Realtime System Design Utilizing AUTOSAR Methodology

Michael Rudorfer, Tilman Ochs, Paul Hoser, Martin Thiede,
Martin Mössmer, Oliver Scheickl and Harald Heinecke

BMW Car IT GmbH
Petuelring 116
80809 Munich, Germany

Abstract

Recently significant effort has been made to create the AUTOSAR standard which specifies new concepts and software architectures for automotive systems. A major goal of the standard is to increase the reusability of functions within these systems, making their design more flexible. One aspect which has to be taken into account when reusing functions in automotive systems are the timing requirements which have to be fulfilled by the system. The task of designing and predicting if these requirements are met by the system is a rather difficult one today.

Often formal specifications of the timing requirements and the timing behavior of the soft- and hardware used within the system are missing. This provokes time-consuming testing of the integrated system for the system designer. In order to exploit the newly by AUTOSAR gained flexibility within system design a more efficient way of handling realtime requirements and evaluating their fulfillment has to be found. New model driven approaches like specified within the AUTOSAR Standard provide a basis for formally validating automotive systems with respect to their realtime requirements giving early feedback during the design phase.

The current version of the AUTOSAR meta model does not yet support realtime requirements and timing behavior to the extent needed for validating a system's correctness towards timing requirements. In this communication we extended the AUTOSAR meta model by these timing information which leads us to a more deterministic system design process.

As proof of concept, the timing and scheduling concepts developed here on top of AUTOSAR are verified using real-world examples taken from series-development which require tough timing requirements, like a headlight height control system or a cruise control system.

1 Motivation

The amount of software in today's cars is increasing rapidly. While in the 1980s and 90s there were only a few electrical functions which were more or less independent of each other we are facing more complex networks of functions today. Those functions are distributed over several electronic control devices (ECUs) in the car and provide their functionality in cooperation with each other.

The interaction leads to a rise in complexity, but apart from that, the sheer number of function makes it clear, that the paradigm of putting one function on one ECU cannot be continued as the car offers only a limited amount of physical space. The consequence is that one has to integrate more functions on one ECU, but of course without interfering each other.

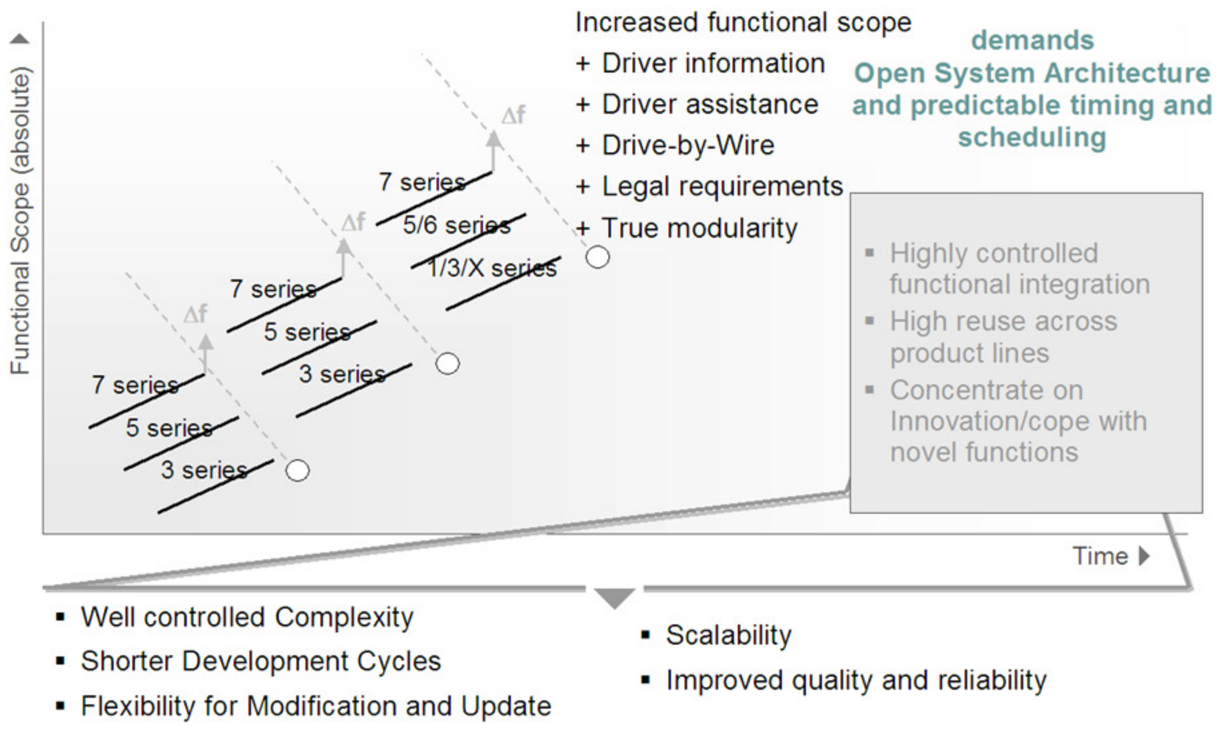
Functional Scope (absolute)
Time
Δf
Δf
Δf
7 series
5 series
3 series
7 series
5 series
3 series
7 series
5/6 series
1/3/X series
Increased functional scope
+ Driver information
+ Driver assistance
+ Drive-by-Wire
+ Legal requirements
+ True modularity
demands
Open System Architecture and predictable timing and scheduling
Highly controlled functional integration
High reuse across product lines
Concentrate on Innovation/cope with novel functions
Well controlled Complexity
Shorter Development Cycles
Flexibility for Modification and Update
Scalability
Improved quality and reliability

Fig 1 - Main Requirements for System Architecture from an OEM's Perspective.

Another use case for function integration is the reuse of software components in different car series.

A usual approach of OEMs is, that new, innovative functions are introduced in the big car series, like for example the BMW 7 series. But those functionalities are also wanted in the upcoming smaller series, so for example the current BMW 3 series has almost the same amount of functionalities as the previous 7 series. As cost limitations are even more strict in the smaller series, the functionalities cannot be redeveloped, but have to be reused.

What the car manufacturers do not want is to redevelop existing software from scratch for each new car series, only because the hardware architecture differs. They have to reuse it, but a change of hardware cannot be avoided by all means. They are inevitable as the cost structure and physical space differs from a high end limousine to a compact car.

One needs a certain flexibility in the design of the overall system, that means one should be able to put a set of software components on one ECU for car series A but put them on different ECUs for series B. This recombination of functionalities should be possible without huge manual effort in development and testing.

2 AUTOSAR

AUTOSAR proposes a solution to this challenge. AUTOSAR is a development partnership between many car manufacturers, tier-ones and tool providers and defines an open standard for an embedded middleware. This middleware shall be capable of achieving the goals of reuse, simple redeployment of functionality and abstraction from underlying hardware and communication busses.

2.1 AUTOSAR RTE Generator

A requirement for reusing software components on different ECUs that are connected to different bus systems is that there is a clear separation between the function and the communication, as the functionality should stay the same whereas the communication can differ from system to system. This separation of concerns is achieved by AUTOSAR's concept of the Virtual Function Bus and its technical realization, the Runtime-Environment (RTE).

In AUTOSAR a lot of information about software components, their interfaces, the hardware topology and the distribution of software components on ECUs are put into a model. The various aspects of the system are separated in different so-called templates. This model provides sufficient information to determine which software component exchanges information with other software components and via which information channel. The RTE is a communication layer that can handle the communication between the software components over various bus systems. As all the information about the communication is present in the model, the RTE can be generated for this very special purpose. The consequence is that the RTE is very lightweight and contains only those communication paradigms that are really required. This meets the hard requirement in embedded software not to waste time and memory.

2.2 AUTOSAR System generator

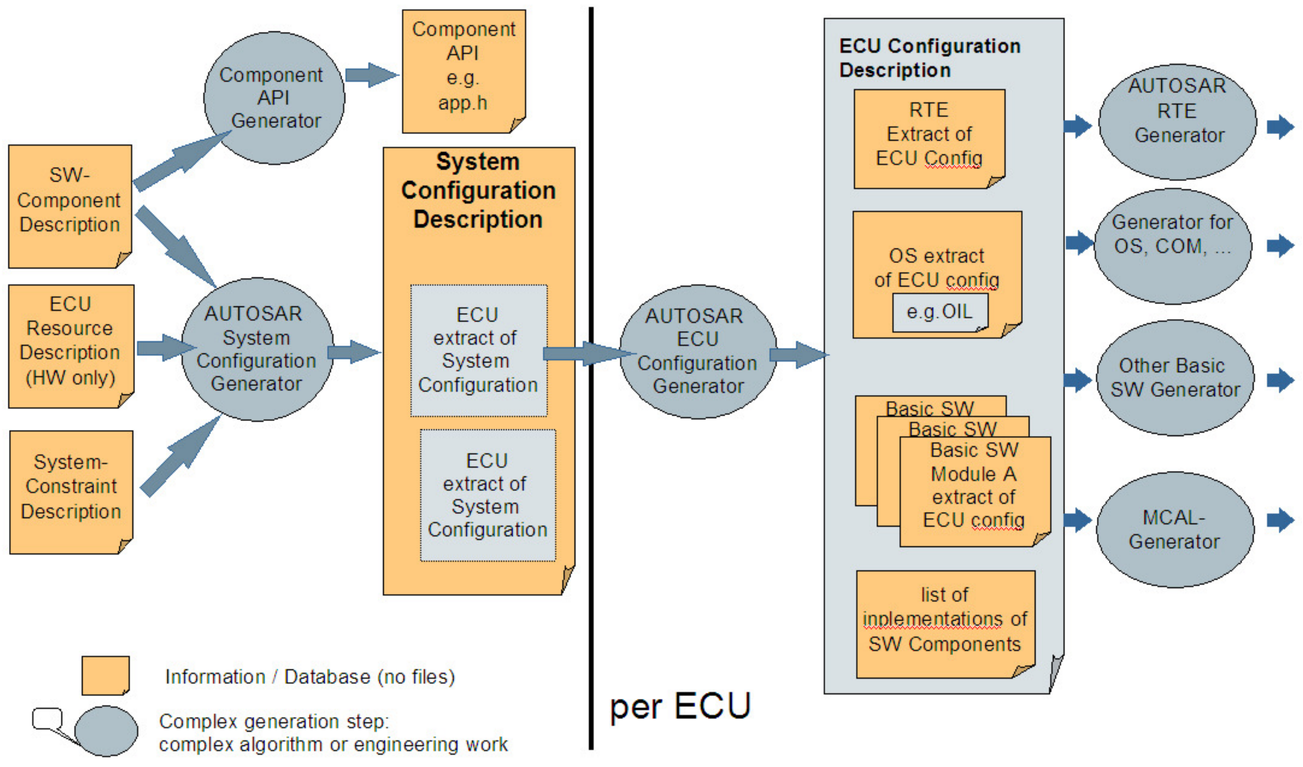
Component API Generator
Component API e.g. app.h
SW-Component Description
ECU Resource Description (HW only)
System-Constraint Description
AUTOSAR System Configuration Generator
System Configuration Description
ECU extract of System Configuration
ECU extract of System Configuration
AUTOSAR ECU Configuration Generator
ECU Configuration Description
RTE Extract of ECU Config
OS extract of ECU config
e.g.OIL
Basic SW
Basic SW
Basic SW Module A extract of ECU config
list of inplementations of SW Components
AUTOSAR RTE Generator
Generator for OS, COM, ...
Other Basic SW Generator
MCAL-Generator
Information / Database (no files)
Complex generation step: complex algorithm or engineering work
per ECU

Fig 2 – AUTOSAR Methodology (taken from Ref. [Scha05]])

The second concept aimed at in AUTOSAR is the system generator. The system generator has the goal of taking the information from the AUTOSAR templates and using it to improve the system design process. It can process the provided system details and generate the configuration of various ECUs as well as the system wide configuration like the bus schedule, the deployment and so on.

Having all those information of a system in the model can help to improve the system design process. A first step would be to verify a given system design if it satisfies all requirements concerning space consumption, communication, and so on. A more visionary approach would be to let a system generator make a proposal for the design of a system. This could be done semi-automatically or, in theory, if every piece of information about a system is contained in the model, even completely automatically. Possible aspects that could be suggested by a system generator would be the deployment of software components on ECUs, the definition of a bus schedule, the partitioning of bus signals to bus messages, the definition of a task schedule on an ECU and lots more.

3 Extending the AUTOSAR Meta-Model

As stated above, a semi-automatic system design process is possible in theory. AUTOSAR provides already many of the required information for such an approach, but it is still far from being complete. One central information that is omitted today are timing requirements.

To be able to determine, if a software component can be put on an ECU it is not sufficient to check if there are enough memory resources on that ECU. One has to make sure, that the software can run fast enough to provide its functionality and that the communication with other components happens in the given amount of time. In a realtime system a correct answer that does not arrive in time can become the wrong answer or at least useless.

Apart from the vision of a system generator, putting timing requirements into a formal model is a big advantage over the situation today. The knowledge that is present in the heads of many developers and system architects is captured and written down in a central place can be reused later. This leads to a better knowledge management.

Once the timing information is part of the meta-model it will be used to describe functionality and their timing requirements more precisely. This information can and will be used to come one step nearer to the envisioned system generator.

3.1 Timing Chains

As stated above AUTOSAR lacks information about timing requirements in its meta-model. This chapter proposes how the meta-model can be extended to hold this vital information and discusses the right extension-point.

In general there are two different kinds of timing requirements for real-time systems: on the one hand there are high-level requirements like end-to-end latencies that specify temporal behaviour of the system on the logical abstraction of system functions. On the other hand there exist timing relevant implementation details of the system-level. These must be derived from the high-level requirements to satisfy their implementation independent specification of timing constraints.

As an example let there be a function called “Automatic Door Opener”. This function opens the car’s doors automatically, when the user’s hand approaches the door knock and the key is present in a close area around the car. The hardware key transmits a digital cryptographic key which is received and verified by an ECU. One typical high-level timing requirement for this overall function could be that the doors must be unlocked within 100 milliseconds after the hand has approached. This kind of high-level end-to-end-latency timing requirements can usually be found in specification sheets. The hardware and software to be used for the technical realization of the function must therefore be able to react within 100 milliseconds. One derived system-level requirement could be that a fast enough processor for verifying the digital key is used, another one could be that a sensor with a minimum sample rate to recognize the approaching hand is used. On the software side there must be an appropriate scheduling for tasks and bus messages, as well as fast sensor and communication drivers.

As in the example the beginning and the ending of high-level requirements often correspond to events like “the hand has approached” or “digital key verified”.

Based on these ideas a meta-model to capture high-level requirements and an implementation to calculate the corresponding latencies on system-level was developed. One basic idea of this methodology is the abstract event. This is not an event in terms of causing a reaction at any receiver side of the event, but an observable happening in a passive manner while a system is running. We distinguish between internal and external events. External events represent the border between technical system and environment. The system interacts with its environment via sensors and actuators. Therefore external events represent the happening of some kind of environmental action that a sensor must recognize and an actuator must cause. For each sensor and actuator component of the system there exists one external event that can be used for the high-level requirements. To describe high-level requirements we also need some internal events that describe actions inside the system borders. At this point it is important to find a good level of abstraction for modelling high-level requirements. This means that there must be internal events that allow to describe a requirement unambiguously from stimulus to response, but that should not make the description too fine grained. Otherwise modelling and algorithmic verification will become too complex. AUTOSAR offers the concept of runnable entities that are a kind of container for functional behaviour being implemented in a programming language. They interact with input and output data elements that are part of a component interface. This makes runnable entities a good candidate for producing internal events. Thus we define the runnable start event and the runnable end event as possible events for attaching high-level requirements to.

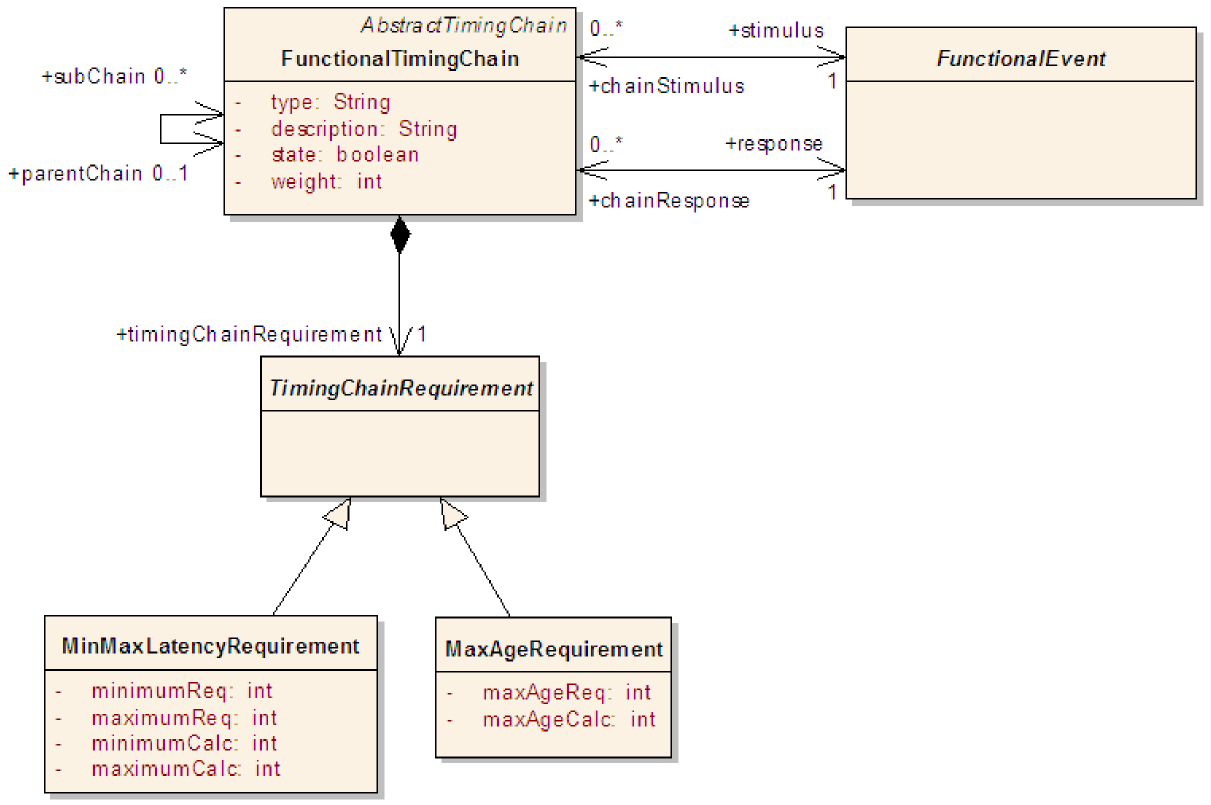

Fig 3 – Excerpt from Functional Timing Chain metamodel ([Sche07])

A so called timing chain models a high-level requirement and is set up with a stimulus and a response internal or external event. From any stimulus occurrence to its corresponding response occurrence there can be specified a maximum and a minimum latency. Similar to the above example a requirement like “from the hand being near the sensor to the digital key being verified there must be at most 70 milliseconds” there can now be modelled a Timing Chain with the sensor component’s external event as stimulus and the digital key verifying runnable entity’s ending event as response. Not in the scope of the abstraction level of this Timing Chain model for example is the time that is used by the basic software or complex device drivers, because only runnable entities are possible event sources.

The AUTOSAR meta-model already offers some technical details of the system, that are needed for verification of the high-level timing chains. These represent requirements on system-level, that must be implemented to meet the high-level chains. The question to be answered is can the high-level requirements, specified in the timing chains be met by a concrete technical realisation of the system. Examples for such details are worst case execution times of runnable entities as well as cycle times of tasks that run them and the scheduling of bus frames that carry data elements. All this system data is used by the verification algorithm, to calculate the maximum and minimum latency of a timing chain and compare it with the specified latency interval.

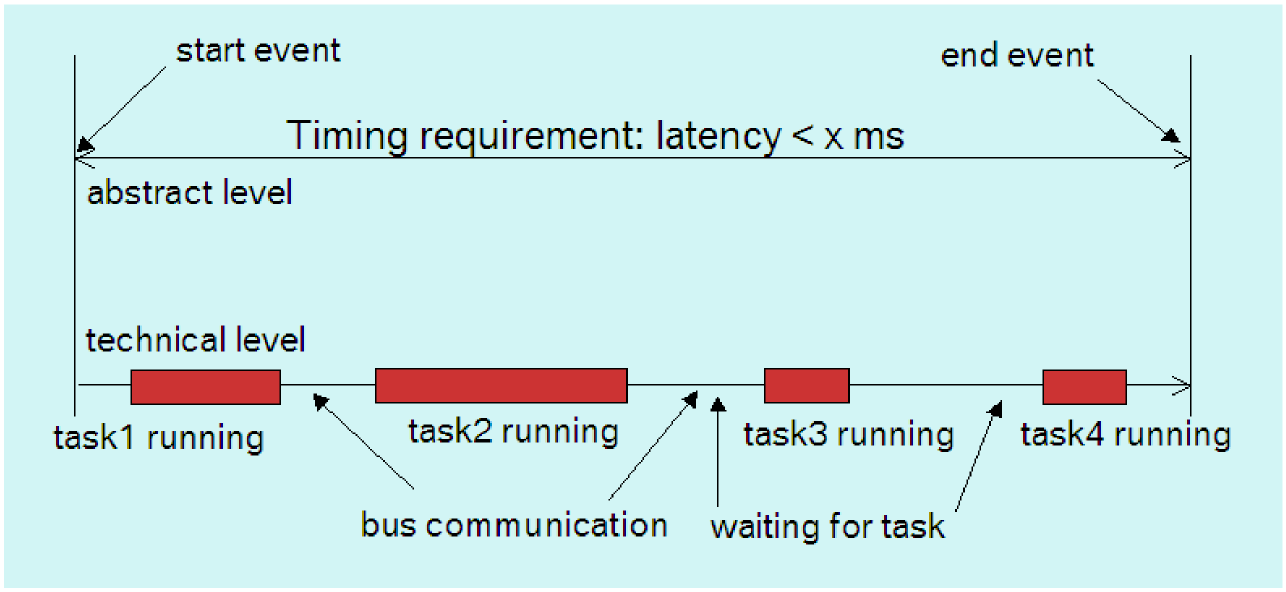

Fig 4 – Abstact level vs technical level

4 Open Issues

4.1 Extension or change of the metamodel

Another question is, where in the metamodel these information are stored. At a first glance, one would tend to place the timing chains in the software component template. At a closer look, this will not be possible. A timing chain is a requirement for the abstract function. At the design time of the function, the technical realisation is still undefined. The functionality can be provided by one or more software components, each of whom may be contained in their own SWCT. So the timing chain cannot be easily attached to one SWCT. Another idea was to add it to the system template, but this does not work either. The timing requirements are a property of the software component types. The system constraints can be different for the reuse of the same functionality in a different car series.

It will be required to investigate this template approach in further detail. Possible changes could be the introduction of a new template. This template could hold the information with references to the abstract events in various SWCTs. This would be an addition to the metamodel. Another way would be the introduction of a clustering of software component templates. This cluster could contain various software components, that form a certain functionality, together with their timing chains, that refer to this functionality. This approach would need to change the existing SWCT structure.

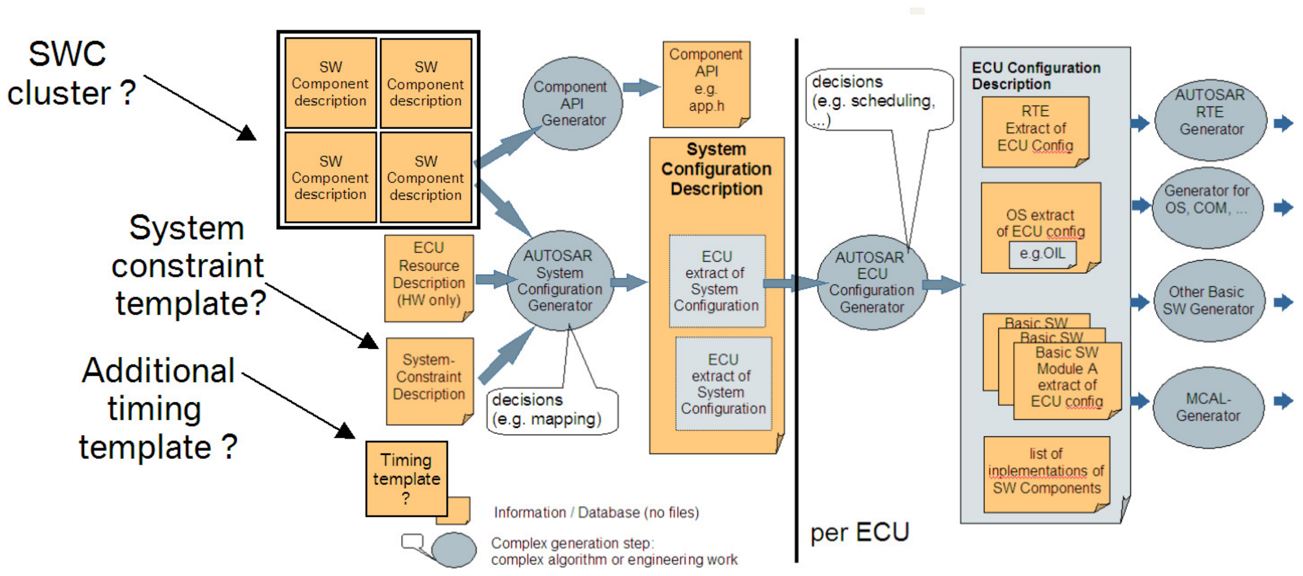
SWC cluster ?
System constraint template?
Additional timing template ?
SW Component description
SW Component description
SW Component description
SW Component description
ECU Resource Description (HW only)
System-Constraint Description
Timing template ?
Component API Generator
Component API e.g. app.h
AUTOSAR System Configuration Generator
decisions (e.g. mapping)
System Configuration Description
ECU extract of System Configuration
ECU extract of System Configuration
decisions (e.g. scheduling, ...)
AUTOSAR ECU Configuration Generator
per ECU
ECU Configuration Description
RTE Extract of ECU Config
OS extract of ECU config
e.g.OIL
Basic SW
Basic SW
Basic SW Module A extract of ECU config
list of inplementations of SW Components
AUTOSAR RTE Generator
Generator for OS, COM, ...
Other Basic SW Generator
MCAL-Generator
Information / Database (no files)
Complex generation step: complex algorithm or engineering work

Fig 5 – Location of timing chains within the model (after [Scha05])

4.2 Complexity of finding a suitable System Design

Timing chains are a prerequisite for a realization of the system generator concept. As tempting and desirable as the vision of a system generator might sound, implementing such a generator is a more than challenging task. Today's automotive systems already consist of up to 70 ECUs and hundreds of functions which depend on each other. Obviously finding a valid system design with respect to timing requirements and logic, CPU, and Bus load is a rather complex task. Due to the large number of functions and ECUs a vast number of potential solutions exist. The high dependency between the functions make the validation of a solution a very complex task itself. For instance moving only one function from one ECU to another ECU can lead to changes on the bus schedule which might have an impact on the schedules of other ECUs. Therefore validating only the schedule of the ECUs directly involved in the function might not be enough. Both the high number of potential solutions and the rather complex validation of possible solutions make the task of finding a feasible system design a highly complex one. This complexity is one of the big challenges which has to be faced when approaching the task of implementing a system generator. If the complexity is not reduced there is a potential danger of not receiving a result within a reasonable time on today's computer systems.

4.3 Abstraction and Assumptions

A common approach towards managing complexity is to abstract from too much detail. A higher level of abstraction means taking less information into account which makes the calculation and validation less complex, thus less time consuming. Abstracting from details means that assumptions have to be made. Due to the highly safety relevant nature of most automotive functions assumptions have to be made in a pessimistic way. A good example for such a pessimistic assumption is to give a task a worst case execution time and assume that every time the task is scheduled it will occupy the CPU for that time although in reality the execution time might vary. This way validating a schedule on an ECU becomes easier as only one execution time has to be taken into account for the task. Otherwise if the model would allow to assign more execution time to a task depending on some kind of condition different cases would have to be validated. This example shows how abstracting from details reduces the complexity of validating a system design. The drawback of making pessimistic assumptions in a model is that no valid system design might be found by the generator although in the real world there is a solution. Especially within the automotive domain cost minimization is an important boundary condition in system design. The resources have to be exploited as far as safety and security requirements allow for it. Making assumptions in a pessimistic way will lead to unused resources which increases the cost of the system. In order to come to a better system design which is closer to the best possible design more detailed information is needed.

The two stated goals of making the complexity transparent and maximizing the resource exploitation make it very important to choose the granularity of abstraction for the model carefully. A too high level of abstraction will lead to results which waste too many resources making the system too expensive and unusable for real world systems. A too detailed model could make the task of finding a system design too complex for today's computer systems.

5 Approach

The key problems stated in the previous chapter are the complexity of the considered algorithmic problems, the level of abstraction of the model and the completeness of the used information.

5.1 Coping with Computational Complexity

The algorithmic problems like calculating an optimal ECU schedule, determining an optimal bus schedule, finding an ideal distribution of software components over the network are very complex and indeed most of them are NP-complete problems. NP-complete problems are a class of problems that are all "as hard as the simplest problem in that class. At present, all known algorithms that compute an exact solution for an NP-complete problem require an exponential amount of time.

For example for the calculation of an optimal static ECU schedule for a given number of tasks, the calculation time grows exponentially with the number of tasks, because finding the solution in general requires to try all possible combinations of distributing the tasks over the timeline.

The good news is, that finding the optimal ECU schedule is not by all means required. Today the problem of finding an ECU schedule is solved by a brain controlled method, the system integrator of the ECU, and there is no statement that this schedule is optimal. It is sufficient to find any ECU schedule that is correct (meaning there are no overlaps of the tasks) and fulfills all defined timing requirements. So the hope here is, that there are algorithms that are able to find a solution for a working ECU schedule in a reasonable amount of time. This solution needs not be the overall optimum but it will do the job in a sense that all requirements, including the timing requirements, are met.

In this context some promising approximations, heuristic algorithms like genetic algorithms or greedy algorithms and probabilistic approaches are examined and evaluated if they can be used to find solutions in acceptable time. Heuristic algorithms do not guarantee to find a solution, but in an average case find a solution in much shorter time. Nevertheless, not finding a solution is not a proof that there isn't one.

5.2 The right level of abstraction

The second point is the appropriate level of abstraction. The trade-off between being too general and being too concise is the central point here. If one is too concise, it can be doubted if all the information that can be modeled will in fact be modeled by a user. One can imaging that if modeling the software system reaches a too fine granularity it's no longer a model but more a different way of doing the implementation. The target user group of the envisioned methodology will not use it if this means implementing the software twice.

On the other hand, if the level of abstraction is too abstract, the calculation of a solution will probably be too pessimistic. It could be possible that an algorithm cannot find a solution although a system architect finds one.

The difference between the system architect and the algorithm is, that the system architect can rely on some implicit knowledge about the system that is not reflected in the model. Exploiting this knowledge allows him to create a working schedule, that wouldn't be found by an algorithm, that relies on the modeled information, that is in that sense incomplete.

5.3 Example: Tasks and runnables

Consider a system that has 6 runnables, each with a worst-case-execution time of 100 ms. Runnable 1 and runnable 2 are located in task A, runnable 3 and 4 are in task B and runnable 4 and 5 are in task C. This gives all three tasks a worst-case-execution-time (WCET) of 200ms each.

Let's say the only slot left on the ECU is 500 ms, so an algorithm will easily find out, that there is no valid schedule for this situation.

The assumption in this case is, that the WCET of a task equals the sum of the WCETs of its contained runnables. What is modeled here is only the runnables and their WCETs and the containment association between runnable and task.

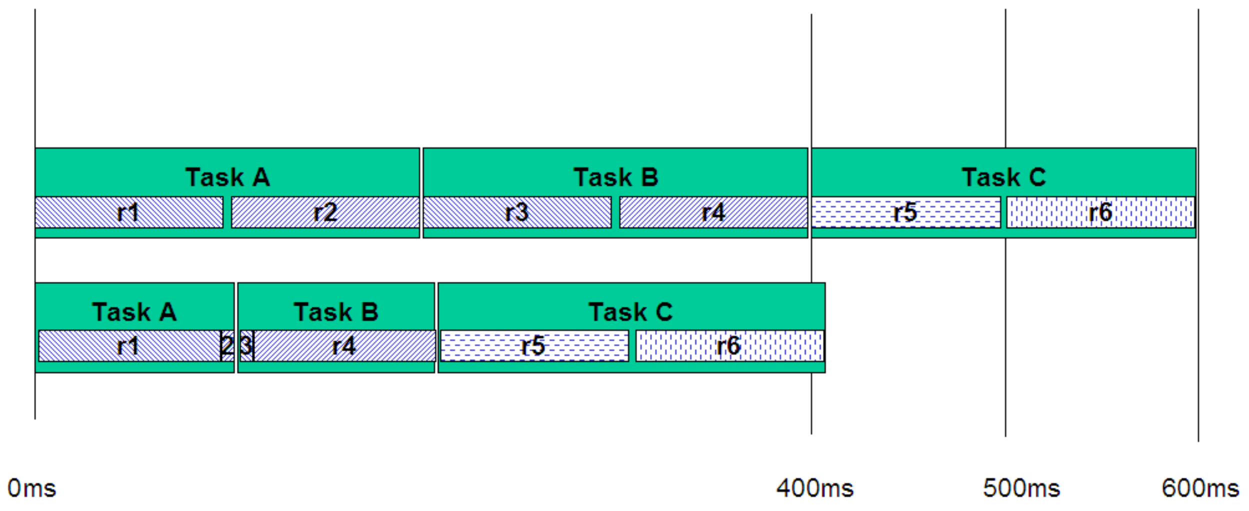

Fig 6 – Intelligent runnable to task mapping

A clever system architect states, that it is still possible to do a scheduling that works. The system architect has insight into the functions and for example knows that runnable 1 and 3 belong to the function "park distance control" (which is only active at speeds lower that 10 km/h) and runnables 2 and 4 belong to a hypothetical function that is only active at speeds higher than 100 km/h. If one of the functions is inactive, the WCET of the runnable drops from 100 ms to let's say 2 ms. If he uses this knowledge he can calculate the WCET of the tasks more accurate, that is less pessimistic. In this example the WCET of task A would no longer be 200 ms, but would be 100 ms + 2 ms = 102 ms. The same is true for task B. There is no more insider information about runnable 5 and 6 so the WCET of task C remains 200 ms.

So using the knowledge of the function the sum of all task WCETs would go down from 600 ms to 404 ms and the tasks could be distributed into the 500 ms slot.

This is a good example how more precise information can reduce pessimism. To make this knowledge accessible to an algorithm one has to extend the meta-model to be capable of capturing dependencies between runnables (like mutual exclusion) or maybe define certain vehicle states that divide functions mutual excluded sets. The algorithm could then exploit the same knowledge that the system architect uses.

5.4 Finding a compromise

Our approach is to start modeling the system and its requirements on a rather abstract level like we described it in chapter 3.1 and take this as a basis to iteratively refine the model as needed.

Starting at a high level of abstraction gives a relatively high probability that the information needed is available and that the model will not become too complicated. The disadvantage is, that the higher the level of abstraction is, the more pessimistic the calculation will be.

But this does not necessarily mean, that the algorithm is wrong. A good algorithm would find better solutions with better input data. In the example above the algorithm would find a solution if there was enough time available (more than 600ms). Refining the meta-model with the cleverer calculation of the WCET would immediately lead to better results.

It can be doubted if all this implicit knowledge about a system can be formalized in a way that it can be put in a model. The challenge is to put enough information in the model to be useful and less enough information to still be able to capture and handle it.

To estimate if the selected level of abstraction is good, we compare the calculated schedulings with schedules from already implemented ECUs. The claim would be, that if a human being with the given information can find a solution, the algorithm should also be able to find one.

Our approach foresees to compare cases, where in real-world examples an algorithm does not find a solution at all and find out, which implicit information about the system the system architect has used to find a solution. This knowledge must somehow be captured in the model.

Even if this information is not complete, it brings us closer to the vision of a system generator. Maybe the full automatic system generator remains a dream of the future, nevertheless a semi-automatic solution would still improve the situation. One could think about a tool, where the system architect is aided by the tools, but still can make decisions on key aspects in his system on his own.

6 Conclusion

Due to the growing complexity of automotive systems providing automated support for designing them will become inevitable in future. The model driven approach specified in the AUTOSAR standard is a good basis for the realization of such an automated support. The current version of the meta model needs to be extended as some required features, especially timing information, are still missing. The timing chain approach is a first step towards adding those information to the AUTOSAR model. Further information, like the mentioned mode-awareness, have to be added to the model later to provide the system generator with information that a human system designer today has because of his experience. Choosing the right granularity of the information by which the meta model is to be extended is crucial. A too precise model might lead to too complex and time consuming algorithms making the automated support unusable within a real system design process. Choosing a to high level of abstraction can lead to a too pessimistic model making the automatic support unusable for the real world. Thus the success of developing an automated system design support usable for real world system design processes is highly dependent on closely coupling the development to real world scenarios. Only this way early feedback can be gained about the quality of the approach's results and its usability within system design processes.

7 References

[AR07] The AUTOSAR homepage http://www.autosar.org

[AR07b] The AUTOSAR homepage http://www.autosar.org, Technical Overview/AUTOSAR Methodology

[Rich06] Richter, Kai. The AUTOSAR Timing Model - Status and Challanges. ARTIST2 Workshop Innsbruck. March 2006

[Scha05] Scharnhorst, Thomas et al.: AUTOSAR – Challenges and Achievements 2005. Electronic Systems for Vehicles 2005, VDI Congress, Baden-Baden, 2005

[Sche07] Scheickl, Oliver: "Spezifikation und Implementierung von Metriken zur Analyse des Echtzeitverhaltens von verteilten automotive Systemen", Masterthesis, München 2007