

# Automotive Real Time Development Using a Timing-augmented AUTOSAR Specification

Oliver Scheickl, Michael Rudorfer

BMW Car IT, Munich  
oliver.scheickl@bmw.de

**Abstract:** The AUTOSAR software architecture standard is enjoying increasing interest and broad acceptance. However, there is still much more potential for the exploitation of AUTOSAR's benefits.

In this article we describe aspired development activities related to the embedded system's timing and we show, how these activities can be linked to the automotive design flow. Besides this we propose an extension of AUTOSAR towards the possibility to specify both the system's timing behaviour and its real time constraints. This is mandatory to realise the described timing activities within AUTOSAR-compliant development.

**Keywords:** Real Time Systems, AUTOSAR, System Model, Timing Analysis, Scheduling

## 1. Introduction

Today's modern automobiles feature multifaceted complex functions based on electronics and software services which are typically provided by interactive distributed automotive real time systems.

The development of these systems is a complex task mainly for the following reasons: first, functions are often **distributed** across the system and may involve several electronic control units (ECUs) and communication busses for their execution; second, each single ECU often participates in many different functions what leads to a **mutual influence** of the functions on each ECU; third, subsystems are often developed by **different teams** and suppliers and have to be integrated; fourth, a decreasing number of ECUs comprise a **higher degree of integration** on each ECU; and finally, from a real time point of view, another reason is that the distributed functions often have to fulfil **stringent timing constraints** to function properly. A continuation of outstanding functional innovations, markedly in the regions of safety, comfort and chassis functions, is expected for future automobile generations. The industry and research community is searching for methods to cope with the increasing complexity of automotive system design.

To be prepared for these upcoming challenges major OEMs and tier-1 suppliers founded the AUTOSAR development partnership in the year 2003, amongst whose members are many automotive OEMs, suppliers as well as software and hardware

companies now. The main goal of the initiative is to define a methodology that supports a distributed function driven development process and to create a standard for the software architecture of automobile ECUs. This standard includes basic software, application software structure and component interfaces. By using a standardised formal specification model for the automotive software structure that all development participants have committed to, AUTOSAR is expected to bring several benefits to the industry concerning the increasing development complexity described above. It is stated that the standard will enable several high-level benefits like smooth integration of third party software and supplier subsystems, easier reuse of software and hardware components and seamless application of diverse development tools [1].

However, as we highlight in this paper, AUTOSAR as a formal system architecture model can gain much more benefit for automotive real time development than the usually promoted high-level benefits. But for that statement to be true we claim that an AUTOSAR system model also must include the system's **timing behaviour** and **timing constraints**. The specification does not address these aspects adequately now. Furthermore, the consideration of the system's timing is a prerequisite for a successful application of the standard and for its continuous deployment within the automotive industry. As an example, seamless integration of a supplier's software components can only be achieved with detailed knowledge of the component's timing behaviour. In general it must be possible to support real time development as a whole from the specification of abstract timing constraints to their concrete verification using the necessary timing properties given by the implementation and supplied by the system architecture model.

As this article focuses on timing activities during the development of automotive real time systems, we describe which typical timing activities can be identified. Besides the mentioned timing analysis activity, there can be applied several other typical timing activities based upon a timing-augmented AUTOSAR model. Their realisation can be automated due to the formal model underneath. Furthermore we explain how they are linked to the automotive design flow.

Today, the OEMs cannot use the full potential of automation at automotive software development due to a lack of common formal modelling. The tools used are mostly independent and isolated, what makes a continuous design flow difficult [2]. This article illustrates an aspired development flow, from the viewpoint of real time development.

The rest of this article is structured as follows. Section 2 illustrates the automotive real time design flow induced by AUTOSAR in an abstract manner. Based on the design flow we identify possible timing activities in Section 3. We explain each activity in detail together with its benefits. Furthermore, related work for these activities is mentioned. Section 4 gives a brief overview of our prototypic extension of the AUTOSAR meta that allows to capture timing constraints and behaviour. We also give an overview of our prototypic implementation of a timing verification tool. Finally, Section 5 summarises the key messages of our work and gives a short outline of future work.

## 2. The Automotive Design Flow

### 2.1 An Automotive Design Flow Abstraction

In this section we detail how the AUTOSAR-driven design flow of automotive real time system development is structured in principle. We will refer to this design flow later in the article. The outlined design flow is an excerpt from the complete AUTOSAR design flow. Of course, the flow is an abstraction and idealisation of the real development process. Though, we use this depiction to explain the process in a simplified way and to identify the main timing properties of an automotive real time system.

The excerpt of the automotive design flow used in this article is illustrated in Fig. 1. The figure is separated in two major parts. The upper part shows the concurrent development of the software architecture and the hardware topology.

The software architecture development includes the definition of software components (SWC), that implement the functionality of high level vehicle functions. Due to the distributed character of functions across the board net, the software components mostly interact with other components. In the software architecture abstraction level the interaction is not expressed in terms of concrete communication (e.g. using a communication bus or local function calls), but as an abstract interaction. AUTOSAR calls this abstraction the *Virtual Function Bus* (VFB). Besides the software point of view, the hardware topology specification defines the board net hardware that is used to execute the software. The specification includes a description of the ECUs (memory, CPU, hardware details) and of the communication busses (bandwidth, configuration) that interconnect the ECUs.

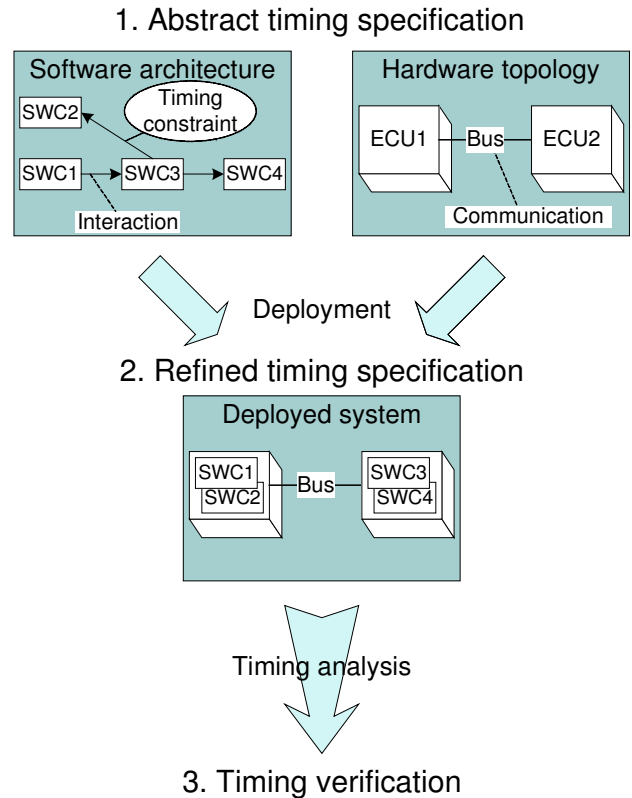


Figure 1: A simplified abstraction of the automotive real time design flow

The lower part of Fig. 1 depicts the system after the deployment step. The deployment allocates software components to ECUs that execute them. Now a concrete communication results from the distribution of the software components across the board net. The system now realises interaction of software components as local communication (components are on the same ECU) or as bus communication (components are on different ECUs).

Note, that in a real development process the depicted design flow is applied iteratively. The presented concepts in this paper abstract from a actual process characteristic.

Now, the main principle of AUTOSAR is to standardise both this development methodology and a common meta model. The well defined software structure enables OEMs to bundle an overall formal system specification and to cooperate with suppliers on a standardised format. Thus, corresponding models contain all the information of the design flow described above, e.g.:

- software component structure
- hardware topology
- basic software configuration
- deployment information

What is not yet fully included in the meta model are both the system's timing properties, that influence its timing behaviour, and the timing constraints, that the system must fulfil.

## 2.2 Timing Information Located in the Design Flow

We refer to *timing information* as the sum of a real time system's *timing properties* and *timing constraints*. Roughly speaking the software architecture and hardware topology specifications include the specification of timing properties of software and hardware entities. Additionally, the software architecture specification must include timing constraints, e.g. on the abstract level of component interaction (see Fig. 1). After deployment the system must fulfil all specified timing constraints with the specified timing properties, i.e. with its specified behaviour. This is verified by means of a timing analysis procedure, leading to a timing verification. Note, that most timing properties can not be determined before deployment. In the following we list the typical timing properties and map them to the design flow depicted in Fig. 1.

- Software architecture
  - Execution period
  - Execution precedence
  - Software interrupt pattern
- Hardware topology
  - CPU clock rate (simplified)
  - Bus bandwidth
  - Hardware interrupt pattern
- Deployed System
  - Mapping of runnable entities to tasks
  - Mapping of data elements to bus frames
  - Task worst/best case execution time
  - Event-triggered cluster
    - Task priority
    - Task pre-emption
    - Bus frame priority
  - Time-triggered cluster
    - Global cycle period
    - Static bus schedule
    - Static ECU schedules
    - Task offset

The following list contains typical timing constraints of an automotive real time system. They are part of the software architecture specification.

- Maximum (or minimum) response time
- Maximum calculation time
- Synchronicity

Of course, the listing of timing properties can be expanded by going into more detail at software and hardware implementations. By doing so, more and more properties that influence the system's timing behaviour may emerge. On the software side, one example is the configuration of basic software modules, e.g. memory services, ECU state management or diagnosis. Regarding the hardware, examples for more detailed timing properties are buffer sizes of communication interfaces as well as buffer strategies. Both influence the time interval until data is actually sent. In principle, an almost arbitrary refinement of these timing properties is possible. It is important to find an appropriate level of abstraction. Otherwise the modelling effort may exceed a reasonable level and model application will be impaired in practice.

An overall formal system architecture model, e.g. given by AUTOSAR, should contain all this timing information as well. We name such a model a *timing-augmented system model*.

## 2.3 Current Challenges regarding Timing

### Task Mapping:

One of the central element in the AUTOSAR functional software architecture are software components. These consist of a set of runnable entities. To execute runnable entities on an operating system (OS) after deployment, runnable entities must be assigned to OS tasks. The simplest solution would be to assign each runnable entity to one task. But this is not practicable due to two reasons: first, the number of OS tasks is limited; second, switching between tasks is time-consuming for an OS and should be avoided. Thus, engineers need a sophisticated method of mapping runnable entities to tasks. Today there exists no formal approach for this mapping. The engineers use their implicit knowledge about the system to obtain a suitable mapping. Of course, this approach is time-consuming and may not find the best solution.

### Static Schedules:

The definition of static schedules for the time triggered clusters of an automobile also is a non-trivial engineering task, that uses a lot of effort. The schedules of all participating ECUs and the bus have to be synchronised (see Sec. 3.6) due to common global time. As far as we know there are no tools in use that help engineers with static scheduling today. Instead engineers developed methods and guidelines that ease schedule definition by reducing the overall complexity. This is done by decoupling ECU and bus schedules using a windowing concept for communication. That is, allowing tasks to communicate only in predefined time slots. This way schedules can be developed locally and altered to a certain extend without global effects.

### Model-based Timing Analysis:

Timing analysis today is done in a rather informal way. There are no models in use that comprise timing constraints and timing properties. There is still much potential to establish formal specification and analysis techniques. AUTOSAR offers the chance to use the model of the system's software and hardware structure as basis for model-based timing analysis. Many system details, like interaction and distribution of software, that are needed by the analysis are already included in AUTOSAR models. An extension towards timing hence is obvious.

Many improvements to the current development are possible by exploiting a timing-augmented system model. Typical challenges of today's development can be handled with such a model-driven approach. In the next section we describe some model-based timing activities as possible solutions.

### 3. Automated Timing Activities to Improve Real-Time Development

The previous section described how the design flow of automotive system development looks like in general. It showed which steps in the process must be done and how they depend on each other. After this more abstract depiction we now define and summarise which tasks can actually appear, that are directly related to timing and scheduling of the real time system.

We will refer to this timing related tasks, called *timing activities*, later in this article. As we assume a formal system model that contains the system's timing information as described in the previous section, these activities could be realised as automated or semi-automated model-based approaches. This is one of the main benefits of using a formal model for timing specification.

An overview of the timing related tasks within model based real time development is depicted in Fig. 2. The figure shows several activities, each of which could be addressed during different development phases. They are arranged around the box labelled with 'Timing-augmented System Model', shown in the middle. Hence the timing-augmented system model is the basis for these activities. The assumed model-based approach either supports or even enables the activities as described in detail below.

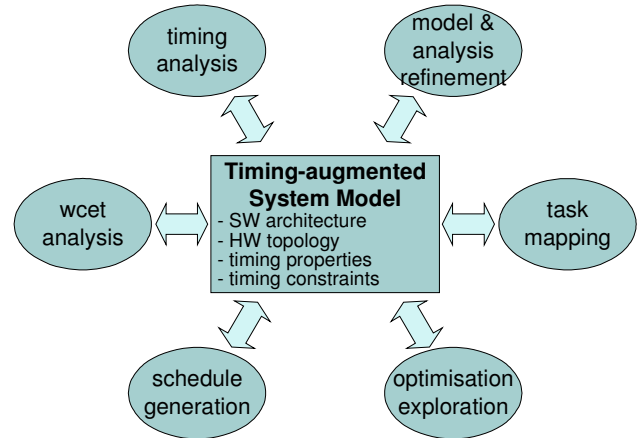


Figure 2: Timing related activities arranged around a timing-augmented system model

#### 3.1 The System Model

First of all we summarise what information a timing-augmented system model should contain. From a timing point of view, the model can abstract from certain system details that are not needed for this specific purpose. There are four main aspects the model must include to be appropriate for the timing related activities. Mainly the model must include the architecture of the system's software and its hardware topology that the software is mapped on (as described in section 2). AUTOSAR is an example of a system model that matches this model prerequisite. Furthermore the model must include the system's timing constraints and timing properties. AUTOSAR does not completely contain timing properties or constraints, respectively. We address this fact later in Sec. 4.

One example of the model's abstraction from system details not used here, is that the model must not contain the functional behaviour of the system, e.g. information on how data is computed. The main importance for the timing analysis purpose lies on the information about data dependencies of runnable entities and their corresponding timing constraints. Assuming an appropriate system model, we now describe the different timing activities based on it.

#### 3.2 WCET Analysis

Practically every real time scheduling and analysis technique is based on the a priori knowledge of the worst (or best) case execution time (WCET) of a task or runnable entity. This information is essential for all timing activities during real time system development. On the one hand, AUTOSAR as an assumed timing-augmented system model abstracts from implementations of runnable entities. On the other hand, implementation details are essential to estimate the worst and best case execution time of runnable entities. Thus timing verification can only

be achieved either on assumptions on the WCET of runnable entities during design phase, or at a later development phase, when the functional implementation is known and the WCET of the deployed runnable entities can be determined.

Despite its importance for an overall timing analysis, WCET analysis is often neglected. Timing analysis techniques assume well known WCET values and do not consider how these are generated. A lot of research has been done on WCET analysis, e.g. in ref [3]. Model-based analysis approaches could use hardware information of the model (e.g. processor speed, memory) to generate WCET values for a given implementation. Today we do not know of any research activities that deal with generation of WCET values using information in the AUTOSAR model.

### 3.3 Task Mapping

After deployment every runnable entity is assigned to a certain ECU. To execute runnable entities they must be mapped to operating system tasks. An automated task mapping approach is not in use today, but a timing-augmented system model offers the possibility to implement an automated task mapping. For example, the following decisions must be taken into account if automated by a task mapping algorithm:

- Periodically executed runnable entities can only be assigned to the same task, if the smallest period is a divider of all other periods, i.e. the periods must be harmonic.
- If the execution of a runnable entity B depends on the execution of another runnable entity A, then A should be scheduled before B within the task.
- Runnable entities with shared memory access should be mapped to the same task to avoid race conditions.

### 3.4 Timing Analysis

One activity depicted from Fig. 2 is timing analysis. Timing analysis means the verification, if the given timing properties fulfil the given timing constraints. Timing analysis is a rather general term that can be split up into two sub-activities needed to be done to obtain an overall timing analysis. It can be distinguished between the *local* and the *global* timing analysis.

The local timing analysis addresses task scheduling questions regarding an ECU or a processor on that ECU, respectively. Hence it can also be named local schedulability analysis or feasibility analysis. The challenge is to find out, whether a given set of processor tasks with certain properties can be scheduled in a way, so that all tasks meet their specific deadlines. The possible task properties, e.g.

period, deadlines or priority, are given by the considered *task model*. They vary across the different approaches available in schedulability literature.

In their fundamental work about schedulability analysis Liu and Layland [14] introduced Rate Monotonic Priority Assignment (RMPA), an approach that assigns the task with the highest period statically the highest priority. They showed that under certain conditions RMPA is an optimal scheduling policy. That means that if RMPA does not lead to an appropriate solution then no other scheduling policy can. However, they assumed a rather restrictive task model in their work which can hardly be found in real systems (e.g. with task deadlines equal to the task's period). Based on their work many other scientists published modifications of the initial approach considering less restrictive task models or arbitrary deadlines [4].

Assuming a pre-emptive task model, each task can be interrupted by higher priority tasks during its execution. This can lead to a response time greater than the basic worst case execution time of the task. Hence, for a given set of schedulable tasks, local feasibility analysis is based upon the *worst case response time* analysis of each task. Worst case response time analysis of tasks for example was addressed by Alan Burns [5] and Ken Tindell [6].

The approaches mentioned so far mainly concentrate on dynamic scheduling with pre-emptive tasks what is reflected in the corresponding task models. Dynamic scheduling can typically be found in an event triggered environment. But in the automotive domain there exists a heterogeneous board net which features both event triggered and time triggered clusters. For a node in a time triggered cluster the schedulability analysis is done implicitly by creating an appropriate static task schedule. Such a schedule fulfils the requirement of schedulability by default. How to find a valid static schedule will be addressed later when we discuss the timing activity called *schedule generation*.

So far we concentrated on local timing analysis in terms of local schedulability analysis. More challenging is the global timing analysis. In contrast to local analysis, the global timing analysis focuses on global timing dependencies across the distributed system, often called *end-to-end* timing constraints. For this analysis not only single ECUs but also bus communication as well as gateways between clusters have to be analysed. For example Tindell developed a technique to calculate the response time of messages on a CAN bus [7].

The global timing analysis has been addressed by many formal approaches. One of the two major approaches that deals with the overall distributed system is the holistic approach introduced by Tindell [8]. The work was the first to combine local

processor schedulability analysis techniques as discussed above with timing analysis of communication messages. This view leads to an analysis technique for a complete distributed system. The authors make assumptions about the task model and especially about the used communication bus (a TDMA bus). With the holistic approach global dependencies can be analysed over a distributed system.

A different kind of timing analysis was introduced by Gresser [9] and Thiele [10], who defined a formal representation of *event streams* to describe an events timing behaviour. Based on their work the compositional global timing analysis approach raised, e.g. [11]. In this approach the components, that the global system consists of, interact by event streams. This leads to a structured analysis technique that decouples complex global dependencies.

Some of the analysis techniques are available as tool implementation from different companies and research institutes as well, e.g. [12].

### 3.5 Analysis Refinement

One problem in model-based timing analysis is, that the results often are very pessimistic. The analysis of the worst case of an end-to-end constraint may offer a value, that is never reached during run time. Especially for event-triggered systems, where timing verification is not as obvious as in the time-triggered domain, the models often include pessimistic assumptions about the timing behaviour of events. Complex dependencies thus can cumulate worst case situations that are not realistic.

During informal analysis a human system designer has in-depth knowledge about the timing behaviour of the systems. He uses this knowledge to analyse the system in a way, that cannot be done by an algorithm that does not have this knowledge. The challenge of analysis refinement is to formalise this knowledge in a way, so that it can be stored in the meta model and used for formal timing analysis as well.

It is still an open challenge to develop appropriate modelling techniques to capture the automotive engineer's in-depth knowledge. A timing-augmented system model as described in this article should be able to allow for analysis refinement.

### 3.6 Schedule Generation

For the timing analysis activity described so far we implicitly assumed, that the system model already includes the complete timing properties. Today it usually is a rather complex manual task to define the appropriate timing properties, i.e. to specify appropriate task priorities, offsets or other attributes.

Though by schedule generation we mean the generation of (some) timing properties of the system instead of manually defining them (see Sec. 2.2 for an overview of timing properties). In general, this can be done in two ways:

- **Heuristic Approach:** Randomly guess (some) timing properties, e.g. by rolling the dice or using a more sophisticated heuristic algorithm. After each generation step a timing analysis as described in Sec. 3.4 is needed to analyse if a good result has been reached, i.e. if the timing constraints are fulfilled. This step can be applied iteratively until an appropriate solution has been found.
- **Constructive Approach:** Given the system's timing constraints it is also possible to automatically generate the timing properties in a way, so that the system fulfils the timing constraints, instead of making a random determination. Thus, ideally there is no timing analysis needed afterwards. We call this a *Constructive Approach*.

Regarding schedule generation, the system's timing properties are called the system's schedules, i.e. bus schedules and ECU schedules. For a generation purpose there must be distinguished between schedule generation for event-triggered and for time-triggered clusters. The generation goals differs for both paradigms, as described below.

In the **time triggered** paradigm the whole distributed system, say a time triggered cluster in an automobile's board net, has a common global time line. The challenge of schedule generation in this case is to find a suitable line up of all cyclic tasks in a way, so that every single task has exclusive processor resource access during his execution time. This is a so called non pre-emptive static task schedule that has to be determined per ECU. As additional constraint the tasks may have to meet some execution deadline that can be derived from different dependencies. Mostly the deadlines derive from communication latency issues. Similar to the ECU schedules there has to be generated a valid static bus schedule. But moreover, the main challenge of schedule generation for static time triggered clusters is that the node and bus schedules must be synchronised with each other. Thus the effort of schedule generation rises with the number of interconnected ECUs and it is known to be a NP-hard problem [13]. One approach to solve the scheduling problem for static, time triggered systems can be found in [13]. The approach uses a genetic algorithm to obtain a feasible static schedule for a synchronous communication bus. However, besides a specific cycle time, the approach does not consider other timing constraints that have to be met by the generated schedule, e.g. communication deadlines.

The dynamic **event-triggered** clusters of an automotive real time system usually use a scheduling policy with static priorities and pre-emptive tasks. Thus, the challenge of scheduling in this case is to define the static priorities of the tasks so that all tasks meet their deadlines. However, the solution of this challenge is surprisingly simple. As mentioned in Sec. 3.4 Liu and Layland [14] introduced the Rate Monotonic Scheduling and showed that it is an optimal scheduling policy. The task's priorities simply result from their period. Deadline monotonic scheduling (DMS) weakens the constraint that deadlines must be equal to periods and allows arbitrary deadlines. So far DMS offers a simple solution for assigning priorities to tasks in a way, that all tasks meet their deadlines. But still there are some key challenges for a practical application of this simple schedule generation:

- It is still not clear, how the deadline of each task should be determined. Often they depend on more complex global dependencies, e.g. an end-to-end timing constraint.
- Practical automotive applications often feature more complex task models, e.g. including non-pre-emptive tasks, interrupts or task offsets. DMS is thus not directly applicable.

Today there are mostly no tools in use that help an engineer to create schedules, nor to generate them automatically, due to the above mentioned practical complexity. Research results in that field are mostly too restrictive. The Genetic Algorithm approach in [15] for example does not cover where the deadlines of the scheduled tasks come from. Instead they are generated randomly beforehand to obtain a task set as input for the schedule generation algorithm. In another work [16] the precedence of program operations (i.e. tasks) is considered as timing constraints for non pre-emptive execution units. Both approaches show the missing practical application for challenges in automated scheduling in the automotive domain. In this field further research has to be done.

### 3.7 System Optimisation

As described in previous sections, a timing-augmented system model contains different types of information about the distributed real time system of an automobile. In the previous section we discussed, how certain parts of the model's information could be generated instead of manually determined, namely the system schedules. Besides this, further generation possibilities uncover at a closer look:

- Generation of certain system properties (other than schedules)
- Generate these properties somehow optimised

Thus, this chapter outlines the system optimisation as another development activity that is related to

timing. Automatic system optimisation can affect almost every property of the system model, theoretically. But it is important to understand, how properties depend on each other and thus influence any optimisation step and its goals.

One of the major goals of optimisation of automotive real time systems is cost reduction. This can be achieved in different ways that sometimes abstract from obvious development or production costs (e.g. maintainability, reliability). Today developers and system designers chiefly optimise implicitly (e.g. high utilisation, short WCET, etc.). But there are only insufficient metrics, informal guidelines and no formal optimisation methods available.

Automated system optimisation in general has the following three preconditions:

- A formal system model that includes timing information is needed. In this article we described such a model and its general contents (see Sec. 3.1). AUTOSAR as upcoming industry standard could be used for that purpose if timing information is added (see Sec. 4).
- Engineers have to solve, what can be optimised by formal methods. The automotive domain offers a complex design flow and a complex real time system. This results in a high-dimensional optimisation system. For formal optimisation an appropriate optimisation goal has to be defined that can be described and optimised formally. Suitable optimisation parameters have to be defined.
- Metrics have to be defined that allow for the measurement of a certain quality. This metrics express the goals of optimisation in terms of formulas.

To analyse what parameters can be optimised and how they depend on each other we figured out several design flow steps and their dependency. They are depicted in Fig. 3.

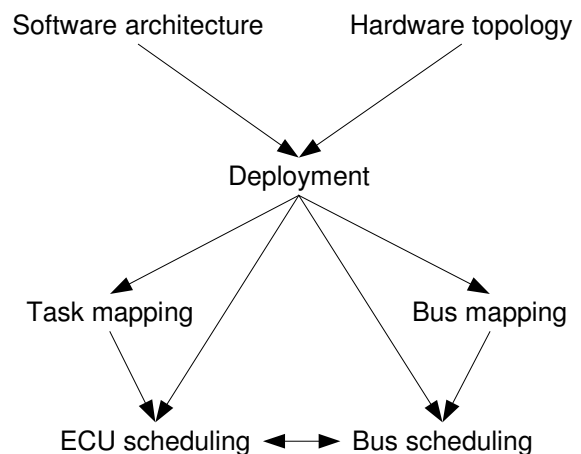


Figure 3: Design flow steps and their dependency

The figure depicts some design flow steps, which have been discussed previously in Sec. 2. Each of these steps is performed manually today. Assuming an underlying overall system model it is conceivable to use optimisation algorithms instead of manual development. But of course there are several constraints in practice:

- The design steps depend on each other, as depicted in Fig. 3. Optimising a step implies optimising (or at least reconfiguring) all subsequent steps as well. For example, if the deployment should be optimised (e.g. towards reduction of bus load) it is also necessary to reconfigure the task and bus mapping and especially the schedules.
- For each step there are often several technical or business-related constraints that prevent a completely free grade of optimisation. Especially software architecture, hardware topology and deployment decisions are constrained in practice. For example, it is not reasonable to deploy sensor software on another ECU than the sensor ECU.
- All in all, model-based system optimisation has a very high-dimensional solution space. Complexity grows with the number of involved design steps. Roughly speaking, optimisation of a single step already is NP-complete. The combination of dependent steps thus potentiates complexity. Appropriate approaches have to be developed for this purpose.

As a desirable long term goal for automotive system design and development we define the *optimising system generator*. It shall generate an optimal technical design, given optimisation metrics, models of the functional network and possible hardware. But as described above this is a very visionary goal.

A more realistic but also valuable short term goal is *semi-automated system optimisation*. Due to the difficult expressivity of technical constraints and the optimisation complexity of an overall system generator, the semi-automated approach shall concentrate on the generation and optimisation of task-mapping, bus-mapping and schedules. This design steps are mostly free of technical constraints. For the software architecture, hardware topology and deployment steps we assume some few possible solutions developed by a human engineer. Instead of automated generation and optimisation of these steps, semi-automated system optimisation concentrates on a comparison of some possible given solutions and finding the optimal scheduling for them. Optimisation complexity can additionally be reduced, if parameters can be set invariant. Towards the long term goal of a system generator, there can be included more parameters and dimensions in the formal optimisation.

System optimisation and design space exploration has been addressed by several research groups. Hamann et al. [17] introduce the concept of traffic shaping, a technique to control the event behaviour by modulating the maximum number of events per time. They introduce traffic shaping as search parameter and show their application of genetic algorithms to optimise some timing related system properties. An overview about system optimisation as well as an example is described by Racu et al. [18]. Furthermore, the authors explain how they included an existing optimisation framework in their timing analysis tool. To measure a certain quality aspect of a distributed system some formal metrics and their interpretation are given by [19]. Metrics are also discussed by [17].

#### 4. A Timing Extension for the AUTOSAR Methodology

In the previous sections of our work, we illustrated the automotive design flow and identified possible timing activities. As we already argued, the timing activities need a so called timing-augmented system model to be performed.

In our related work we already addressed timing specification and verification for AUTOSAR compliant systems [19], [20]. We developed and applied a concept for the specification of timing constraints and properties. Therefore, we extended the AUTOSAR meta-model within our prototypic AUTOSAR proof-of-concept development environment. Besides specification, we also implemented a timing analysis technique to verify, whether the modelled system fulfils its specified timing constraints. This section describes the basic concepts of our proposed AUTOSAR extension.

##### 4.1 Timing Specification

In our timing specification concept for AUTOSAR, we distinguish between high-level constraint specification and system-level property specification (see Fig. 4). The meaning of this separation is as follows: The meta-model allows for the specification of timing constraints on a level, that abstracts from details given by a concrete implementation. The concrete implementation, including deployment and scheduling for example, implies certain timing properties, e.g. task periods or worst case execution times. Timing analysis then uses this properties to verify, whether the implementation fulfils the constraints. But it is possible to specify constraints without the complete knowledge about the system-level before implementation.

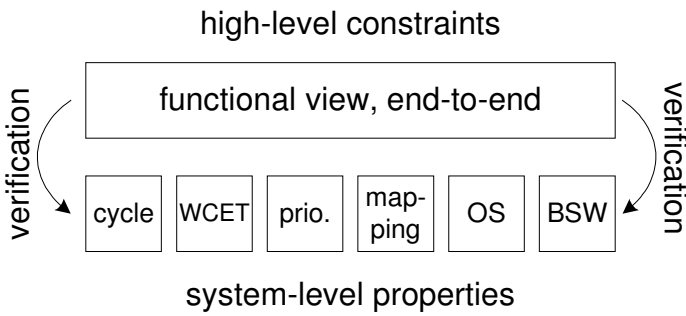


Figure 4: High-level definition of timing constraints and system-level verification of timing properties

Currently, the meta-model enables an engineer to specify end-to-end latencies, as these are typical common high-level timing constraints [19]. The specification takes place at the implementation-independent level of software components, their decomposition into runnable entities and the runnable's communication. The system-level property specification can be performed after deployment. It includes the mapping of runnable entities to tasks and the scheduling of these tasks, for example. We refer to this concrete parameters as *implementation details*. The constraints that specify end-to-end latencies are called *timing chains*.

Timing chains have two very important features (see Fig. 5). First, they are defined using so called *functional events*. Functional events are certain time instances that can be observed during system runtime. The beginning of a timing chain is a functional event as stimulus, its end is a functional event as response. Hence, a timing chain specifies the maximum (or minimum) time interval between every two consecutive occurrences of their stimulus and response events. For a detailed explanation of the causality of two functional events see [19].

The second important feature of timing chains is, that they can be decomposed into so called *subchains*. This enables an engineer to refine a constraint specification. For this purpose, the response event of a subchain must be equal to the stimulus event of its successive sub-chain. A high-level function can thus be decomposed into smaller high-level sub-functions.

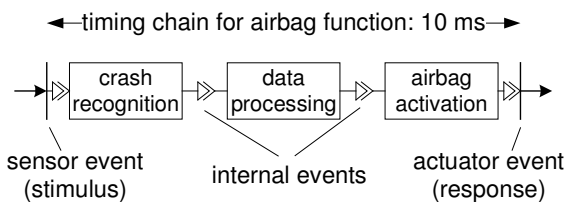


Figure 5: Function decomposition using events

## 4.2 Timing Analysis

For timing analysis, i.e. for verification of timing chains on system-level, an algorithm has to determine the concrete occurrences of these events and calculate the latency between them. For this purpose it is important, that only this functional events can be used for specification, whose occurrences can be determined. We call this *observable events*. Currently the following events can be observed:

- External sensor event (a sensor is triggered by its environment)
- External actuator event (an actuator is triggered by the system)
- Start of a runnable entity
- End of a runnable entity
- Transmission of a bus frame

We implemented a prototypic algorithm to verify timing chains for time-triggered clusters. Due to the deterministic character of such systems the prototype can calculate tight bounds for the end-to-end latencies. For the implementation we introduced the concept of *atomic chains*. Atomic chains are timing chains, that can not be refined any more. The maximum possible refinement is given by the observable events of the system.

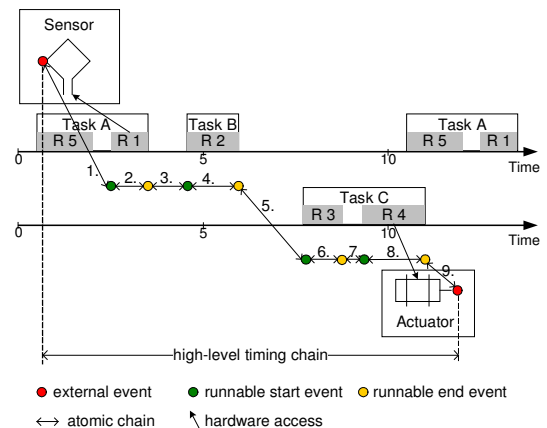


Figure 6: A high-level timing chain completely decomposed into atomic chains

A timing chain is called *computable*, if there is a causality between stimulus and response and there are enough implementation details to decompose the timing chain completely into atomic chains. Once a timing chain is computable the algorithm determines the latency for all timing chain occurrences during a global system cycle. There might be different results for different timing chain occurrences. All results have to fulfil the specified latency constraint. Fig. 6 depicts one occurrence of a high-level timing chain and its complete refinement into atomic chains. The timing chain constrains the

latency between the recognition of a sensor event and the corresponding action of an actuator.

## 5. Summary and Future Work

With our work we emphasised the importance of model-based approaches for the development of automotive real time systems. AUTOSAR as upcoming industry standard offers the possibility to capture timing constraints and properties in a common model. As we explained, this enables the further exploitation of AUTOSAR. However, the standard needs a timing extension to fulfil the requirements of a timing-augmented system model. Such a model enables several model-based timing activities, which we described in this article. Furthermore we outlined our prototypic timing extension of AUTOSAR and our prototypic tool support for timing verification.

With our current and future work we want to transfer the theoretic and prototypic principles described in this paper to practice. Hence we are involved in creating an AUTOSAR sub-group that shall actually extend the official standard towards timing specification. Besides this the practical application of the developed specification, concepts must be ensured. One of our future research activities is schedule generation and system optimisation, as discussed in sections 3.6 and 3.7.

As a result of our work this paper highlighted the importance of a formal timing approach to be prepared for the management of complexity.

## 6. References

- [1] H. Fennel, S. Bunzel, und H.H.E. al, "Achievements and Exploitation of the AUTOSAR Development Partnership," *Proceedings of Convergence 2006*, 2006.
- [2] M. Broy, "Challenges in automotive software engineering," *ICSE '06: Proceeding of the 28th international conference on Software engineering*, pp. 33-42, 2006.
- [3] J. Engblom u. a., "Worst-case execution-time analysis for embedded real-time systems," *Journal of Software Tools for Technology Transfer*, vol. 14, 2001.
- [4] N.C. Audsley u. a., "Hard Real-Time Scheduling: The Deadline Monotonic Approach," *Proceedings 8th IEEE Workshop on Real-Time Operating Systems and Software*, 1991.
- [5] A. Burns, "Preemptive priority-based scheduling: an appropriate engineering approach," pp. 225-248, 1995.
- [6] K. Tindell, "Deadline Monotonic Analysis," *Embedded Systems Programming*, pp. 20-38, Juni 2000.
- [7] K. Tindell, A. Burns, und A. Wellings, *Calculating ControllerArea Network(CAN) message response times*. 1995.
- [8] K. Tindell und J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocess. Microprogram.*, vol. 40, pp. 117-134, 1994.
- [9] K. Gresser, "An Event Model for Deadline Verification of Hard Real-Time Systems," *Proc. of fifth Euromicro Workshop on Real-Time Systems*, vol. 5, 1993.
- [10] L. Thiele, S. Chakraborty, und M. Naedele, "Real-time Calculus for Scheduling Hard Real-Time Systems," *Proceedings of Circuits and Systems*, 2000.
- [11] K. Richter und R. Ernst, *Event model interfaces for heterogeneous system analysis*. 2002.
- [12] R. Henia u. a., *System Level Performance Analysis - the SymTA/S Approach*. 2005.
- [13] R. Nossal und T. Galla, "Solving NP-Complete Problems in Real-Time System Design by Multichromosome Genetic Algorithms," *Proceedings of the SIGPLAN 1997 Workshop on Languages, Compilers, and Tools for Real-Time Systems*, pp. 68-76, 1997.
- [14] C.L. Liu und J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *Journal of the ACM*, vol. 20, pp. 46-61, 1973.
- [15] C.A.G. Pico und R.L. Wainwright, "Dynamic Scheduling of Computer Tasks Using Genetic Algorithms," *International Conference on Evolutionary Computation*, pp. 829-833, 1994.
- [16] T. Chung und H. Dietz, *Adaptive genetic algorithm: Scheduling hard real-time control programs with arbitrary timing constraints*. 1995.
- [17] A. Hamann u. a., "Design Space Exploration and System Optimization with SymTA/S " Symbolic Timing Analysis for Systems," *RTSS '04: Proceedings of the 25th IEEE International Real-Time Systems Symposium (RTSS'04)*, pp. 469-478, 2004.
- [18] R. Racu, A. Hamann, und R. Ernst, "Automotive System Optimization using Sensitivity Analysis," *IESS*, pp. 57-70, 2007.
- [19] O. Scheickl, *Spezifikation und Implementierung von Metriken zur Analyse des Echtzeitverhaltens von verteilten automotive Systemen*. TU München, Fakultät für Informatik, 2007.
- [20] M. Rudorfer u. a., "Being on time using AUTOSAR methodology," *Elektronik automotive*, vol. S2 - Special Issue AUTOSAR, pp. 12-14, 2007.