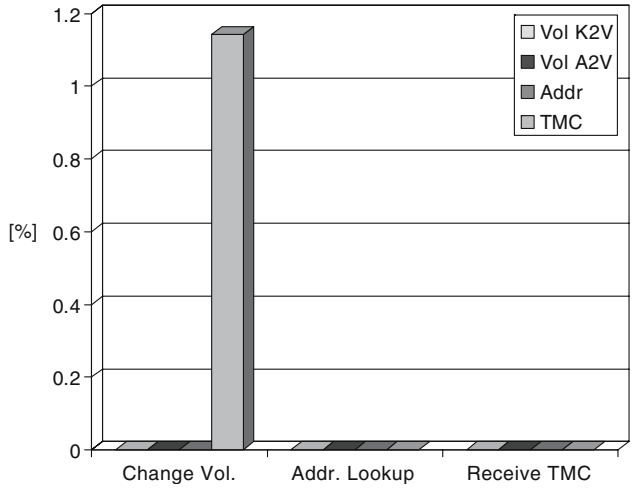

Fig. 16 Sensitivity towards changes in the input data rates

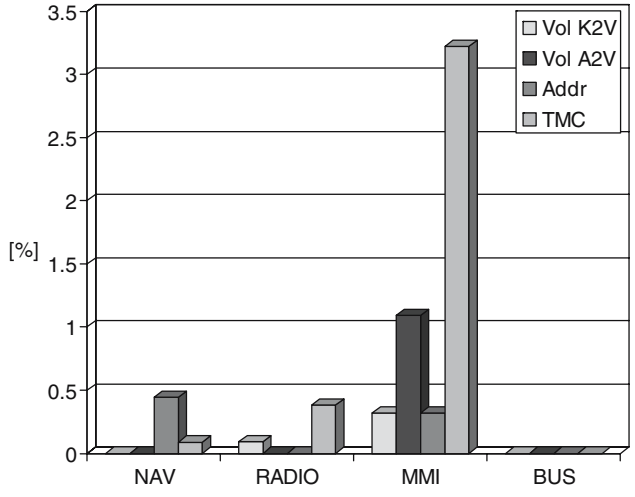

Fig. 17 Sensitivity towards changes in the resource capacities

From Fig. 18, we see that indeed at its given operation point, the end-to-end delay of the TMC message handling in architecture (a) is very sensitive towards changes of the MMI processor capacity. And the analysis reveals that with a decrease of the MMI processor capacity to 89% of its initial capacity, we cannot guarantee finite response times anymore.

To sum up, the above analysis results suggest that increasing the capacity of the MMI processor would make architecture (a) more robust. To support this statement, we individually increase the capacity of each resource by 20%, and we then analyze how much we can increase the input data rate of the "change volume" scenario while still fulfilling the requirements. Remember, with the initial resource capacities, we can increase the data rate of the "change volume" scenario by 7% and the data rate of the other two scenarios by a factor of more than 20 while still guaranteeing all requirements. From this analysis, we learn that increasing the resource capacities of the RAD processor, the NAV processor, and the BUS does not allow to increase the input date rate of the "change volume" scenario more than with the initial capacities, while increasing the MMI processor capacity allows us to increase the data rate of the "change volume" scenario by 60%.

Problem 3 Suppose system architecture (d) is chosen for further investigation. The results of Problem 1 indicate that architecture (d) is probably over-dimensioned. So how should the two processors in this system architecture be dimensioned, to obtain an economic system that still fulfills the end-to-end delay requirements of all scenarios?

We compute the upper bound to the end-to-end delay of every event stream in architecture (d) for different processor capacities. The results are shown in Fig. 19.

In the plots in Fig. 19, the NAV processor capacity is varied in steps of 5% from 100% down to 10% of its initial capacity. At the same time, the MMI/RAD processor capacity is varied in steps of 5% from 100% down to 20% of its initial capacity.

As we see from the plots, the delays of the "change volume" scenario are not much affected by changes of the NAV processor capacity and the delay of the "address look-up" scenario is not much affected by changes of the MMI/RAD processor capacity. On the other hand, the delay of the "TMC message handling scenario" is affected by the changes of both processor capacities. From the results, we learn that we could decrease both the NAV processor capacity as well as the MMI/RAD processor capacity down to 25% of their initial capacity (i.e., 29 and 33 MIPS, respectively), while still guaranteeing the fulfillment of all system requirements.

5 Conclusions and future work

Real-Time Calculus was originally intended for stream-based applications. We have shown in this case study that it is also well-suited for modeling control-oriented and distributed software-intensive systems. Creating MPA models is a relatively simple task that requires little effort. The models presented in this paper were composed manually and analyzed within the same working day. Quantifying the event rates and resource capacities actually took up more time than building the model because this information is seldom readily available.

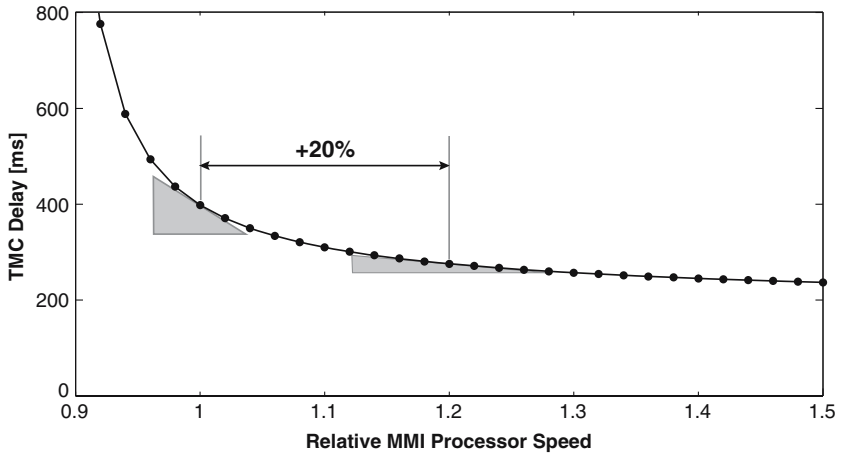

Fig. 18 TMC delay versus MMI processor speed

The method can play an important role in the typical dialog of a design team because of the short cycle time. Recalculation of a model is fast. If input data or analysis results are doubted, which is always the case in the early design stages, it is easy to change the data and investigate the consequence of that change. Especially non-technicians find this way of working satisfactory because they normally are reluctant to accept results coming from complex modeling and analysis exercises that they cannot oversee and understand completely. The attention is shifted away from the method towards the problem, hence it influences the design process positively.

A disadvantage of the Real-Time Calculus technique is that it can be too pessimistic. Since the calculus is based on both best and worst cases of event streams and available resources, it might be that a design is oversized to meet a worst case requirement, whereas in practice this situation does occur rarely. In soft real-time systems this

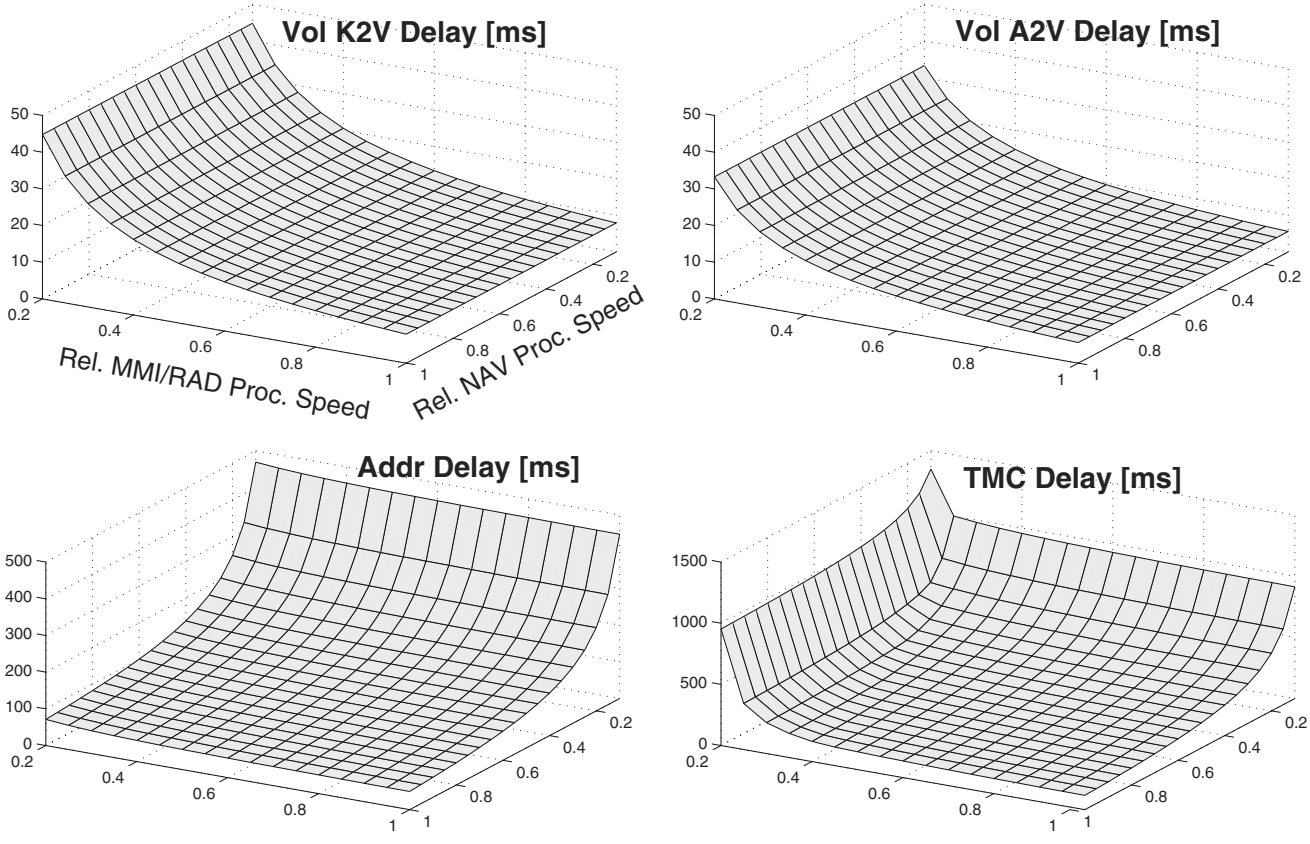

Fig. 19 Delays versus processor speed in architecture (d)

might entail a design which is too expensive. The current approach to counter this phenomenon is to increase the level of detail in the model, while still preserving the simplicity of the approach. The future research goal is that a few critical parts of a system may be modeled in detail, while other less critical parts may be modeled on a higher level of abstraction. With this approach, the model of system parts that seem to be critical may even be refined during the analysis process.

Modular performance analysis is a composable technique. Abstract performance components can actually consist of other MPA models. This approach has not been explored in this paper, neither have we investigated its impact on the analysis speed.

Future work. A Java implementation of the Real-Time Calculus, using *Matlab* as the user interface, is currently under development. These tools are inspired by a prototype implementation that was made earlier using *Mathematica*. Our aim is to compare MPA to other performance analysis techniques based on the case study presented in this paper. Comparison would include (but is not restricted to) classical scheduling analysis techniques, timed automata, Markovian, and traditional simulation. Furthermore, we plan to evaluate larger case studies, in particular, to investigate the scalability of MPA. Integration with UML tools, in particular, through the profile for schedulability, performance, and time [13] is needed to make MPA tool support acceptable to industry. The complete MPA model of the case study presented here can be found at http://www.mpa.ethz.ch.

Acknowledgments The authors wish to thank Erik Gaal, Evert van de Waal, Jozef Hooman, Jan Broenink, Lou Somers, Frits Vaandrager and the anonymous reviewers for providing feedback and comments to the paper. Furthermore, we would like to thank Siemens VDO Automotive, in particular Roland van Venrooy, for their support. This project is partly supported by the Netherlands Ministry of Economic Affairs under the Senter TS program. Part of the presented work is funded by the Swiss National Science Foundation (SNF) under the Analytic Performance Estimation of Embedded Computer Systems project 200021-103580/1, and by ARTIST2.

Appendix: Min-Plus and Max-Plus Calculi

Both min-plus and max-plus calculi define a special algebra (the min-plus dioid and max-plus dioid, respectively). Traditionally, we are used to work with the algebraic structure $(\mathbb{R}, +, \times)$, i.e., with the set of reals endowed with the operations of addition and multiplication that possess a number of properties such as associativity, commutativity, distributivity, etc.

In difference to this, min-plus calculus works with an algebraic structure $(\mathbb{R} \cup \infty, \vee, +)$. Here, the operation of addition becomes the computation of the infimum (or the minimum), and the operation of multiplication becomes the addition. Most axioms known from conventional algebra still apply to this algebraic structure. And in max-plus calculus, the infimum and minimum are simply replaced by supremum and maximum.

In Real-Time Calculus, we often need to compute convolutions and deconvolutions defined in min-plus and max-plus calculi. These operations are defined as follows [10]:

The min-plus convolution $\otimes$ and the min-plus deconvolution $\oslash$ of two functions f and g are defined as:

$$(f \otimes g)(\Delta) = \inf_{0 \le \lambda \le \Delta} \{f(\Delta - \lambda) + g(\lambda)\} \tag{15}$$

$$(f \oslash g)(\Delta) = \sup_{\lambda \ge 0} \{f(\Delta + \lambda) - g(\lambda)\} \tag{16}$$

The max-plus convolution $\overline{\otimes}$ and the max-plus deconvolution $\overline{\oslash}$ of two functions f and g are defined as:

$$\left(f \,\overline{\otimes}\, g\right)(\Delta) = \sup_{0 \le \lambda \le \Delta} \{f(\Delta - \lambda) + g(\lambda)\} \tag{17}$$

$$\left(f \,\overline{\oslash}\, g\right)(\Delta) = \inf_{\lambda \ge 0} \{f(\Delta + \lambda) - g(\lambda)\} \tag{18}$$

For more information on min-plus and max-plus calculi see [10] and [1].

References

1. Bacelli, F., Cohen, G., Olsder, G.J., Quadrat, J.P.: Synchronization and Linearity: An Algebra for Discrete Event Systems. Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons Ltd, New York (1992)
2. Balsamo, S., Di Marco, A., Inverardi, P.: Model-based performance prediction in software development: A survey. IEEE Trans Softw Eng **30**(5), 295–310 (2004)
3. Chakraborty, S., Künzli, S., Thiele, L.: A general framework for analysing system properties in platform-based embedded system designs. In: Proceedings of 6th Design, Automation and Test in Europe (DATE), Munich, Germany, pp. 190–195 (2003)
4. Cortellessa, V., Mirandola, R.: Deriving a queueing network based performance model from UML diagrams. In: Proceedings of 2nd International Workshop on Software and Performance (WOSP), Ottawa, Ontario, Canada, pp. 58–70 (2000)
5. Cruz, R.L.: A calculus for network delay. IEEE Trans Information Theory **37**(1), 114–141 (1991)
6. Gries, M.: Methods for evaluating and covering the design space during early design development. Tech. Rep. UCB/ERL M03/32, Electronics Research Lab, University of California at Berkeley (2003)
7. Grotker, T., Liao, S., Martin, G., Swan, S.: System Design with SystemC. Kluwer, Dordrecht (2002)
8. IEEE/EIA: ISO/IEC 12207:1995 Standard for Information Technology – Software life cycle processes. The Institute of Electrical and Electronics Engineers, Inc. (1998)

9. Kienhuis, B., Deprettere, E., Vissers, K., van der Wolf, P.: An approach for quantitative analysis of application-specific dataflow architectures. In: ASAP '97: Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures and Processors, IEEE Computer Society, Washington, DC, USA, p. 338 (1997)
10. Le Boudec, J.Y., Thiran, P.: Network Calculus - A Theory of Deterministic Queuing Systems for the Internet. No. 2050 in Lecture Notes in Computer Science (LNCS). Springer, Berlin Heidelberg New York (2001)
11. Maxiaguine, A., Chakraborty, S., Künzli, S., Thiele, L.: Evaluating schedulers for multimedia processing on buffer-constrained soc platforms. IEEE Design Test **21**(5), 368–377 (2004)
12. Maxiaguine, A., Künzli, S., Thiele, L.: Workload characterization model for tasks with variable execution demand. In: Proceedings of 7th Design, Automation and Test in Europe (DATE) (2004)
13. Object Management Group: UML Profile for Schedulability, Performance and Time Specification (2004). URL http://www.uml.org/. Version 1.1, ptc/04-02-01
14. Schioler, H., Larsen, K.G., Jessen, J., Dalsgaard, J.: Cync - a method for real time analysis of systems with cyclic data flows. In: Proceedings of the 13th International Conference on Real-Time Systems, Automation and Test in Europe (RTS). Paris, France (2005)
15. Shin, I., Lee, I.: Periodic resource model for compositional real-time guarantees. In: Proceedings of the Real-Time Systems Symposium (RTSS), IEEE Press, pp. 2–13 (2003)
16. Smith, C.U., Williams, L.G.: Computer Performance Evaluation: Modeling Techniques and Tools, chap. Performance Engineering Evaluation of Object-Oriented Systems with SPE*ED™. No. 1245 in Lecture Notes in Computer Science (LNCS). Springer, Berlin Heidelberg New York (1997)
17. Thiele, L., Chakraborty, S., Naedele, M.: Real-Time Calculus for scheduling hard real-time systems. In: Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS), vol. 4, pp. 101–104 (2000)
18. Wandeler, E., Maxiaguine, A., Thiele, L.: Quantitative characterization of event streams in analysis of hard real-time applications. In: 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) (2004)
19. Wandeler, E., Thiele, L.: Abstracting functionality for modular performance analysis of hard real-time systems. In: Asia South Pacific Design Automation Conference (ASP-DAC) (2005)
20. Wandeler, E., Thiele, L.: Characterizing workload correlations in multi processor hard real-time systems. In: 11th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), IEEE, San Francisco, USA, pp. 46–55 (2005)