System Architecture Evaluation Using Modular Performance Analysis - A Case Study ⋆

Ernesto Wandeler & Lothar Thiele, ETH Zurich,
Marcel Verhoef ⋆⋆, Chess Information Technology, and
Paul Lieverse, Siemens VDO Automotive

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 system life cycle. Simulation based techniques are often not sufficiently productive. This paper presents an alternative, analytical, approach based on Real-Time Calculus [1] developed at ETH Zurich. Modular Performance Analysis (MPA) 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.

1 Introduction

Todays' embedded systems industry is mainly characterised 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 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 realised that these investments are insufficient to become and stay competitive, because the gains achieved still fall well short of 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 At the time the product is ready, it may be outdated alltogether or it may be produced at lower cost using some other technology that has just become available.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 changed market conditions) is essential. This evaluation process shall be light-weight and reliable such that new products will meet customer requirements, can be produced faster and at minimum cost.

⋆ This case study has been carried out as part of the BODERC project under the responsibility of the Embedded Systems Institute, `www.embeddedsystems.nl`. This project is partly supported by the Netherlands Ministry of Economic Affairs under the Senter TS program.

⋆⋆ and Radboud University Nijmegen, NIII - Informatics for Technical Applications

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. *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.

SystemC ([2], [3] and [4]) is an example of 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 analyse 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, such as for example Matlab/Simulink [5]. 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 derivates. 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 analyse 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 more than once. Application of simulation based techniques is often postponed 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. 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 characterise 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 [1] 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, Modular Performance Analysis with Real-Time Calculus is presented. This technique is then applied to a case study of a distributed in-car radio navigation system, in which we annotate UML Message Sequence Charts to feed the analysis method with data. Several architectural derivates 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. [7] 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 [8], which is also mentioned in the overview of Gries [6]. Whereas 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 [9] which is in turn based on max-plus algebra [10]. Note that max-plus algebra has been applied to a wide variety of problems, including scheduling and performance analysis of dynamic discrete event systems. Implementations of the basic mathematical primitives are available for well-known mathematics packages such as *Scilab*, *Matlab* and *Maple*, see [11]. The method proposed in this paper uses a notation that is much closer to current engineering practice, while maintaining the sound mathematical basis provided by the Real-Time Calculus.

Some of the techniques used in this paper can also be found in other software oriented performance analysis methodologies mentioned in [7]. For example, Software Performance Evaluation (SPE) based methodologies, such as [12], also annotate Message Sequence Charts with information on resource utilization. The analysis is however done differently: SPE uses queueing networks as the underlying analysis methodology, whereas we use Real-Time Calculus. Cortelessa et al. [13] describes a methodology in which the queueing network based performance model is automatically derived from UML use case diagrams, sequence diagrams, and deployment diagrams. In our paper, the models are still constructed by hand but this could certainly be automated.

Spade [14] and Artemis [15] are examples of simulation based performance analysis methodologies. They address both hardware and software, where hardware is modelled separately from the required functionality of the system. However, the models used in Spade and Artemis are less abstract than the models we use in our case study because an executable model is required for simulation. Also, the cost of evaluation is higher due to the simulation approach, in particular for Spade. The latter has been addressed by Živković [16] in Artemis by using more abstract, yet executable, representations of the application.

Contribution of this paper. We show in this paper 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. The technique 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 Modular Performance Analysis and Real-Time Calculus

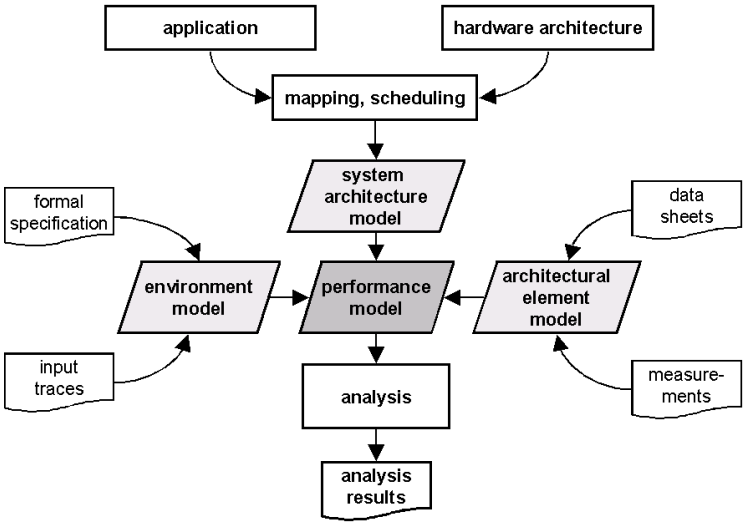


Fig. 1. Elements of Modular Performance Analysis and methods to obtain them.

During the early phases of a system life cycle, we are interested in exploring the limitations of our system design. For example we want to know about the fulfillment of timing requirements, the utilization of resources, and memory consumption. In this paper we present an approach called Modular Performance Analysis (MPA). The central idea is to construct a *performance model* of a system that captures the properties of the different system elements, such that the model can be used for analysis using Real-Time Calculus.

Real-Time Calculus [1] is based on the theoretical framework called Network Calculus [9] that provides a means to deterministically reason about timing properties of data flows in queueing networks. Real-Time Calculus extends the basic concepts of Network Calculus to the domain of real-time embedded systems. The analysis will show whether or not the system design meets the requirements and in particular whether or not there are potential bottlenecks in the design. Often a simple parameter sweep will be sufficient to investigate the sensitivity of a design.

Figure 1 presents an overview of the presented approach. Here, a performance model is derived from several other models; the *environment* model, the *architectural element* model and the *system architecture* model.

The environment model describes how the 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 Message Sequence Charts can be used for this purpose.

The architectural element model provides information about the properties of the computing and communication resources that are available within the system, for example, processor speed and communication bus bandwidth: information that is typically found in data sheets, benchmarks or that can be obtained from measurements on existing systems.

The system architecture model is based on the applications and the hardware architecture. It captures the mapping of application tasks to architectural elements and specifies the scheduling schemes used on the architectural elements, as well as the priorities of different tasks that are mapped on the same architectural elements.

In this paper we will use UML Message Sequence Charts to describe the different application scenarios, their task structure as well as their mapping to architectural elements. To demonstrate Modular Performance Analysis, we will use a few ad-hoc hardware architectures together with fixed mappings, scheduling schemes and task priorities. We did not attempt to explore the design space automatically.

In our approach, performance models are described using three basic primitives: *arrival curves*, *service curves* and *performance components*. We will first provide some definitions and then show how performance models are constructed.

2.1 Abstract Element Models

Event Stream Model Events can be modeled using so-called *arrival curves*, which were first introduced in [17]. An event stream model is a pair of functions $\bar{\alpha}^l : \mathbb{R}_{\geq 0} \rightarrow \mathbb{N}$ and $\bar{\alpha}^u : \mathbb{R}_{\geq 0} \rightarrow \mathbb{N}$. For any value $t \in \mathbb{R}_{\geq 0}$, $\bar{\alpha}^l$ (which is called the lower arrival curve) returns the *minimum* number of events that can be observed in *any* time interval of length t. Similarly, $\bar{\alpha}^u$ (which is called the upper arrival curve) returns the *maximum* number of events. The arrival curves are related as follows:

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

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

where $\bar{\alpha}^l(0) = \bar{\alpha}^u(0) = 0$.

The timing information of standard event models, such as *periodic*, *periodic with jitter*, *periodic with bursts*, *sporadic*, or of any other event stream with a known timing behavior can be represented by an appropriate choice of $\bar{\alpha}^u$ and $\bar{\alpha}^l$ [8]. Moreover, it is

also possible to determine the tightest values of $\bar{\alpha}^u$ and $\bar{\alpha}^l$ corresponding to any given finite length event trace, obtained for example from observation or simulation.

Example 1. Figure 2 shows on the left side the abstract model of an event stream. From the arrival curves, we see that the stream can be modeled as having a period $p = 1\ ms$ and a jitter $j = 400\ \mu s$.

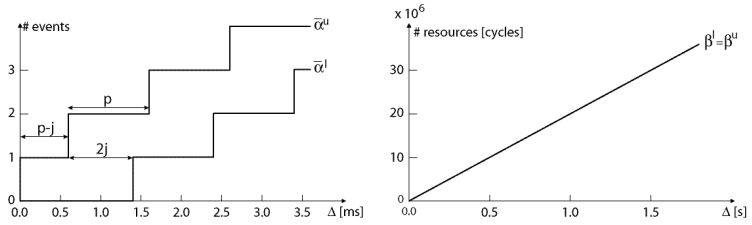


Fig. 2. An abstract event stream (left) and an abstract resource model (right).

As defined above, the arrival curves $\bar{\alpha}^u$ and $\bar{\alpha}^l$ denote the number of events that arrive on an event stream in a given time interval. However, for performance analysis we are not so much interested in the number of events that arrive, but rather in the resource demand these arriving events produce on an architectural element. We denote *resource based arrival curves* using α^u and α^l. Here, we use a very basic method to obtain resource based arrival curves that is by multiplying the event based arrival curves with a constant that represents the resource demand of a single event. Similarly, we obtain event based arrival curves by dividing resource based arrival curves with the resource demand of a single event. More advanced methods for these transformations that allow arriving events to be of one of several different event types, each having a different resource demand, are described in [18] and [19].

Resource Model A resource model is a pair of functions $\beta^l : \mathbb{R}_{\geq 0} \rightarrow \mathbb{N}$ and $\beta^u : \mathbb{R}_{\geq 0} \rightarrow \mathbb{N}$. For any value $t \in \mathbb{R}_{\geq 0}$, β^l (which is called the lower service curve) returns the *lower bound* on the resource capacity that can be observed in *any* time interval of length t. Similarly, β^u (which is called the upper service curve) returns the *upper bound* on the resource capacity. The service curves are related as follows:

Definition 2 (Service Curves). *Let $C[s,t)$ denote the number of processing or communication units available from a resource over the time interval $[s,t)$, then the inequality*

$$\beta^l(t-s) \leq C[s,t) \leq \beta^u(t-s), \forall s < t \tag{2}$$

holds, where $\beta^l(0) = \beta^u(0) = 0$.

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. We may also model communication resources, where the service curves describe the amount of transmittable bits in a given time interval.

Example 2. The right side of Figure 2 shows the abstract model of a processing resource. From the upper and lower service curves β^u and β^l, we see that the processing unit is unloaded and has a fixed processing capacity of $20 \cdot 10^6$ cycles per second.

2.2 Performance Components

Performance components are the basic building blocks to construct a performance model. They define the semantics of how application tasks are executed on architectural elements and at the same time build the basis for performance analysis. For our abstraction, an incoming event stream, represented as a set of upper and lower arrival curves, flows into a FIFO buffer in front of a *performance component*. The performance component is then triggered by the events of this buffered incoming event stream. It generates events on an outgoing event stream while being restricted by the availability of resources, represented as a set of upper and lower service curves. Resources that are not consumed by the performance component will be made available again on the resource output of the performance component, again represented as a set of upper and lower service curves.

We describe and analyze such a component using Real-Time Calculus. Figure 3 shows the set of equations that describe the processing of abstract event streams and resources by a performance component. For more details about these formulas see [8], [1] and [9].

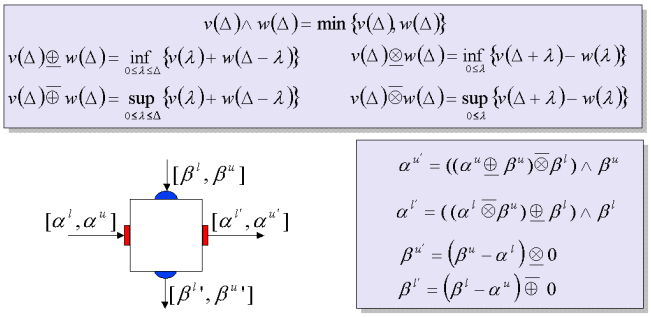


Fig. 3. A basic performance component with abstract models as input and output and Real-Time Calculus to process internal transformations.

2.3 System Composition and Analysis

Performance components can be connected into a network according to the model of a system architecture. Event flows that exit performance components (denoted by outward horizontal arrows) can be connected to an event flow input of another performance component. Similarly, resource capacity that is not consumed by a performance component (denoted by an outward vertical arrow) can be connected to a resource input of another component. Together with the models of the system resources (service curves) and incoming event streams from the environment (arrival curves), we then obtain a performance model of a complete system that can be used for performance analysis.

Scheduling policies on a resource can thereby be expressed by the way we link and distribute resources to performance components. In the most basic case, the performance component with the highest priority may use the complete resource capacity of an architectural element (denoted by an inward vertical arrow directly from the initial service curves of an architectural element). The performance component with the second highest priority then only gets the resources that were not consumed by the highest priority component (denoted by a vertical arrow from the higher priority component to the lower priority component), and so on. This way of linking corresponds to preemptive fixed priority scheduling. For more details on this and other scheduling policies see [8].

Figure 4 shows the performance model of a system, where the events of two incoming event streams A and B are processed on a shared *I/O* component and on a non-shared *DSP* and micro-processor μP respectively. All components are connected by a shared *Bus*. The linkage of the different resources with the performance components corresponds to the use of a preemptive fixed priority scheduler, where the processing of stream A has a higher priority than the processing of stream B.

In Real-Time Calculus, upper bounds of the maximum delay experienced by an event at a performance component and the maximum number of backlogged resource demands from events of the stream that is waiting to be processed, are given by the following inequalities:

$$delay \leq \sup_{t \geq 0} \left\{ \inf\{\tau \geq 0 : \alpha^u(t) \leq \beta^l(t+\tau)\} \right\} \tag{3}$$

$$backlog \leq \sup_{t \geq 0} \{\alpha^u(t) - \beta^l(t)\} \tag{4}$$

We can use these inequalities to compute the upper bounds of the maximum delay and maximum backlog at every performance component. We can then combine these results to obtain the delay experienced by an event at the complete system and to compute the required size of all buffers in the system. Vice versa, we may for example also use the inequalities to compute the minimum required resource capacity of a resource when a constraint on the delay or the backlog is given.

The upper bounds to the maximum delay and maximum backlog at an example performance component are indicated in Figure 5 by d_{max} and b_{max}, respectively. Intuitively, we can see in this figure that d_{max} and b_{max} are bounded by the maximum possible resource demand (upper resource based arrival curve) that is processed with the minimum possible amount of available resources (lower service curve).

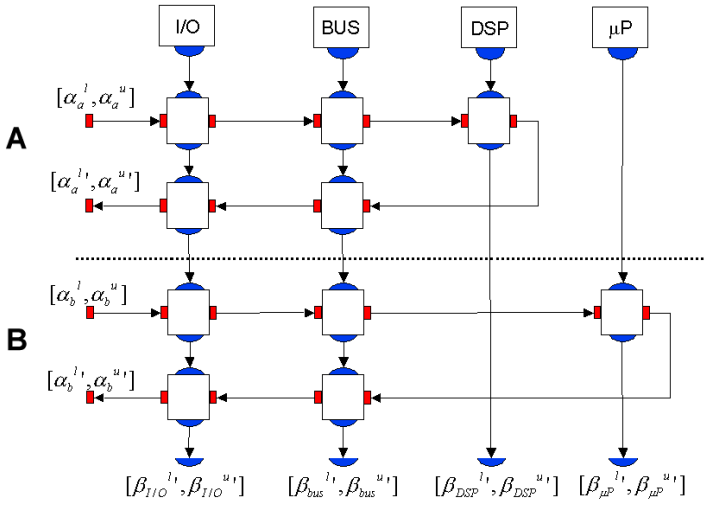


Fig. 4. Performance model of a system with two incoming event streams.

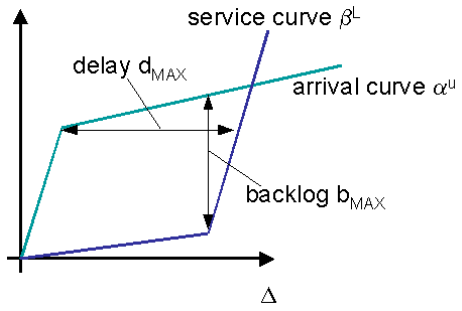


Fig. 5. Delay and backlog obtained from arrival and service curves.

3 Application Scenario: A 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. An overview of the system is presented in Figure 6, 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 (for example tacho signal and GPS, both 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 (Traffic Message Channel). TMC is broadcasted along with the audio signal of radio channels.

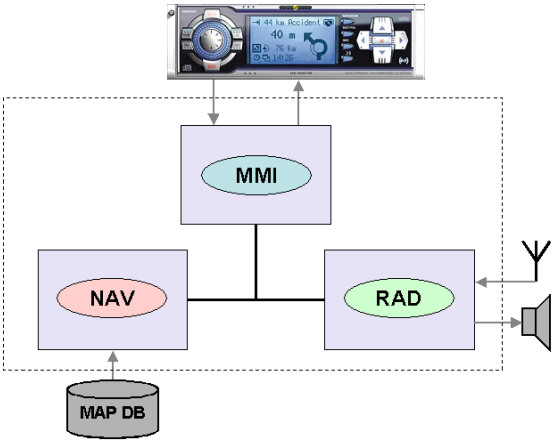


Fig. 6. High-level overview of a distributed radio navigation system

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 the key usage scenarios and main (sub)system functions
2. quantify the event rates, message sizes and execution times in each scenario
3. identify the resource units and the way they communicate or interact
4. quantify the capacity of the resource units and the communication infrastructure
5. compose a Modular Performance Analysis (MPA) model, calculate and evaluate

A general description of a new product is normally made during the initial phase of a typical industrial product creation process. For example, an *Operational Concept Description* from the IEEE 12207 system life cyle standard [20] may be produced. Such a document does not only list functional and non-functional requirements, restrictions and boundary conditions 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 Message Sequence Charts are analysed 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 analysed, 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 Message Sequence Charts in this paper is realistic. It simulates current practice, where coarse assumptions are made whenever a new design is composed initially. 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 for example 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 synchronised 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. In particular rapid volume changes can cause unwanted side-effects, such as clipping, which are disturbing to the user. This scenario is shown in detail in Figure 7. Note that three operations are identified, `HandleKeyPress`, `AdjustVolume` and `UpdateScreen`. Execution times, event rates and message sizes are estimated and annotated in the Message Sequence Chart together with the principle timing requirements applicable to this scenario.

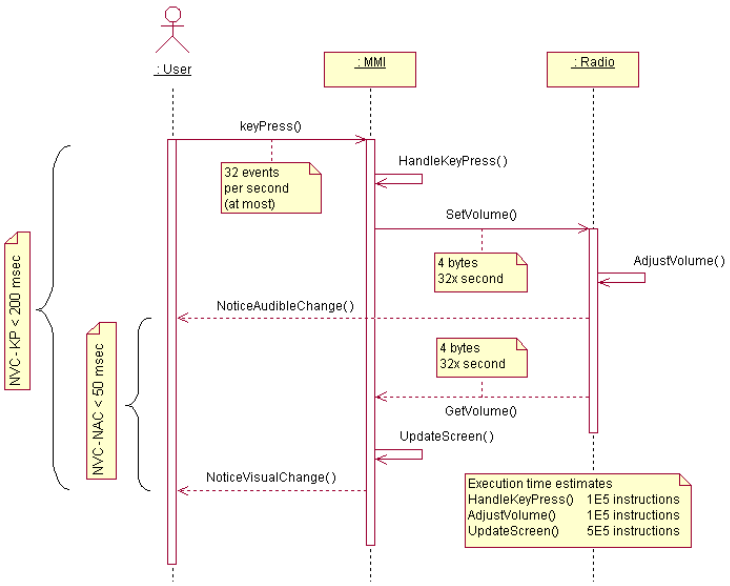


Fig. 7. Annotated Message Sequence Chart for the "Change Volume" scenario

2. “Address Look-up” - Destination entry is supported by a smart “typewriter” style interface. The display shows the alphabet and the user selects the first letter of a city (or street). By turning a knob the user can move from letter to letter; by pressing it the user will select the currently highlighted letter. The same process is followed for the second and third letter and so on, until the list of potential candidates is reduced to a reasonable amount. The user is then presented with a simple pick-list of fully expanded city (or street) names instead of the alphabet. 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 Figure 8. 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.

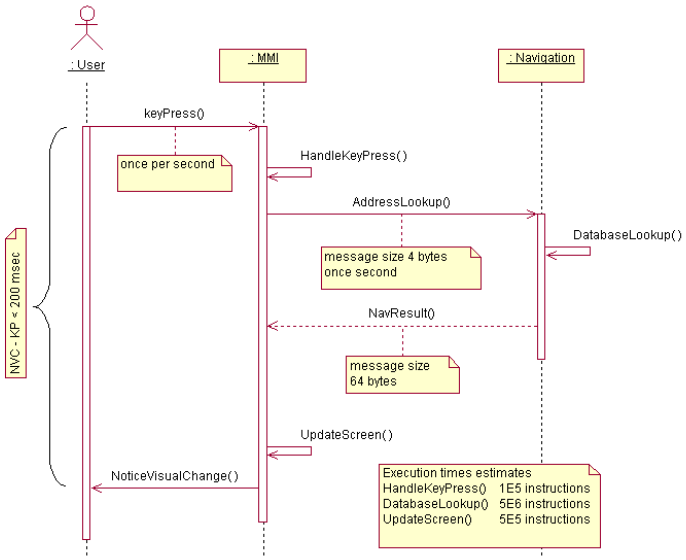


Fig. 8. Annotated Message Sequence Chart for the “Address Look-up” scenario

3. “TMC Message Handling” - Digital traffic information is very important for in-car navigation systems. It enables features such as automatic replanning of the planned route if 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 route. RDS TMC is such a digital traffic information service. TMC messages are broadcasted by radio stations together with stereo audio sound. Traffic information messages are received while you are listening to your favorite radio station. RDS TMC messages are encoded; only problem location identifiers and message types are transmitted. The map database contains two look-up tables that allow the re-

ceiver to translate these identifiers into map locations and human readable RDS TMC message texts. Sometimes multiple messages need to be combined to reconstruct a full message. The TMC message handling scenario is shown in Figure 9. Note that the radio only receives the TMC messages, decoding is performed on the navigation subsystem because the map database is needed for that. Only fully decoded and relevant messages are forwarded to the user.

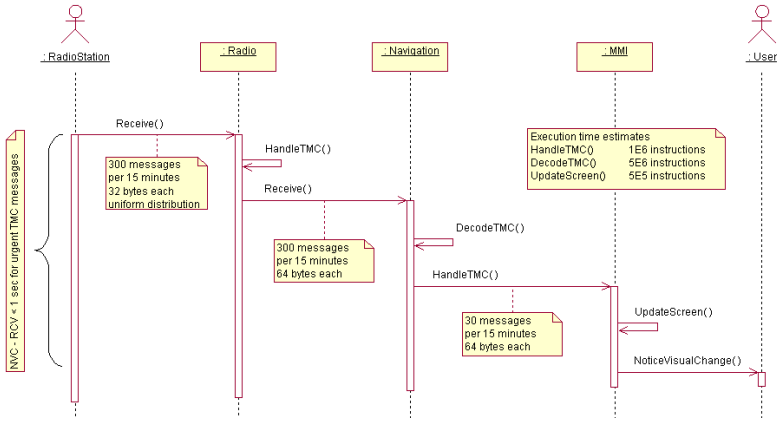


Fig. 9. Annotated Message Sequence Chart for the “TMC Message Handling” scenario

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 address at the same time. Nevertheless, all global timing requirements must be met by the proposed system architecture. The architecture shown in Figure 6 suggests to assign the three clusters of functionality each to its own processing unit. The processing units are interconnected by a single communication bus. Under which circumstances does this architecture meet our requirements and is this the best architecture for our application?

Figure 10 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 step 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 datasheets 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. The execution time estimate for this transfer function is simply set to zero for all proposed architectures except architecture B during evaluation.

Sufficient information is now available to construct an MPA model for the bus architecture. This model is shown in Figure 11, for architecture A. Note that the resources occur as column headings in the model. The communication bus is also a resource.

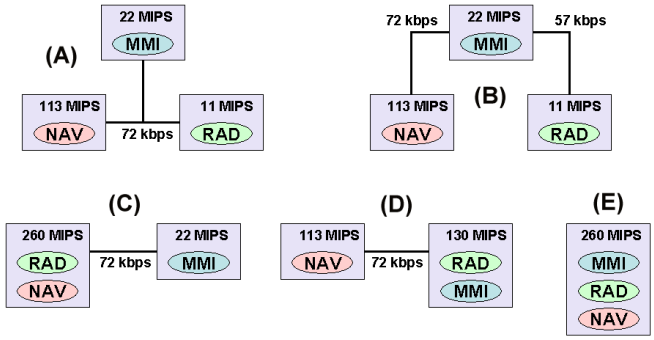


Fig. 10. Alternative system architectures to explore

Observe that all outgoing horizontal arrows from the basic performance components of NAV, RAD and MMI (representing their respective output message flows) connect to an input of a BUS basic performance component, which encodes the notion of the *shared* communication medium because they occur in the same column. In the case of architecture B, two BUS resources (columns) would exist in the model instead of one.

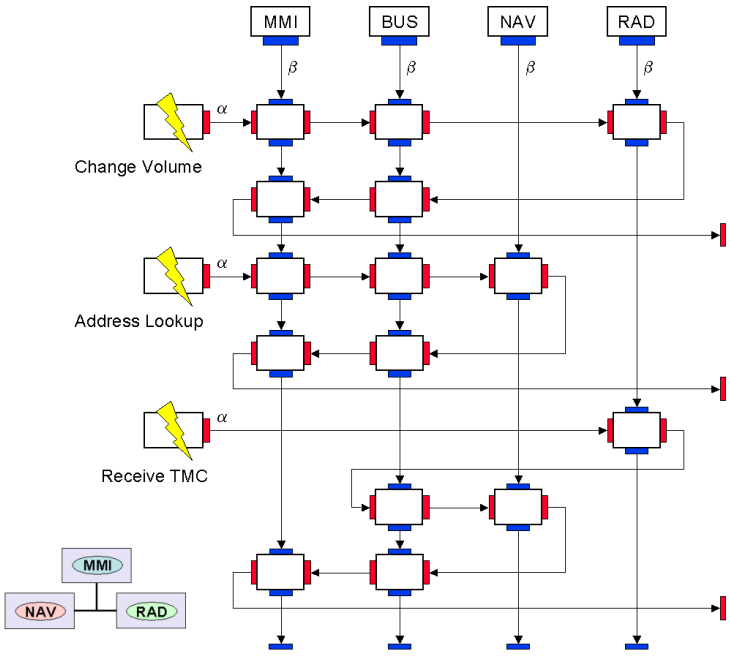


Fig. 11. MPA model for system architecture (A) of Figure 10

The scenarios (Message Sequence Charts from the previous modeling step) are depicted as the rows of the model. Each row starts with a load scenario symbol, which is connected to the input of a basic performance component. The event flow of the Message Sequence Chart can be followed, in horizontal direction, in the MPA model. Take for example the "Address Look-up" scenario from Figure 8. Events arrive at the MMI where `HandleKeyPress` is executed and the result is forwarded, via the communication bus, to NAV. `DatabaseLookup` 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 Message Sequence Charts. The resource model data (β) is extracted from the system architecture diagram. Figure 12 shows the load scenario data for "Address Look-up" and the resource model data for the NAV computation unit.

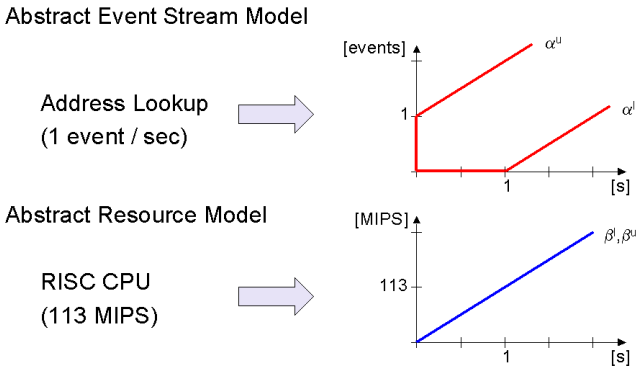


Fig. 12. Example event stream (load scenario) and resource model

The order of the rows in the MPA model determine the priority of the event stream. In this case, the "Change Volume" scenario is assigned a higher priority than "Address Look-up". 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 order of the load scenarios in the vertical direction. An 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 Modular Performance Analysis presented in section 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 section 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 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 a system architecture accepted by our analysis fulfills all system requirements in reality.

Without any attempts to optimization, computing the results shown in the figures in this section took less than $30\ s$ per figure on a Pentium Mobile 1.6 GHz using Mathematica 5. The only exception are the plots in Figure 17 that took around $140\ s$ each to compute.

4.1 Analysis Methods

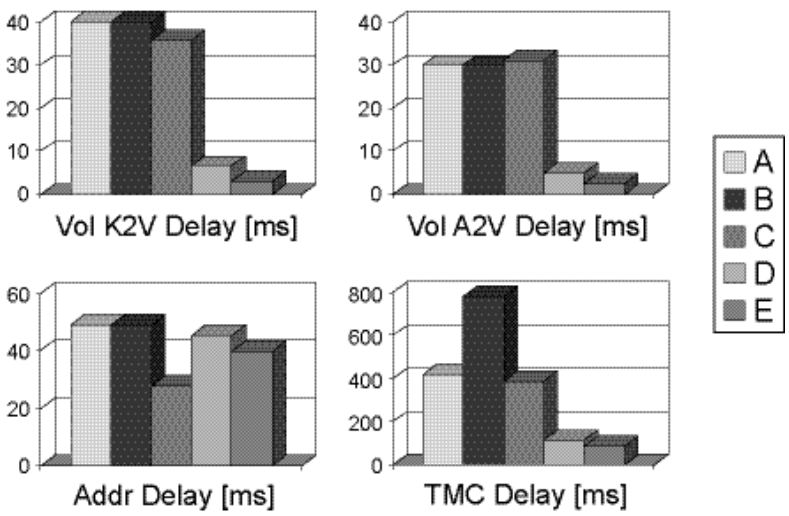
To compute the end-to-end delay of an event stream in a given system architecture, we first construct the performance model of the given system architecture. The event streams and the system resources are thereby modeled as shown in Figure 12. By applying the transformations given in Figure 3 at every performance component of the performance model, we eventually end up with the output event streams and the remaining system resources. We then use formula (3) 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 to obtain a hard upper bound on the maximum end-to-end delay of the event stream in the given system architecture.

4.2 Design Problems and Results

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

Problem 1. In Figure 10, 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" scenario and the "Address Look-up" scenario 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 take the performance model depicted in Figure 11 and evaluate it twice, once with the arrival curves of "Address Lookup" set to zero and once for the arrival curves of "Change Volume" set to zero. For both cases, 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 for all end-to-end delays the larger value obtained from the two analysis runs. From the results presented in Figure 13, we see that all proposed system architectures fulfill the requirements 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, which is very intuitive because there is much less communication overhead.



- `K2V` represents the delay between `keyPress()` and `NoticeAudibleChange()` in Figure 7
- `A2V` represents the delay between `NoticeAudibleChange()` and `NoticeVisualChange()` in Figure 7
- `Addr` represents the delay between `keyPress()` and `NoticeVisualChange()` in Figure 8
- `TMC` represents the delay between `Receive()` and `NoticeVisualChange()` in Figure 9
- Note that all requirements are met by all architectures

Fig. 13. Upper bounds to the end-to-end delays for the five proposed system architectures.

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 Figure 14. 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 Figure 14 shows us that if we increase the data rate of the "Change Volume" scenario by $1\ \%$ (i.e. to $32.32\ events/s$), the end-to-end delay of the TMC message handling increases by $2.76\ \%$ of its specified maximum end-to-end delay (i.e. $2.76\ \%$ of $1000\ ms$ or $27.6\ ms$).

From the results shown in Figure 14, 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 $8\ \%$, 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

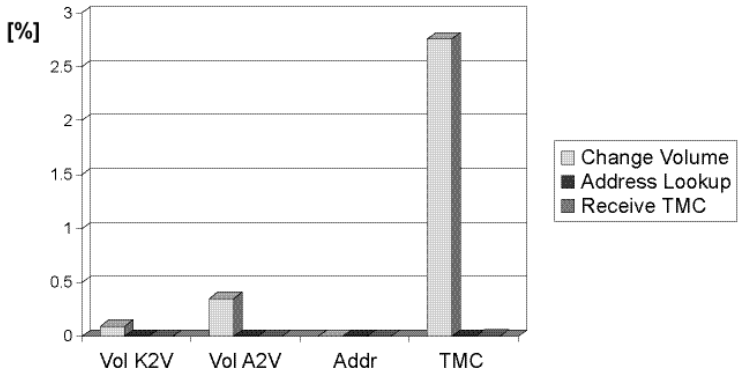


Fig. 14. Sensitivity of architecture A towards changes in the system input data rates.

sensitivity results are shown in Figure 15. 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 Figure 15 we know that if we decrease capacity of the MMI processor by $1\ \%$ (i.e. 21.78) MIPS, the end-to-end delay of the TMC message handling increases by $3.19\ \%$ of its specified maximum end-to-end delay (i.e. $3.19\ \%$ of $1000\ ms$ or $31.9\ ms$).

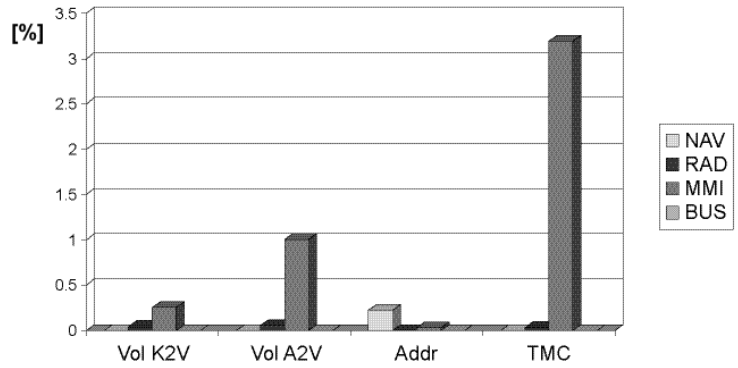


Fig. 15. Sensitivity of architecture A towards changes in the system resource capacities.

From the results shown in Figure 15, 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 Figure 16.

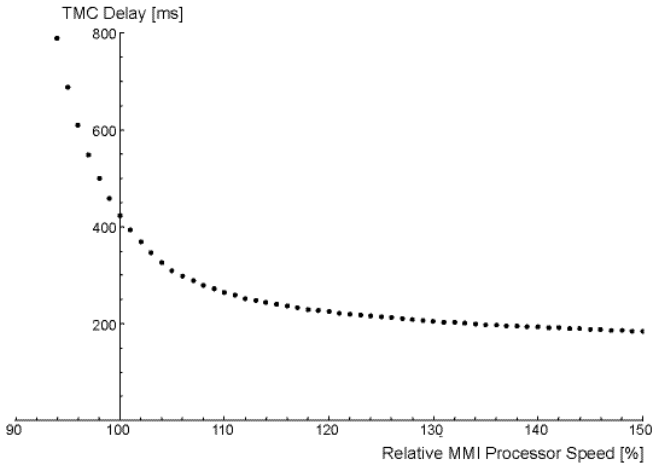


Fig. 16. TMC delay versus MMI processor speed in architecture A.

From Figure 16, 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 $88\ \%$ 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 analyse the effect of individually increasing the capacity of each resource by $50\ \%$ of the maximum allowable increase in the input data rate of the "Change Volume" scenario. Remember, with the initial resource capacities, we can increase the data rate of the "Change Volume" scenario by $8\ \%$ 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 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 NAV processor capacity allows to increase the data rate of the "Change Volume" scenario by $11\ \%$. But increasing the MMI processor capacity allows to increase the data rate of the "Change Volume" scenario by $65\ \%$.

Problem 3. Suppose system architecture D gets chosen for further investigation. How should the two processors in this system architecture be dimensioned, to obtain an economic system that still fulfills all end-to-end delay requirements of the three Use-cases?

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 Figure 17.

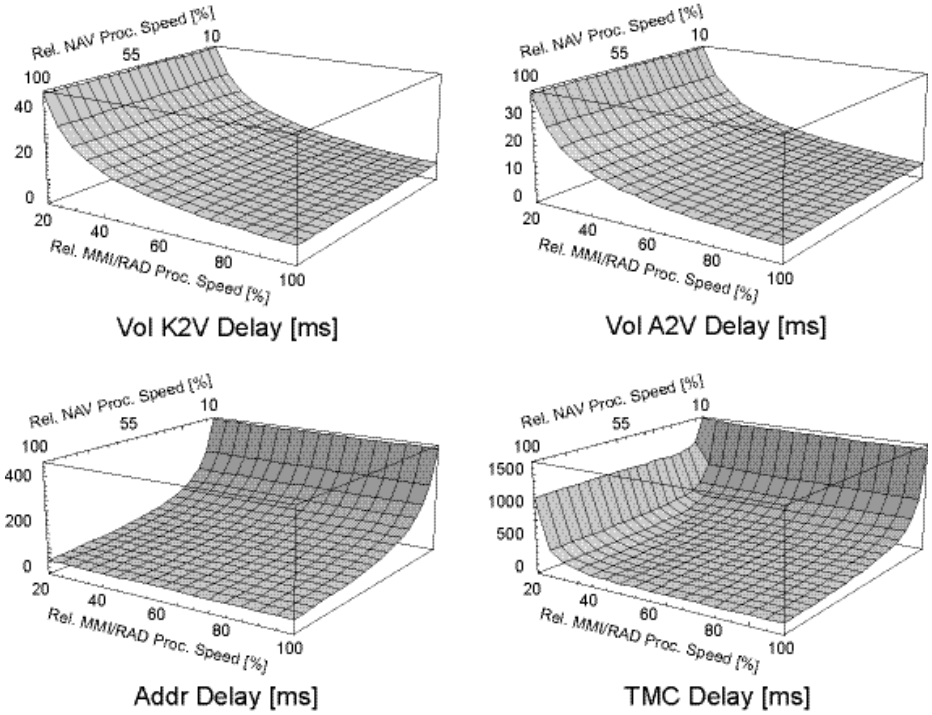


Fig. 17. Delays versus processor speed in architecture 4.

In the plots in Figure 17, 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 is 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 MIPS 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 analysed 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.

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 takes just a minute. 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 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. Robust tool support is required if this method is to succeed in industry. The Message Sequence Charts shown in this paper are currently annotated by hand; however, this should obviously become an integral part of a case tool. Alternatively, an architecture description language such as ACME [21] could be used as a specification vehicle. In both cases, automatic model extraction is required and this must be implemented. Currently, *Mathematica* is used to compute the Real-Time Calculus results. Despite its capabilities and flexibility, it is not the system architect tool of the trade, so a more dedicated solution is required. Several prototype tools are already available at ETH Zurich (among others EXPO and MOSES) which can be used as a basis for further development.

Acknowledgements. 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.

References

1. Thiele, L., Chakraborty, S., Naedele, M.: Real-time calculus for scheduling hard real-time systems. In: Proc. IEEE International Symposium on Circuits and Systems (ISCAS). Vol-

ume 4. (2000) 101–104
2. Liao, S., Tjiang, S., Gupta, R.: An Efficient Implementation of Reactivity for Modeling Hardware in the Scenic Design Environment. In: Proceedings of DAC 1997, ACM (1997)
3. Baird, M.: SystemC 2.0.1 Language Reference Manual. Open SystemC Initiative (2003)
4. Grotker, T., Liao, S., Martin, G., Swan, S.: System Design with SystemC. Kluwer Academic Publishers (May 2002)
5. The Mathworks: The Matlab / Simulink Homepage. `http://www.mathworks.com` (2004)
6. Gries, M.: Methods for evaluating and covering the design space during early design development. Technical Report UCB/ERL M03/32, Electronics Research Lab, University of California at Berkeley (2003)
7. Balsamo, S., Di Marco, A., Inverardi, P.: Model-based performance prediction in software development: A survey. IEEE Transactions on Software Engineering **30** (2004) 295–310
8. Chakraborty, S., Künzli, S., Thiele, L.: A general framework for analysing system properties in platform-based embedded system designs. In: Proc. 6th Design, Automation and Test in Europe (DATE), Munich, Germany (2003) 190–195
9. Le Boudec, J., Thiran, P.: Network Calculus - A Theory of Deterministic Queuing Systems for the Internet. LNCS 2050, Springer Verlag (2001)
10. Bacelli, F., Cohen, G., Olsder, G., Quadrat, J.P.: Synchronization and Linearity: An Algebra for Discrete Event Systems. Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons Ltd (1992)
11. Quadrat, J.P.: The MaxPlus Algebra Homepage. `http://www.maxplus.org` (2004)
12. Smith, C.U., Williams, L.G.: Performance Engineering Evaluation of Object-Oriented Systems with SPE*ED^{TM}. Number 1245 in Lecture Notes in Computer Science (LNCS). In: Computer Performance Evaluation: Modeling Techniques and Tools. Springer-Verlag, Berlin, Germany (1997)
13. Cortellessa, V., Mirandola, R.: Deriving a queueing network based performance model from UML diagrams. In: Proc. 2nd International Workshop on Software and Performance (WOSP), Ottawa, Ontario, Canada (2000) 58–70
14. Lieverse, P., van der Wolf, P., Vissers, K., Deprettere, E.: A methodology for architecture exploration of heterogeneous signal processing systems. Journal of VLSI Signal Processing for Signal, Image, and Video Processing **29** (2001) 197–207
15. Pimentel, A.D., Hertzberger, L.O., Lieverse, P., van der Wolf, P., Deprettere, E.F.: Exploring embedded-systems architectures with Artemis. IEEE Computer **34** (2001) 57–63
16. Živković, V.D., de Kock, E., van der Wolf, P., Deprettere, E.: Fast and accurate multiprocessor architecture exploration with symbolic programs. In: Proc. 6th Design, Automation and Test in Europe (DATE), Munich, Germany (2003)
17. Cruz, R.: A calculus for network delay. IEEE Trans. Information Theory **37** (1991) 114–141
18. Maxiaguine, A., Künzli, S., Thiele, L.: Workload characterization model for tasks with variable execution demand. In: Proc. 7th Design, Automation and Test in Europe (DATE). (2004)
19. 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)
20. IEEE/EIA: ISO/IEC 12207:1995 Standard for Information Technology – Software life cycle processes. The Institute of Electrical and Electronics Engineers, Inc. (1998)
21. Garlan, D., Monroe, R.T., Wile, D.: ACME: Architectural description of component-based systems. In Leavens, G.T., Sitaraman, M., eds.: Foundations of Component-Based Systems. Cambridge University Press (2000) 47–68