Software Plattform Embedded Systems 2020

GEFÖRDERT VOM

Deliverable D3.3.A – ZP-Subtask3.3.3 View-specific Analyses

Matthias Büker, OFFIS
Philipp Reinkemeier, OFFIS
Eike Thaden, OFFIS
Raphael Weber, OFFIS

Tuesday 5th April, 2011

Projektbezeichnung	SPES2020	
Verantwortlich	Philipp Reinkemeier	
QS-Verantwortlich	Jörg Holtmann (UPB)	
Erstellt am	10.11.2009	
Zuletzt geändert	05.04.2011	
Freigabestatus		Vertraulich
		Projektöffentlich
	✓	öffentlich
Bearbeitungszustand		in Bearbeitung
		vorgelegt
	✓	fertig gestellt

Weitere Produktinformationen

Erzeugung	Matthias Büker (MB), Philipp Reinkemeier (PR), Raphael Weber (RW), Eike Thaden (ET)
Mitwirkend	Ingo Stierand

Änderungsverzeichnis

Änderung			Beschreibung der Änderung	Autor	Zustand
Nr.	Datum	Version			
1	10.11.2009	V0.1	Initiale Version	PR, RW, ET	in Arbeit
2	27.01.2010	V0.9	Vorgelegt	PR, RW, ET	Vorgelegt
3	05.04.2011	V1.0	Überarbeitete Version	ET, MB	abgeschlossen

1 View-specific Analyses

The SPES design process utilises the common idea of model-based development. The aim is to reduce complexity of the design process and to iteratively refine a system solely based on models towards a final implementation. This allows for analysing functional and non-functional aspects of the system on various levels of abstraction. Moreover, the model-based approach enables requirement refinement, and to validate whether satisfaction of requirements defined for a given abstraction level implies satisfaction of the requirements defined for a more abstract level. A description of the system of abstraction layers as defined in SPES is provided in [10].

In the SPES context, analysing a system model means to verify the system against the requirements specified for the system. Usually requirements refer to one or more viewpoints on the system. For instance, latency-requirements refer to the real-time point of view and might also be important for safety analysis. There are two approaches to deal with these requirements and viewpoints. Firstly, for each viewpoint on the system, a different model could be created, along with requirements based on a formalism dedicated to and optimised for the particular viewpoint. That means for example one model addresses the structure of the system, another the behaviour, etc. Despite the benefit of a strong separation of concerns, the main drawback is that it is difficult to relate the different models and requirements to each other, which is important in order to assess dependencies between viewpoints. The second alternative is to integrate multiple viewpoints on the system into a single model, thus allowing an easier way to analyse these inter-viewpoint dependencies. For this reason the latter approach was chosen for the SPES2020 project. Doing so, a general notation is needed to express requirements for different viewpoints in a uniform way.

This results in *integrated models* as shown in Figure 1.1. The semantics of these models is required to capture the logical and physical structure of the system (the rectangles in the figure), its dynamics, i. e. the behaviour, and properties of the components of the system.

Each component of the model is enriched by a set of contracts that describe the requirements on the component. Thereby, for different viewpoints (real-time, safety, etc.) separate contracts are specified. The single-model approach taken in SPES allows analysis techniques to either concentrate on a specific viewpoint or to analyse inter-viewpoint dependencies. In order to apply an analysis technique, an instance of the meta–model will be transformed into a representation suitable for the respective analysis. For this transformation only the parts relevant for the analysis of the system model need to be extracted. For example, an analysis of the geometry of the system does not require properties regarding the real-time perspective.

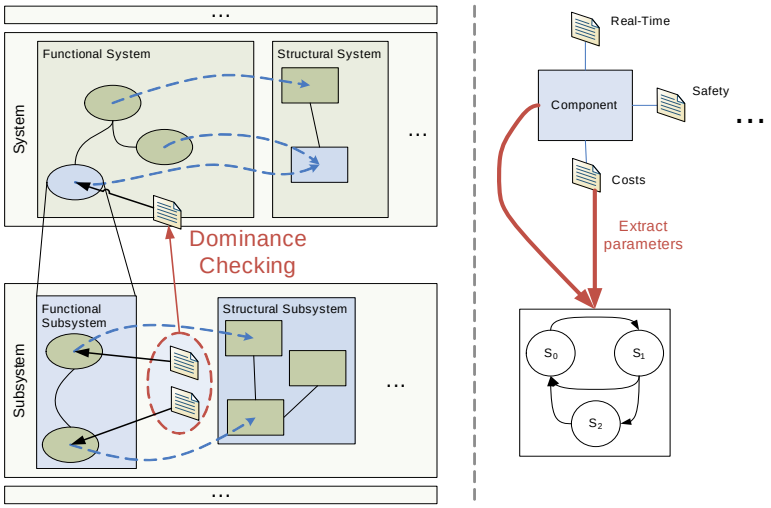

Figure 1.1: System of vertical abstraction layers and requirements

1.1 Real-Time Analysis

Real-time analysis and verification is an important task to ensure correctness of distributed embedded real-time systems. The phrase real-time implies that the correctness of the functions of the system not only depends on the results, but also on the time instant at which the results are available. Requirements regarding real-time belong to the group of non-functional requirements, which describe quality aspects of functions [5].

Real-time analysis is used to verify, that the real-time requirements imposed on the system are met in all workload scenarios. Exemplary requirements are latency/deadline requirements. Furthermore, originating from control theory, there are often requirements on how frequently a function must be executed in order to keep the controlling system synchronous with the controlled environment. During the design process of a real-time system, these initial application requirements are broken down and refined according to the function decomposition and refinement.

Obviously for such an analysis, information is needed about the execution times of the functions of the systems as well as dependency relationships between functions. The latter induces an execution order between functions. Apart from properties of the functions of the system, a model of the environment at an appropriate abstraction level is also needed. This model represents the stimuli of the real-time system and determines bounds for the expected workload.

At some steps in the design process, functions are mapped to architecture elements that realise the functions. These elements can be either dedicated hardware components or software-programmable units. As these elements execute the functions, their properties also have an impact on the satisfiability of real-time requirements. Most notably the WCET (worst case execution time) of a function strongly depends on the processing element that executes it. For example, software processors might be oper-

ated at different frequencies. In order to save costs, usually resource sharing is applied, meaning multiple functions are mapped to a single architecture element. Evidently a single resource can only execute one function at a time. This requires multi-cores not to be considered as a single resource, but instead each core is considered a resource. The sharing of a resource by multiple functions must be organised/scheduled. Additionally there might exist constraints with regard to scheduling of the resources, such that a function requires atomic access to it. These aspects of resource sharing need to be considered in real-time analysis, as they influence the response times of functions, i.e. the time a result is available.

1.1.1 Analytical vs. computational models

In order to formally verify the behaviour of a system against a given set of real-time requirements, a formal representation of the behaviour of the system including its aforementioned non-functional properties is needed.

There exist two fundamental approaches for timing analysis. Either the system can be described by a computational model, which can be model-checked also against complex requirements or analytical models are utilised to determine response times and verify deadlines and synchronisation requirements. Computational models provide a very high expressiveness which unfortunately requires very complex timing analysis. Thus computational models are usable only for small systems. In contrary analysis methods based on analytical models scale very well with the size of the system but the expressiveness of these models is limited. Thus a combination of these two approaches is promising. In [6] it has been shown, that the results of scheduling analysis using task networks based on *event streams* can be translated into a semantically equivalent timed automata representation. For this purpose cyclic timed automata (CTA) are used, that consist of a set of timed automata templates, instantiated and parametrised according to the given task network. The underlying architecture, executing the tasks and providing communication channels, is implicitly considered in the CTAs, as the task network, which serves as input for the scheduling analysis, already contains the relevant properties.

After the transformation into CTAs, model-checking can be applied to prove the adherence of the system to complex requirements, specified in well-defined formalisms like e.g. Live Sequence Charts (LSC) [4].

1.1.2 Task Networks with Functional Extensions

In the following a model for so-called *function networks* [3] is briefly introduced that extends traditional task networks by functional properties. A function network is a bipartite graph consisting of tasks or function nodes, and data nodes connected by channels. The temporal behaviour of a function network is defined in terms of event streams. Event streams characterise the time instants of event occurrences at defined "observation points" like task activations. Events are emitted by event sources and

flow through the network until they disappear at an event sink, which may be a task or a data node. The path an event traverses within a function network is referred to as event flow. Event flows may be joined together or split at certain nodes so that not only single events but combinations of them may be observed. In order to describe which combination of events may appear at an observation point (denoted as port) we use event sets.

The modelling of task dependencies on multiple incoming events causes the corresponding event streams to be joined. Here we differentiate between different kinds of activations as depicted in Figure 1.2. If each occurrence of an event causes an ac-

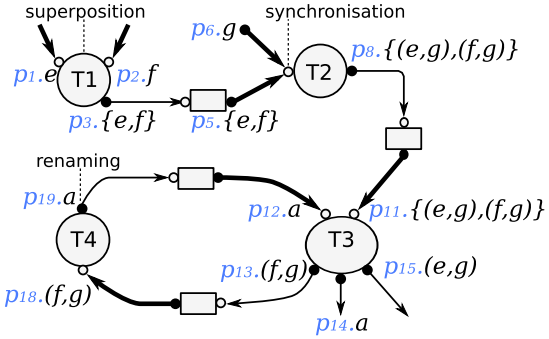

Figure 1.2: Events and event flows

tivation of a task, we call it an *OR* activation. The corresponding event streams are superposed implying the appendant event set. If multiple incoming events are necessary to activate a function node we have an *AND* activation and the corresponding event streams are synchronised. The associated events are combined to a product event set. Furthermore, a *renaming* of events is allowed, which is for example useful to model loops as depicted in Fig. 1.2 where (f,g) at port p_{18} is renamed to a.

Function networks extend the expressiveness of classical task networks by functional elements, and a differentiation of data and control flow by using data nodes and specific channels. The functional extension is achieved by combining the concept of event sets and an abstract representation of the functional behaviour of tasks as a state-transition-system where one transition refers to one activation of the task. The decision which transition is triggered at a certain activation point depends on the state and the incoming event set. We denote such extended tasks *function nodes* (centre of Figure 1.3). In this example, the transition system depicted at the right of the function node consists of one state s_0 and two transitions. The first one, for example, is triggered if the event e occurs at port p_1 and sends an event h at output port p_3 after a delay of δ_1 time units.

Another important extension is the introduction of *data nodes* and directed channels. Control flow is expressed by *activation channels* meaning a dependency of function nodes on the writing events of a data node. Data flow is represented as *read channel* where a function node reads from a data node at its activation. Write channels are used

to write data into a data node and may model both data and control flow. All channel types imply an access on the corresponding data node either by reading the current data or writing new data. Data nodes may be of different type as shown in Figure 1.3: *Source* nodes serve as event sources without any incoming write channels, *Signal* nodes forward events at their input ports immediately to their output ports without storing any data permanently, and *FIFO* and *Shared* data nodes are persistent data storages implementing FIFO-buffers and shared variables, respectively. The connection of nodes and channels is realized by *ports* (small circles) that represent the observation points in the system.

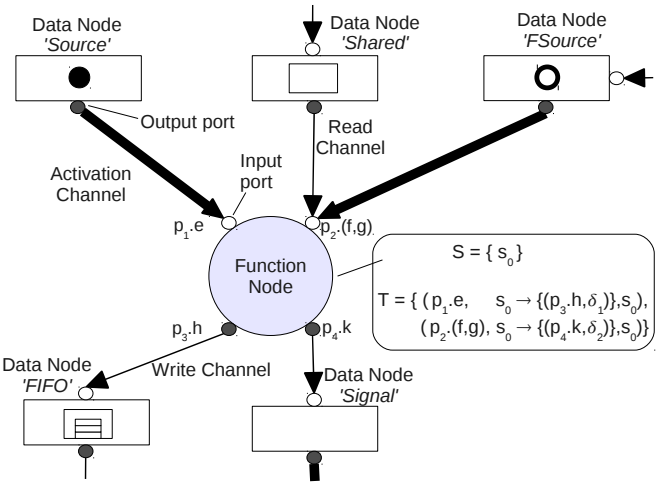

Figure 1.3: Function Network Elements

To sum up, in contrast to traditional task networks, function networks allow the modelling of e.g. mode- or data-dependent execution times as well as internal states for tasks. Depending on this states, interaction behaviour (communication) of tasks can differ. These kind of functional information is usually neglected in current scheduling analysis techniques, leading to great over-approximations, which may cause feasible systems to be rejected. There is certainly a trade-off between the more accurate analysis results, which can be achieved by considering the additional properties, and the complexity of the analysis. But the advantage of the formalism of function networks is, that the desired accuracy can be selected. That means, by omitting the specification of internal task states, more efficient analytical methods can still be applied for scheduling analysis, whereas otherwise model-checking techniques can be used.

1.1.3 Analysis approach

With a mapping of the elements of the reference architecture to function network elements and the extraction of platform properties to a platform model, the entire system is formally represented. A dedicated real-time analysis tool, which will be developed during the course of the SPES project, can now calculate the timing information the

designer is interested in. Besides the standard properties like response times, this also includes more complex requirement, like checking maximum buffer sizes and synchronisation constraints.

Towards a thorough analysis, the real-time *requirements* must be considered. In [14] *real-time contract patterns* are described, that allow the specification of real-time requirements using common event patterns. These patterns were defined in the SPEEDS project. For SPES similar patterns will be provided. For most of the patterns, checking a system against a requirement can be done directly on the basis of the pattern. Where this is not applicable, the patterns are translated to equivalent timed automata (observer). In order to check these requirements during the analysis, the observer automata constructed from the patterns are included within the model, and satisfiability is checked.

For this computational model-checking to be scalable, it is currently investigated how the state space of the system can be appropriately reduced according to the actual analysis task. The approach is to take a complete specification of the inputs and to constructively build the states of a resource and its deployed tasks, which can be reached based on the inputs. Additionally, certain states might be combined, thereby realising a safe over-approximation. This is also very common in the domain of analytical scheduling analysis. These abstraction mechanisms would allow the user to decide about the trade-off between the desired granularity and the speed of the analysis.

It is obvious that the system model under analysis must contain the properties serving as inputs for the analysis. For instance, if no information about execution times is available, verification of latency requirements makes no sense. However, this does not mean that real-time analysis would be restricted to specific abstraction levels of the reference architecture. Even, when exact execution times and similar properties can not be given, safe over-approximations can be used at higher abstraction levels.

1.1.3.1 Modelling the Architecture

Beside the formalism for modelling the functions of the system, also the underlying platform where the functions are mapped to needs to be captured. It will be evaluated at which granularity system architectures can be represented for the analysis. As for the state representation of the function network, both precision and speed of the analysis depends on the abstraction level of the system architecture representation. Traditional real-time scheduling analysis relies on simple assumptions about the execution times of tasks. On the other hand, representation of the internal states of the underlying processor greatly increases precision of such execution times, as recent approaches show [7]. However, the characteristics of the platform that are inevitable for real-time analysis, must be reflected. This includes e.g. the topology information, which means the number of ECUs and their connecting buses, as well as properties like the types of system buses and their bandwidth.

In Figure 1.4 the previously described analysis approach for analysing task networks is sketched. A simplified AUTOSAR model of a brake controller has been

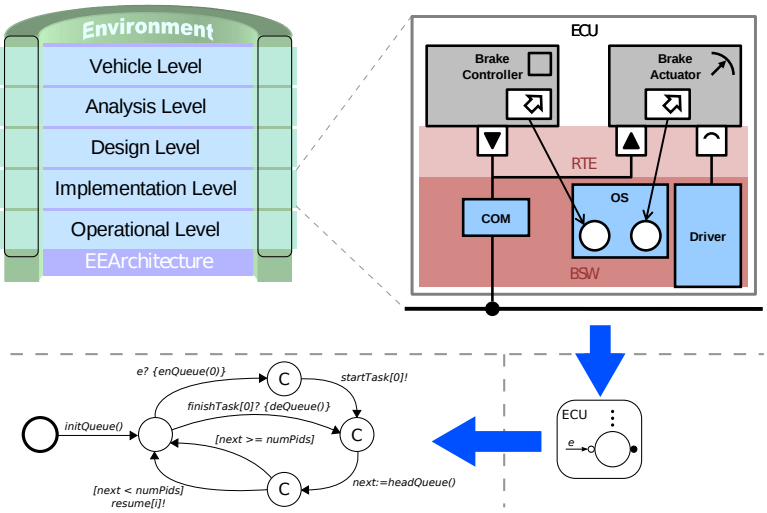

Figure 1.4: Analysis approach

extracted from an instance of the reference architecture. ”AUTOSAR (AUTomotive Open System ARchitecture) is an open and standardized automotive software architecture, jointly developed by automobile manufacturers, suppliers and tool developers” [1].

An application software component contains the application logic of the brake controller (ECU: Component ”Brake Controller”), and a sensor-actuator software component controls the brake actuator (ECU: Component ”Brake Actuator”). The behavior of both components can be expressed in various ways, e.g. a state charts or any programming language (not shown in the Figure). Based on this model a representation in the function network formalism is created, which incorporates architectural properties, e.g. the fact that software tasks are running on processors, potentially scheduled together with other software tasks. For this representation semantically equivalent timed automata can be constructed, which are suited for model-checking with e.g. UPPAAL.

1.1.4 Contracts for real-time viewpoint

The previously discussed approach for real-time analysis is suitable for analysing the timeliness of the system with respect to some real-time requirements. On design entry however not all requirements are known. Instead, the designer would probably start by specifying requirements for the application that are successively refined and decomposed during the design process, along with the system components (see Figure 1.5).

For example, assume a brake controller should be created. During the design process, the controller is decomposed into further functionalities. An initial contract requires the latency between receiving a brake request and actuating the brake to be not more than 10 msec. Along with the decomposition of the brake function, this

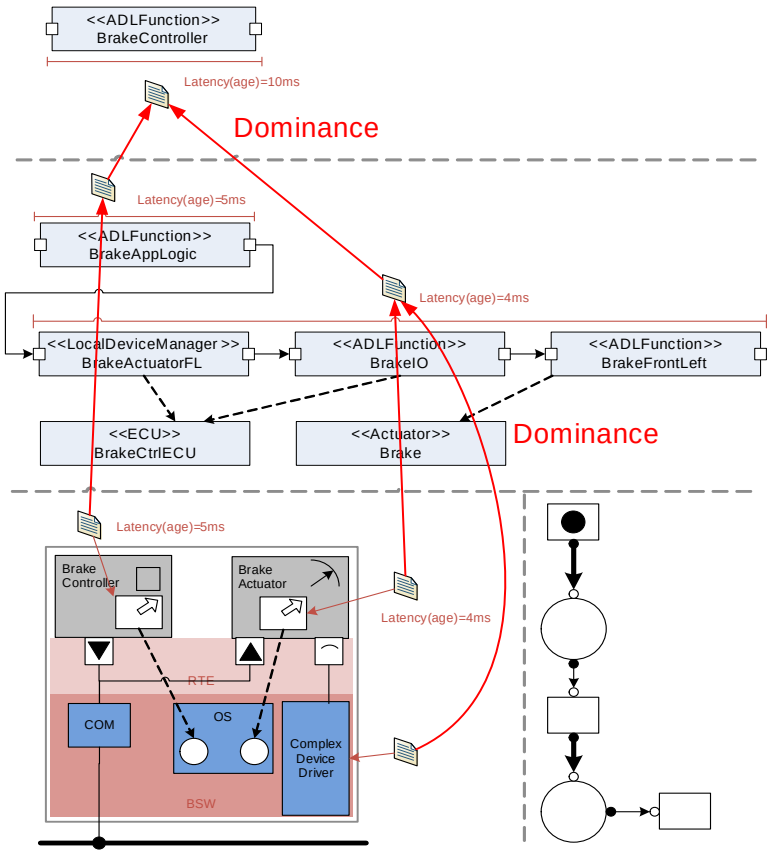

Figure 1.5: Real-time requirements tracing and analysis

latency-requirement is broken down into several requirements. A way to fomalize requirements is to use contract-based design. As in [8] contracts consist of strong and/or weak assumptions and a guarantee (promise). At a certain level of abstraction (here AUTOSAR) the designer may want to analyse the adherence of his design to real-time requirements, which is checking the *satisfaction* of real-time contracts. More precisely, it must be verified that the implementation M guarantees the promise P of the contract C, whenever the assumption A holds ($M \cap A \subseteq P$). While the analysis can verify the requirements at this level, the relation to the initial application requirements must also be verified. In Figure 1.5, this *satisfaction relation* is shown. Proving the dominance relation between contracts reasoning about latency is quite easy (addition and comparison of the latency intervals).

1.2 Costs Analysis

There are several types of costs in embedded systems design. Some occur in the planning phase of the system, other while assembling the system, while using it or while disassembling and scrapping/recycling parts of it. In short, costs are occurring dur-

ing the whole lifetime of an embedded systems. Capturing all those costs is a way too large task for the 3-years project SPES2020. Therefore we plan to concentrate on commonalities of these costs and support the use of arbitrary cost models by providing generic cost specification methods.

Analysis tools to work with these generic specification are not necessarily available. However they could be integrated into the open system architecture developed in ZP-AP-3. The information required for them to work properly can already be expressed in the model with the help of the cost viewpoint described in this section and that about costs viewpoint.

While it is hard to provide tools for analysis of arbitrary cost functions, it is possible to provide analysis tools for some simple special cases. The next subsections describe two tools which can be used to evaluate different cost related properties of a model given that the neccessary cost information is available. Both tools will be developed/adapted to be usable on models in the meta-model format used throughout ZP-AP-3.

1.2.1 Component Costs

In the costs viewpoint it is possible to attach cost requirements to each component in a model. The simple cost requirement used for this analysis tool is an upper limit which may not be exceeded by this component in any case. For a simple component without hierarchy this requirement can easily be verified as soon as its actual cost information is known. To make things more flexible costs are categorized using freely configurable cost categories. A category consists of a name which is referred to by cost specifications and cost requirements.

1.2.1.1 Cost Specifications

For each component it is possible to directly specify its costs for a given category. As example the costs for one ECU in a model are determined by its type. While an ARM7 processor type might cost around 6 EUR, a PowerPC based one could cost around 45 EUR (prices are fictitious and only for illustration). The costs might also be undetermined at the moment if the best ECU type has still to be determined in an analysis step.

Whenever a component has subcomponents the designer has the choice: Either he/she writes a fixed cost value to the parent component directly, or relates the costs of the parent component to those of the subcomponents. Each specification consist of a cost expression. Examples:

A cost expression which relates the cost for one component to the costs of its subcomponents could be written like this:

```
A.costs[category=acquisition]:=
  sum{B.costs[category=acquisition] | B subcomponent of A}
```

Of course the value for one cost category may depend on values of other categories. Something like this example is possible too:

```
A.costs[category=overall]:=
  A.costs[category=aquisition] +
  A.costs[category=maintance]
```

A formal language for cost expressions will be defined later during the implementation phase of ZP-AP-3.

1.2.1.2 Cost Requirements

A cost requirement consists of a cost expression for a specific cost category and in addition is required to contain a relation operator ($<, \leq, =, \geq, >$). This requirement can be put on every component in the system. Whenever the costs for this category are already known for a component, its cost requirement can be validated immediately. Usually this is not the case. It can be expected that cost requirements are mainly found on higher (more abstract) levels of the component hierarchy.

1.2.1.3 Analysis Tool

The cost analysis prototype will be able to analyze a subclass of the possible cost expressions and find values for them. Those values are then written to the result model of the cost analysis step. After the values are known the tool can check cost constraints against them. The first prototype of the cost analysis will support basic operations (sums, basic selectors on model elements, etc.). Depending on the needs of the SPES industry partners this tool can be extended later to support more sophisticated expressions.

1.2.2 Cost Optimization

Model based costs calculation itself is an interesting feature of the ZP-AP-3 tool chain. But certainly costs cannot be seen completely isolated in the model. Reducing the costs, e.g. by using a cheaper ECU type for an ECU usually leads to reduced performance for the tasks deployed on that ECU. This might have implications on other viewpoints, in this case obviously for the realtime viewpoint. Exchanging the ECU type forces the designer to re-evaluate the realtime properties of the system (if this step was already finished). Other viewpoints are affected, too. The cheaper ECU might have a higher power consumption, a larger weight, bigger physical dimensions, weaker safety capabilities, etc. The example shows that changing one system parameter possibly affects many system properties directly or indirectly. Optimizing for costs is therefore a hard task where tool support would be useful.

In ZP-AP-3 a prototype analysis tool combining the two dimensions costs and realtime will be integrated into the open system platform. The goal of this tool is to support

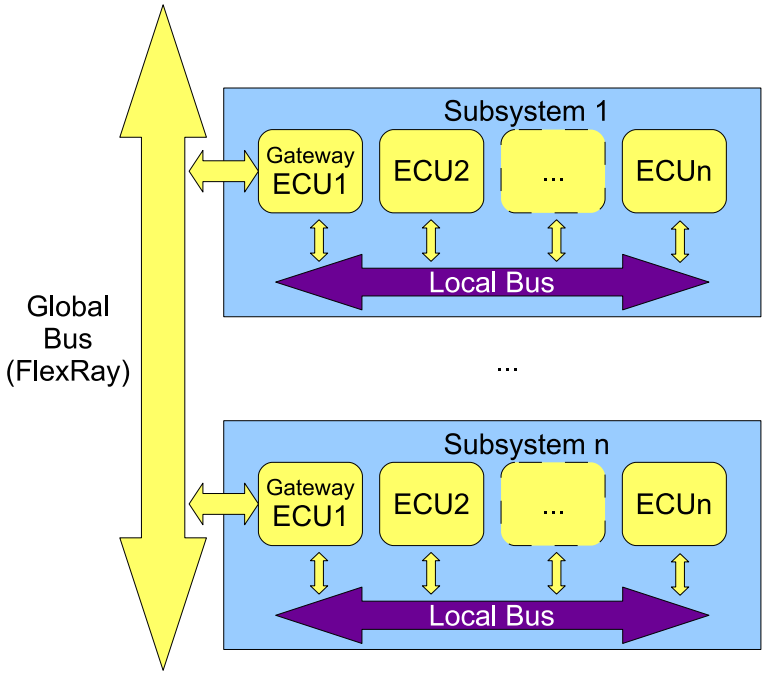

Figure 1.6: Cost Optimization: Global System

the user during extending an existing subsystem with additional ECUs causing minimal costs while still respecting all deadlines of tasks and signals on that subsystem. Currently the cost optimization tools expects a subsystem as input. A subsystem consists of an architecture of ECUs and buses and their interconnection on the one hand and a tasknetwork consisting of tasks and signals send/received by them on the other hand. Tasks are required to be periodically activated (by a timer). For each task, its worst case execution time (WCET) per ECU type, its period and its deadline which has to be smaller or equal to the period has to be known. For each signal, its transmission time per bus type and a deadline for transmission are required. ECUs have a type attribute which is used to choose the right WCET when deploying a task to them. Each bus has a type attribute, too, which is used to identify the corresponding signal transmission time when deploying a signal to this bus. A partial deployment is required for the subsystem. That means that most tasks and signals are already deployed to an ECU or bus. An example of an architecture with several subsystems is given in figure 1.6

The analysis is then capable of deploying the remaining tasks and signals to the existing ECUs and buses without invalidating any deadline requirements if there is free capacity left over. Otherwise it can decide to add a new ECU to the subsystem. Of course this is not for free but costs money depending on the ECU type of the new ECU. The goal of the analysis is to minimize the sum of all costs arising from inserting new ECUs.

1.3 Safety Analysis

Safety analyses are concerned with the examination of safety–critical properties along with the question "How does the system react in case of a failure?". In order to determine safety–critical properties, the designer has to specify all characteristics of a system e. g. via contracts, requirements, or constraints. As described in [14] there are different viewpoints, one of which is the safety viewpoint that will be described in this section.

The first step in a safety assessment is to identify and describe the item (a system, multiple systems, or a function under safety analysis), and to develop an adequate understanding of it. This essential step is necessary in order to perform subsequent phases of the safety assessment which are based on the item definition and the safety concept derived from it. To achieve a satisfactory understanding of the item, information about its functionality, interfaces, or environmental conditions is required.

The second step in a safety assessment process is to analyze the items interactions and how they impact the system's behavior. For these analyses the behavioral model of the functionalities shall be defined. This will be possible with the SPES meta–model. The SPES meta–model will also allow to define the safety goals (overall safety requirements) at the system level. With this information a number of different analysis techniques can be utilized.

1.3.1 Hazard Analysis

The hazard analysis is based on the item definition. A hazard is a potential source of harm, due to a possible item malfunction in conjunction with a particular scenario's condition. These malfunctions include all possible functional anomalies derivable from each foreseeable source, either internal (system faults) or external (e. g. foreseeable misuse). Scenario conditions can be a complete scenario description including all variables and/or states that characterize the functionalities or affect them, e. g. vehicle operation modes and environmental conditions. A malfunction can therefore cause several hazards with different consequences.

The inputs of the hazard analysis and the risk assessment are system boundaries, a functional description of the system, scenarios (operation modes and environmental conditions), and potential failure modules. The outputs are safety goals and Automotive Safety Integrity Levels (ASILs, for the automotive domain) or Design Assurance Levels (DALs for the avionic domain).

The SPES meta–model provides ways to describe environmental conditions along with functional system description. Thus, it supports the determination of the ASILs as defined in the ISO/DIS 26262 [**?**] or DALs as defined in DO178B [**?**].

The Automotive Safety Integrity Levels are based on the assessment of the exposure (frequency), the controllability and the severity of a fault. Four Automotive Safety Integrity Levels are defined, starting from A (lowest level) to D (highest level).

Design Assurance Levels in airborne vehicles are similar, except that here A is the highest level and E the lowest. The following levels are defined: Level A (Catastrophic), Level B (Hazardous), Level C (Major), Level D (Minor), Level E (No effect).

According to these documents ASILs/DALs can also be derived and decomposed in multiple lower ASILs/DALs. The SPES meta–model will offer means to associate each architectural element to the related safety requirements and corresponding ASIL/-DAL. Note that each decomposition requires the definition of new additional requirements to be assigned to the appropriate decomposed element.

In further analyses it can then be determined for example if there is a common cause of failure (CCF) for multiple components. In particular safety analyses should support:

- finding single points of failure (SPFs, see [12]) and common causes of failure (CCF, see [2])
- verification of safety concepts and requirements
- identifying conditions that could result in a violation of safety goals or requirements

In order to achieve these goals the SPES meta–model supports modeling of the safety goal and the ASILs/DALs. Furthermore, it will support partial derivation of analysis information from the specified model, in order to conduct e. g. a failure modes and effects analysis (FMEA) [11] or a fault tree analysis (FTA) [13]. We will now shortly describe the FTA and FMEA.

1.3.2 Fault Tree Analysis (FTA)

A major aspect of safety analyses is to find the reasons that cause a hazard. To achieve this, the Fault Tree Analysis (FTA) can be utilized. The FTA is a top–down method that systematically breaks down hazards to their causes. The result is then visualized in a tree structure. Thus, the FTA rather helps systematically analyzing hazards than to detect them.

The first step in the FTA is to define the system bounds. This includes the definition of the hazard, the events the analysis has to take into account, the physical system bounds, and the initial system state. For example, there is a huge difference between a car in city traffic and a car on the autobahn being analyzed. Once the system bounds are defined, the occurrence of a hazard (top–event in an FTA) is iteratively decomposed into its causes. To visualize this process a standardized graphical notation (IEC 1025 Standard) is used. Figure 1.7 displays an example fault tree.

An FTA combines lower level causes for the top–event by using appropriate gates. Depending on which constellation of causes is necessary to lead to the top–event, these intermediate events are connected through OR or AND gates. Intermediate events are causes which have to be analyzed more deeply until the granularity is fine enough for the intended analysis. Causes which do not have to be analyzed in more detail

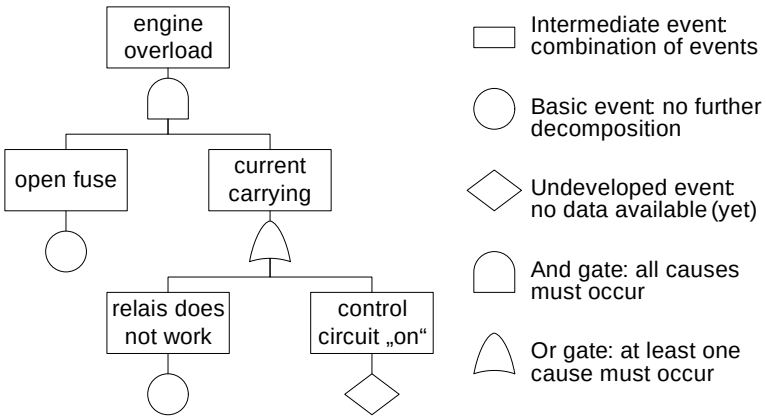

Figure 1.7: Example fault tree of an overloaded engine.

represent the leafs of the fault tree. They are either basic events or undeveloped events which have not (yet) been detailed. The latter can be for example system part failures which have not been described yet. Note that each event in an FTA is a fault (like error states, malfunctions, or disturbances) and thus not desired.

The example fault tree in Figure 1.7 examines the top–event "engine overload". This can be caused by an "open fuse" and "current carrying" wire. While the event open fuse is a basic event, the current carrying event is further detailed. The "current carrying" event can be caused by a broken relais or the status "on" of the control circuit. Again, the broken relais is a basic event, whereas the "control circuit on" event is an undeveloped event and might be described later. Note that this example is fictional and makes no claim to represent an actual situation.

For a qualitative analysis all basic events leading to the top–event are combined in a cut set. They can be derived from the structure of a fault tree, for example the cut set of the example fault tree in Figure 1.7 is:

$$\{\{\text{open fuse}, \text{broken relais}\}, \{\text{open fuse}, \text{control circuit on}\}\}$$

in disjunctive normal form (DNF). Cut sets with only a single element are called single point of failures (SPFs). This single event is enough to cause the top–event. Another indicator for weak spots of a system is if a basic event occurs in multiple cut sets (open fuse, in our example). The weak spot is the component causing the corresponding basic event.

Apart from the qualitative assessment the FTA can also analyze a system in a quantitative manner. Given information about the probability of the occurrence of basic events, it is possible to calculate the overall probability of the occurrence of the top–event. For this simple quantitative analysis the occurrence of basic events has to be statistically independent. The "merging points" of the fault tree are then represented as simple arithmetic operators, the AND is represented by a multiplication and the OR is represented by an addition of the probabilities. The accumulated probability of the

top–event of our example in Figure 1.7 is then:

$$P(\text{engine overload}) = P(\text{open fuse}) * (P(\text{broken relais}) + P(\text{control circuit on}))$$

If the system structure described using the SPES meta–model is decomposed in the same manner as a fault tree, then it is simple to generate the structure of a fault tree from the model specification [9]. However, normally additional information e. g. about the basic event combination (AND- and OR–gates) or the differing cause decomposition is required. Nevertheless, traceability links between system blocks should provide quite valuable information which could be relevant not only for an FTA. The probabilities of the occurrence of basic events can be stored in user attributes.

1.3.3 Failure Mode and Effects Analysis (FMEA)

The Failure Mode and Effects Analysis represents a preventive safety analysis approach. It is utilized to identify and assess potential failure causes as early as possible, e. g. during the early design phases. This helps to prevent control and further failure costs during the production or even the operation phase. Furthermore, a systematic failure analysis approach inhibits further repetition of design faults in other products.

FMEAs should be applied in early product life cycle phases e. g the concept and design phase, since it proved to be most beneficial for a cost–benefit analysis. The sooner a fault is discovered, the cheaper its correction can be.

The FMEA can be decomposed into the following five steps:

1. **Determine the system/process** A system or process consists of elements to describe the system or process. In this step the life cycles under analysis have to be determined e. g. service of the facility, maintenance of the facility, or repairs.

2. **Represent the functionality** Elements have different functions or tasks in a system. So for each element they have to be determined. The interaction between functions is called function structure and can be described with function trees or function block diagrams.

3. **Fault analysis** A fault and hazard analysis has to be conducted for each system/process, element, and function. Fault functions that are derived from functions represent potential faults or hazards resulting from the respective function. Possible fault or hazard causes are fault functions of subordinated elements. Implications of faults and hazards are fault functions of superordinated elements.

4. **Risk assessment** In order to assess the risk there are three criteria which have to be rated in numbers to calculate the RiskPriorityNumber (RPN). The RPN is used to prioritize the corrections of fault and hazard causes. The three criteria are:

   S denotes the **S**everity of a fault effect. The higher the number the more severe the implications are.

O denotes the **O**ccurrence probability of a fault cause.

D denotes the **D**etection probability of the fault cause.

The RPN is then calculated by multiplying the S, O, and D numbers: $RPN = S * O * D$.

5. Counter measures and corrections A high RPN implicates changes with the following priority (starting with the highest):

1. Change of concept to eliminate the fault cause or to reduce its importance.
2. Increase the concept's reliability to minimize the occurrence probability of the fault cause.
3. Improve the detectability of the fault causes to avoid further testing.

In order to minimize the risk additional measures can be described along with a person in charge and a deadline. After the optimization or the change of a concept all five steps of the FMEA have to be executed again. For increased concept reliability or improved detectability step 4 and 5 have to be repeated.

Usually the FMEA results are then organized in a table where each row shows the effect of (a subset of) the failure–modes under consideration, if any. Furthermore, each row contains information about the counter measures and their effects. This organized overview over FMEA items, hazards, and their effects on the safety of a system helps to identify critical safety issues and to define a measurable quality assessment.

Bibliography

[1] AUTOSAR Website. URL: `http://www.autosar.org`.

[2] Arp 4761 guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment, sae arp 4761., 1996.

[3] Matthias Büker, Alexander Metzner, and Ingo Stierand. Testing Real-Time Task Networks with Functional Extensions Using Model-Checking. In *14th IEEE Conference on Emerging Technologies and Factory Automation, 2009*, 2009.

[4] Werner Damm and David Harel. LSCs: Breathing Life into Message Sequence Charts. *Formal Methods in System Design*, 19(1):45–80, 2001.

[5] Werner Damm, Angelika Votintseva, Alexander Metzner, Bernhard Josko, Thomas Peikenkamp, and Eckard Böde. Boosting re-use of embedded automative applications through rich components. In *Proceedings, FIT 2005 - Foundations of Interface Technologies*, 2005.

[6] Henning Dierks, Alexander Metzner, and Ingo Stierand. Efficient Model-Checking for Real-Time Task Networks. *Second International Conference on Embedded Software and Systems*, 0:11–18, 2009.

[7] Reinhold Heckmann, Marc Langenbach, Stephan Thesing, and Reinhard Wilhelm. The influence of processor architecture on the design and the results of WCET tools. *Proceedings of the IEEE*, 91(7):1038–1054, 2003.

[8] OFFIS. Spes2020 architecture modeling. Technical report, OFFIS, 2010.

[9] Thomas Peikenkamp, Antonella Cavallo, Laura Valacca, Eckard Böde, Matthias Pretzer, and Ernst Moritz Hahn. Towards a unified model-based safety assessment. In Janusz Górski, editor, *SAFECOMP*, volume 4166 of *Lecture Notes in Computer Science*, pages 275–288. Springer, 2006.

[10] Daniel Ratiu, Wolfgang Schwitzer, and Judith Thyssen. A System of Abstraction Layers for the Seamless Development of Embedded Software Systems. SPES 2020 Deliverable D1.2.A-2, The SPES 2020 Project, October 2009.

[11] D. H. Stamatis. *Failure Mode and Effect Analysis: FMEA from Theory to Execution.* Amer Society for Quality, 1995.

[12] W. E. Vesely, F. F. Goldberg, N. H. Roberts, and D. F. Haasl. Fault tree handbook. Technical report, Division of Systems and Reliability Research, Office of Nuclear Regulatory Research, U.S. Nuclear Regulatory Commission, 1981.

[13] W. E. Vesely, F. F. Goldberg, N. H. Roberts, and D. F. Haasl. *Fault Tree Handbook*. U.S. Nuclear Regulatory Commission, Washington, DC, 1981.

[14] Raphael Weber, Eike Thaden, and Philipp Reinkemeier. Specification of the Reference Architecture. SPES 2020 Deliverable D3.2.A, The SPES 2020 Project, OFFIS, November 2009.