Chapter 4
Methods for the Domain-Spanning Conceptual Design

Harald Anacker, Christian Brenner, Rafal Dorociak, Roman Dumitrescu, Jürgen Gausemeier, Peter Iwanek, Wilhelm Schäfer, and Mareen Vaßholz

Abstract. The development of self-optimizing systems is a highly interdisciplinary task, as several domains are involved. Existing design methodologies do not adress this issue, as they focus on the respective domain; a holistic domain-spanning consideration of the system occurs – if at all – only rudimentally. The partial solutions developed by the respective domains may be optimal from the point of view of this domain. However, it does not automatically mean, that the sum of the optimal domain-specific solutions forms the best possible overall solution: "the whole is more than the sum of its parts". This especially holds true for the early design phase, the conceptual design. Its result is the so-called principle solution, which is further refined in the domain-specific design and development. Thus, a great need for methods arises which support the domain-spanning conceptual design for self-optimizing systems in a holistic manner. In this chapter we will introduce such methods. In particular, we will explain the specification technique for the domain-spanning description of the principle solution of a self-optimizing system. Furthermore, methods are explained which support the creation of the principle solution. This includes a method to ensure the consistency of application scenarios, a method for the design of the system of objectives, which is crucial for a self-optimizing system, as well as a method for the re-use of proven solutions for recurring problems (solution patterns). Finally, some analysis methods are explained that are performed on the specification of the principle solution. These are: the early analysis of the reliability and the analysis of the economic efficiency.

The development of self-optimizing systems is structured into the domain-spanning conceptual design and the domain-specific design and development as explained in Chap. 3. During the conceptual design, experts from the domains of mechanical, electrical/electronic, control and software engineering work together and develop the principle solution. The involvement of the different domains in the

J. Gausemeier et al. (eds.), *Design Methodology for Intelligent Technical Systems*, Lecture Notes in Mechanical Engineering,
DOI: 10.1007/978-3-642-45435-6_4,

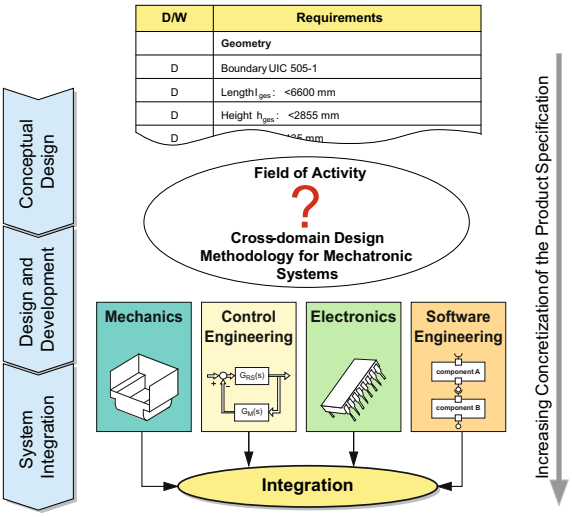


Fig. 4.1 Central challenge: a specification technique for the description of the principle solution of a self-optimizing mechatronic system [24]

development process for self-optimizing systems as well as the integration of partial intelligence in self-optimizing systems call for new development methods as well as new development tools. Existing design methodologies need to be fundamentally extended. This especially concerns the conceptual design. Certainly, the basic structure of the phases of existing design methodologies (formulation of requirements, definition of functions, etc.) [43] also applies for self-optimizing systems. Nevertheless, such aspects as domain-spanning understanding, modeling of application scenarios, partial intelligence and system behavior have to be considered as well. Due to the involvement of different domains, devices have to be provided which allow fundamental understanding of the whole system by all developers from the very beginning of the development process. The gap between the list of requirements, which is more or less a rough specification of the total system and, hence, leaves much space for interpretation, and well-established specification techniques of the domains involved needs to be closed (Fig. 4.1) [24]. Otherwise, time and cost intensive iterations and failure can emerge when the engineers have to integrate their results in the domain-specific design and development.

To overcome these challenges a holistic description of the principle solution for the whole system is necessary. It describes the basic structure and operational mode of the system, as well as its desired behavior. Moreover, it considers different aspects such as environment, requirements and application scenarios - just to name a few. These different aspects form a coherent system as all aspects correlate with each other. To secure the overall consistency of the principle solution is a challenge, which can only be overcome with an adequate software support. Hence, a software tool which supports the modeling of the principle solution is also a necessary.

A detailed analysis of the state of the art has shown that there are a number of approaches for the specification of mechatronic systems [24]. None of these approaches, however, fulfill the aforementioned requirements to a full extent. In

order to address this need for action, we have developed a specification technique CONSENS [8] for the domain-spanning description of the principle solution for self-optimizing systems, which is introduced in the following section [24].

This chapter is structured according to the reference process for the conceptual design (cf. Sect. 3.2). First, the specification technique CONSENS for the domain-spanning description of the principle solution for self-optimizing systems is introduced (Sect. 4.1). Sect. 4.3 - 4.5 explain methods, which support the creation of the principle solution. Sect. 4.3 shows, how to ensure the consistency of application scenarios regarding the discrete behavior specified in them. In Sect. 4.4 it is explained, how the system of objectives, the backbone for self-optimization, is modeled. In Sect. 4.5 we will describe, how solution patterns are used during the conceptual design, i.e. how established solutions for recurring problems can be reused during the specification of the principle solution. The method for the product structuring for self-optimizing systems in the conceptual design is explained in Sect. 4.6. Finally, we will explain how first analyses can be conducted on the description of the principle solution at an early stage. These are the early analysis of the reliability (Sect. 4.7) as well as the analysis of the economic efficiency (Sect. 4.8).

4.1 Specification Technique CONSENS for the Description of Self-optimizing Systems

Rafal Dorociak, Roman Dumitrescu, Jürgen Gausemeier, and Peter Iwanek

In accordance to the reference process for the conceptual design (cf. Sect. 3.2) the specification technique CONSENS is used to describe the domain-spanning principle solution for the self-optimizing system [18, 24]. The **principle solution** describes the basic structure (e.g. components of the system and interactions between them), operational mode of the self-optimizing system, and its desired behavior. The principle solution forms the basis for the communication and cooperation of the domains involved (e.g. mechanical and software engineering) in the course of the further domain-specific design and development.

The description of the principle solution of self-optimizing systems consists of eight interrelated aspects. As shown in Fig. 4.2 these aspects are requirements, environment, system of objectives, application scenarios, functions, active structure, shape and behavior. The aspects are computer-internally represented as partial models. The aspects relate to each other and ought to form a coherent system. We will describe each aspect in the following.

Environment: There are many interrelations between the system and its environment. Therefore, it is important to analyze the environment of the system to ensure that the final system will work properly in it, without any restrictions caused by not considered interactions. For this purpose the specification technique CONSENS offers the aspect environment. This aspect describes the embedding of the system within its environment; the system itself is treated as a "black-box". In particular,

[8] CONceptual design Specification technique for the ENgineering of complex Systems.

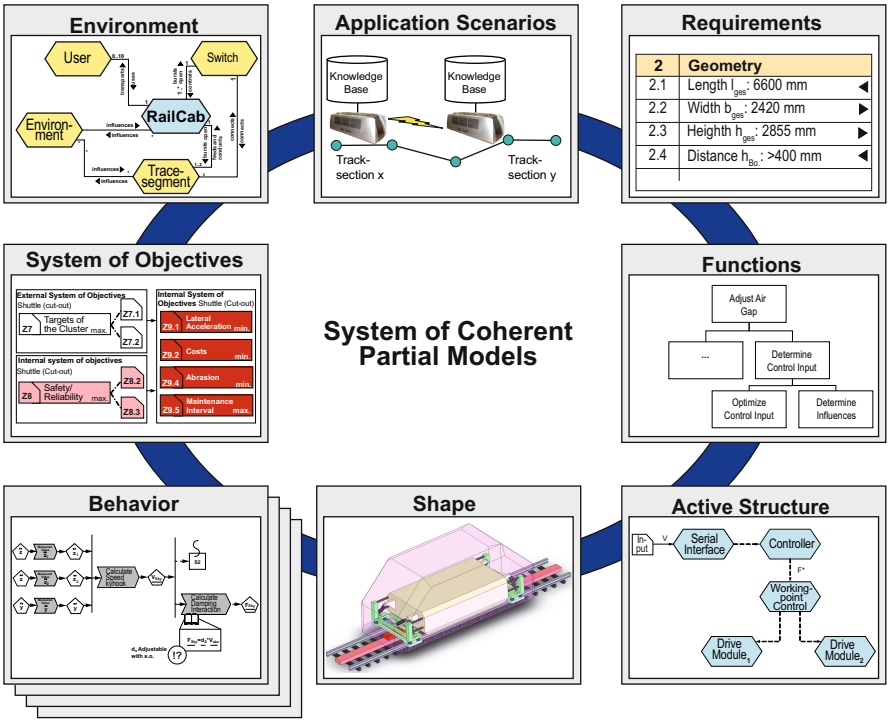


Fig. 4.2 Partial models for the domain-spanning description of the principle solution for self-optimizing systems [24]

other elements of the environment (e.g. the user, other technical systems or the underground) and their interrelation with the system are described. Relevant influences of the environment on the system such as weather, temperature and humidity are described as well. Influences which have an disturbing impact on the system operation are marked as such. The identification of the relevant influences is supported by respective catalogues and check lists. The interrelations between the system under development and elements of the environment are represented as flows. In principle, three types of flows can be distinguished: information flows, energy flows, and material flows.

Figure 4.3 shows the specification of the environment of the RailCab. In particular, it is shown that the driving behavior of the RailCab is affected by the weight of users and cargo as well as influences from the environment, the state of the track sections as well as the abrasion of the RailCab itself.

Application Scenarios: Application scenarios form first concretizations of the system's behavior. They describe the most common operation modes of the system and the corresponding behavior in a rough manner. Every application scenario describes a specific situation (e.g. start-up, failure of the system or an interaction with

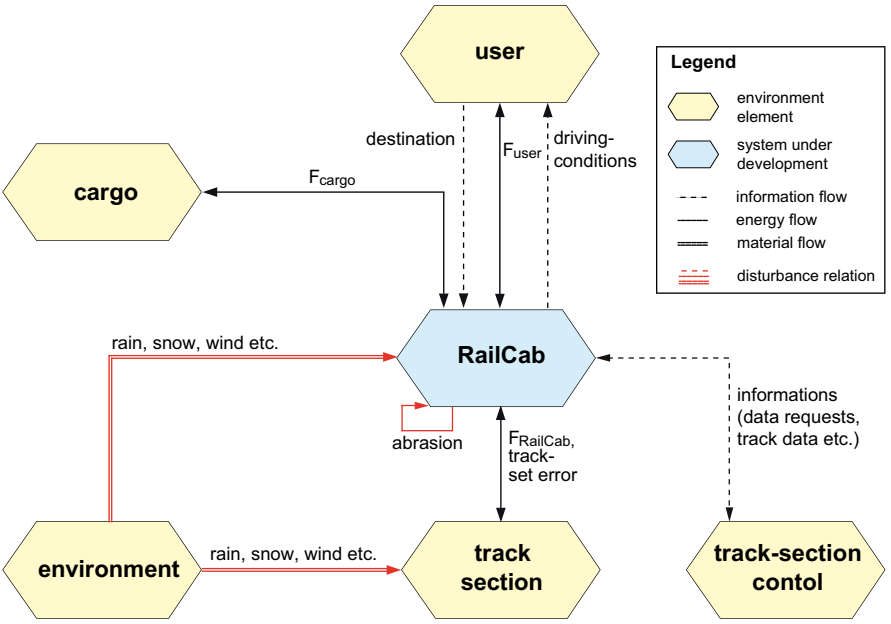


Fig. 4.3 The partial model environment of the RailCab (excerpt) [24]

the user), and the required behavior of the system for this situation. Thus, application scenarios characterize a problem and the possible solution for it. By modeling the application scenarios requirements and potential operational modes for the system can be identified. Fig. 4.4 shows the application scenario "AS12: Drive onto next track section" for the RailCab as an example. The description of an application scenario includes e.g. general information, like a title, an ID, the date of change, and a textual characterization of the application scenario; to gain greater insight into the application scenario a sketch can be added.

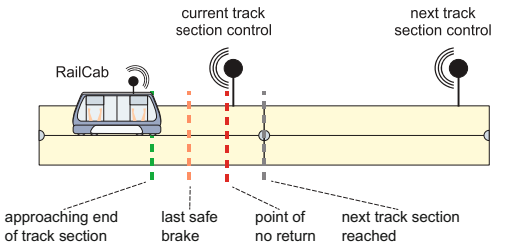
Requirements: Based on the general problem definition and the aspects environment and application scenarios, the requirements for the system under development can be defined and modeled. Requirements present an organized collection of requirements that need to be fulfilled by the system under development (e.g. features of the system, overall size, performance, quality). Requirements allow the engineering team to expose what is expected from the future system. They form a corner pillar for the validation and verification in further development phases. Requirements are represented in tabular form; the requirements list. requirements can be decomposed into sub-requirements to structure multiple requirements. For example; "height", "length" and "width" can be sub-requirements of the "size" of an element. Each requirement in the requirements list has an ID, is verbally described, and, if possible, concretized by corresponding parameters (e.g. temperature, length, velocity) and values (e.g. the RailCab should be able to reach a velocity of 100 km/h). Several checklists assist the identification of requirements; see for example

Fig. 4.4 Description of the application scenario "Drive onto next Track Section" (excerpt) [24]

A	**Application Scenario** Drive onto next track section	Jul. 17, 2010	**AS12**	Page: 1

Description of the partial development task AS12: When the RailCab is driving on a track section, it is at some point notified that it approaches the end of the track section. Then, the RailCab must obtain the information, whether it may enter the next track section, from the corresponding section control. This information must be available to the RailCab before the RailCab reaches the point of the last safe brake. This point precedes the point of no return, beyond which it cannot be guaranteed that braking will safely stop the RailCab before it enters the next track section.

Principle solution for AS12: The RailCab, when reaching the end of the track section, sends a request to enter the next track section to the section control responsible for the next track section. Then the section control replies, stating whether entering the track section is currently allowed or not. The reply is sent in time for the RailCab to receive it before it reaches the point of the last safe brake.



Reference to Application Scenario Specific Cut-out of
- Requirements
- Environment
- Active Structure
- ...

Fig. 4.5 Requirements on the RailCab (excerpt)

Requirements list		
No.	**Requirement description**	**D/W**
1	Geometry	
...	...	...
1.9	Entrance should be possible from both sides.	D
1.10	Optimal aerodynamics for single and convoi drive modes.	D
1.11	Modular construction.	D
...	...	...
2	Kinematics	D
2.1	The vehicle has a steering system.	D
...	...	...
7	Safety	
...	...	...
7.9	Provide emergency mechanisms and exits.	D
7.10	Minimize sensitivity to the side wind.	W
...	...	...

[11, 43, 47]. Requirements are separated into demands and wishes [43]. If needed, the requirements can also be divided into functional (e.g. the doors of the RailCab should be able to close automatically) and non-functional requirements (e.g. the doors of the RailCab should be red). An excerpt of the requirements list for the RailCab is shown in Fig. 4.5; demands are marked with "D", wishes with "W".

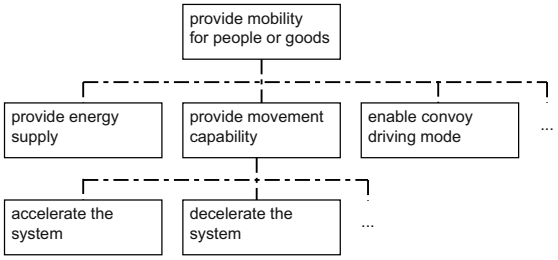


Fig. 4.6 Functions of the RailCab (excerpt)

Functions: Based on the requirements, the functions for the system under development can be defined. The aspect functions describes the hierarchical subdivision of the desired functionality of the system. A function is the general and required relationship between input and output parameters, with the aim to fulfill a task. For the specification of function hierarchies, we use a catalogue with functions which is based on the works of Birkhofer (1980) [5] and Langlotz (2000) [35]. This catalogue has been extended by functions, which describe self-optimizing functionality. Functions are realized by solution patterns and their concretizations. Starting with the overall function (e.g. provide mobility for people or goods), a subdivision into sub functions takes place (e.g. accelerate the system) until useful solution patterns can be found for the functions (e.g. linear motor). The use of solution patterns is described in a detailed manner in Sect. 4.5. Figure 4.6 shows a section of the function hierarchy of the RailCab. After the definition of the overall and sub-functions the classification scheme by Zwicky (morphological matrix) can be used, for the systematic combination of certain solutions [43]. In this classification scheme, the sub-functions and the appropriate solutions are entered into the rows of the morphological matrix. By systematically combining a solution fulfilling a specific sub-function with the solution for a neighbouring sub-function, one obtains an overall solution in the form of a possible conception. In this process, only those solution that are compatible should be combined [43].

Active Structure: Based on the functions and the combination of the chosen solutions the active structure for the system under development can be modeled. Thus, in contrast to the aspect environment (Black-Box view on the system and its context), the active structure concretize the system (White-Box view). The active structure defines the internal structure and the operational mode of the system. It describes system elements (e.g. chosen solutions), their attributes as well as the relationships between system elements (material, energy and information flows as well as logical relationships). Depending on the level of concretization, system elements may be described abstract (e.g. temperature sensor) or specific (e.g. resistance thermometer). If necessary, it is also possible to model elements of the environment (e.g. user) and their interaction with elements of the system (e.g. interaction of the user with the human-machine-interface of the system).

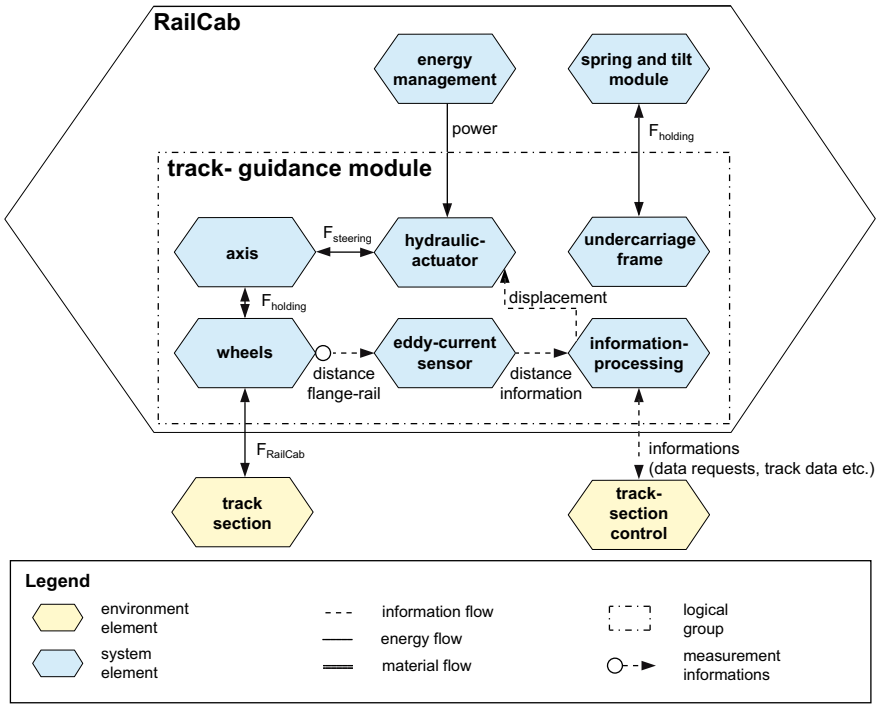


Fig. 4.7 The partial model active structure of the RailCab (excerpt) [24]

Figure 4.7 visualizes an excerpt of the active structure for the RailCab. The active structure consists of system elements such as the Energy Management, the Spring and Tilt Module or the Track-guidance Module. To show system elements of the Track-guidance Module at the same hierarchy level as the other modules, logical groups can be used. The track-guidance module consists of eddy-current sensors (in Fig. 4.7 only one of them is shown), the hydraulic actuator, the axis, the wheels etc. The hydraulic actuator can change the position of the axis and thus of the wheels. In addition, the wheel has an mechanical contact to the rail. The eddy-current sensor measures the distance between the flange and the rail. This is specified with a measurement information flow. The information of the eddy-current sensor are sent to the information processing unit of the Track-guidance Module. The information processing unit calculates the needed displacement force, based on the sensor information and information from the track-section control. The needed displacement will be sent to the hydraulic actuator, which changes the steering position then. A closed control loop between sensor, actuator, information processing, axis, and the wheels results.

System of Objectives: This aspect describes external, inherent and internal objectives of the system and their interrelations. An excerpt of the system of objectives of the RailCab is shown in Fig. 4.8. External objectives are set from the outside of

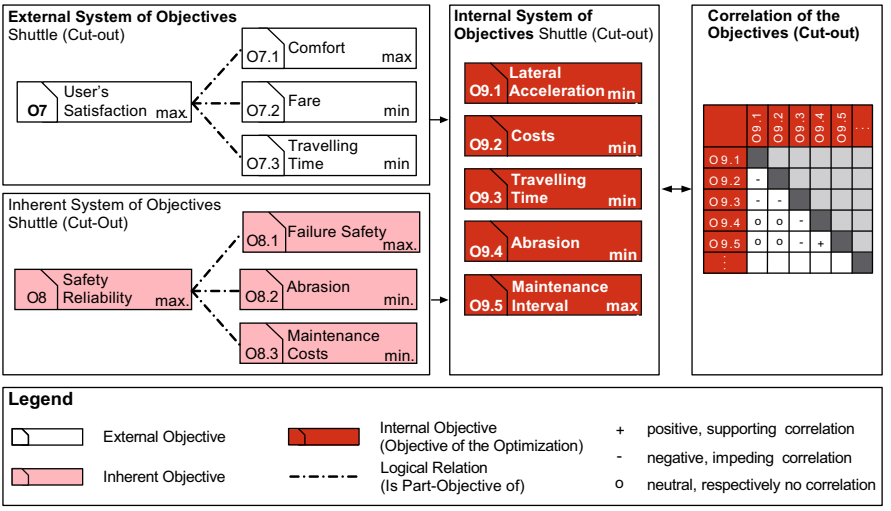


Fig. 4.8 The partial model system of objectives of the RailCab (excerpt) [24]

the self-optimizing system; they are set by other systems or by the user (e.g. "maximize user satisfaction"). Inherent objectives reflect the design purpose of the self-optimizing system. Inherent objectives of the RailCab are for example the objectives "maximize dependability" and "minimize energy consumption". Objectives build a hierarchy and each objective can thus be refined by sub-objectives (e.g. "maximize safety" is a possible sub-objective of the objective "maximize dependability", "maximize comfort" and "maximize driving speed" are possible sub-objectives of the objective "maximize user satisfaction"). Inherent and external objectives that are pursued by the system at a given moment during its operation are called internal objectives. The selection of internal objectives and their prioritization occurs continuously during the operation of the system. "Maximize comfort" and "maximize safety" are examples of internal objectives. Only the internal objectives are part of the self-optimization. During the operation of the self-optimizing system some of its objectives may be in conflict to each other, as they can not be pursued both to the full extend at the same time. In such cases a prioritization of the objectives has to take place. For instance, during the adjustment of the driving speed the objectives "maximize driving speed" and "minimize energy consumption" are in conflict to each other, as energy consumption typically increases with increasing driving speed. Such potential mutual influences between internal objectives are modeled in an influence matrix. In particular, the influence matrix shows which objectives may influence each other in a negative way. Such a potential negative mutual influence may be an indication for the need for self-optimization. In Sect. 4.4 we will describe how the system of objectives is designed.

Shape: This aspect describes the first definition of the shape of the system within the conceptual design. In particular, the working surfaces, working places

and frames of the system are described in a rough manner. In mechanical engineering the aspects shape and active structure form the core of the principle solution. It is very important to model the draft of the shape during the domain-spanning conceptual design. For example, geometric restrictions for wires or mechanical components can be recognized by the different domains involved and the communication and cooperation between them is improved. Thus, expensive corrections can be avoided (e.g. too short wires in the Airbus A380 in 2006 [7]). The computer-aided modeling is performed using 3D CAD systems.

Behavior: A self-optimizing system is characterized by different kinds of behavior (e.g. kinematic, dynamic and reactive behavior). In order to describe the behavior of such systems, a group of behavior models is used: there are three partial models to specify the behavior. We distinguish between the partial models behavior–states, behavior–activities and behavior–sequence. The usage of the diagrams depends on the underlying development task. Additional kinds of behavior, such as kinematics, dynamics or electro-magnetic compatibility can be specified additionally.

- The partial model **behavior–states** describes all possible states of the system, all possible state transitions as well as events which initiate state transitions. Events correspond to external influences on a system or a system element as well as to already finished activities. For example, the lighting system of the RailCab can have two different states: lights on and lights off. The user-event "switch power button" causes a state transitions from "lights off" to "lights on" and vice versa.
- The partial model **behavior–activities** describes the operation process of the system , i.e. operations and tasks of the system that are performed during its operation. This especially includes operation processes which are performed in order to implement the self-optimization process (e.g. "determine the fulfillment of current system objectives", "select adequate parameters and configuration", etc.). We call such operation processes adaptation processes.
- The partial model **behavior–sequence** describes the interaction of several systems or system elements. The messages being exchanged during the interaction of those system elements are modeled in a chronological order. In Sect. 4.3 some examples of behavior description with sequence diagrams are introduced in a detailed manner.

It is necessary to alternately work on the aspects and the corresponding partial models although there is a certain order. This order is defined by the reference process for the conceptual design (cf. Sect. 3.2). In contrast to other system modeling approaches such as UML [24, 44] or SysML [20] the specification technique CONSENS is strongly interconnected with the reference process and focuses on self-optimizing mechatronic systems.

As stated before, the partial models form a coherent system and are strongly interconnected. These interconnections are modeled as **cross-references** between partial models. Tab. 4.1 shows some examples of such partial model spanning cross-references. There are e.g. bidirectional cross-references between requirements and functions, between requirements and system elements as well as between system

Table 4.1 Interrelations between the partial models (excerpt) [24]

Construct	Partial Model	Kind of Interrelation	Construct	Partial Model
System Element	Active Structure	Realizes	Function	Functions
System Element	Active Structure	Performs	Activity	Behavior/Activities
System Element	Active Structure	Takes	State	Behavior/State
System Element	Active Structure	Persuades from	Objective	System of Objectives
System Element	Active Structure	Has (opt.)	Volumes	Shape
Activity	Behavior/Activities	Results from	Function	Functions
Requirement	Requirements	Sets Boundaries for	Volumes	Shape
Requirement	Requirements	Decides	Function	Functions
Function	Function	Results from	Requirement	Requirements
Influence/Event	Environment	Activates	State	Behavior/State
Influence/Event	Environment	Activates	Activity	Behavior/Activities
•••				

elements and functions (e.g. a "System Element" from the "Active Structure" "Realizes" a certain "Function"' from the partial model "Functions"). Based on the specification of cross-references, analyses such as requirements traceability can be realized [23].

4.2 Software Support for the Specification of the Principle Solution

Rafal Dorociak and Jürgen Gausemeier

To secure the overall consistency of the principle solution and to manage its complexity, a software support is necessary. The software tool **Mechatronic Modeller** supports the creation and editing of the specification of the principle solution [23, 25]. It was developed within the research project "VireS – Virtual Synchronization of Product Development and Production System Development" founded by the German Federal Ministry of Education and Research (BMBF) in cooperation with the software company itemis. The Mechatronic Modeller is a dedicated software solution, which is fully aligned with the specification technique CONSENS. It offers a separate editor for each partial model. Figure 4.9 shows the graphical user interface of this software tool.

Within the model browser the elements of the principle solution are presented as a tree. This tree can be used to navigate within the principle solution. The currently processed partial model is shown on the right in the diagram view together with a

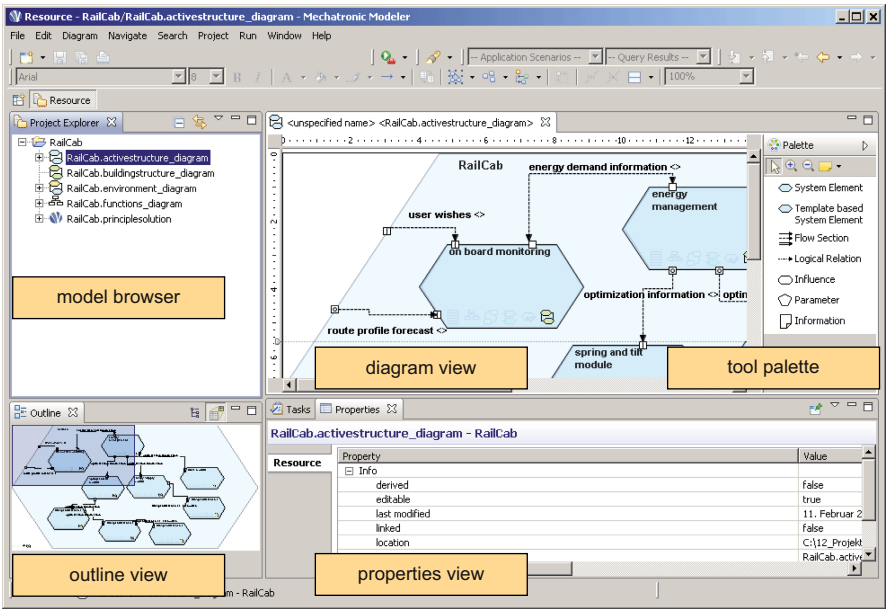


Fig. 4.9 Screenshot of the Mechatronic Modeller showing the active structure editor [23]

tool palette. Within this diagram view the respective partial model can be modified. Using the tool palette new elements can be added. The outline view (bottom left) continuously shows the outline of the whole diagram. Within this view the user can navigate through the whole diagram. This allows the user to navigate to sections of the partial model which are currently not shown in the diagram view.

A so-called metamodel has been defined for the specification technique. It defines [52]:

- which model elements are available during the description of the principle solution as well as how they are related to each other (abstract syntax); for instance, states can be linked to other states using a relation, and
- criteria for well-formedness (static semantics); for instance, the names of states must be unique in the scope of the statechart.

In particular, the metamodel describes all possible interrelations between the different partial models.

The Mechatronic Modeller is based upon this metamodel. Thus, each principle solution modeled with the Mechatronic Modeller is computer-internally represented as a data model, which is an instance of this metamodel. Given that all constraints have been formally defined in the metamodel, the conformance of such a model to the metamodel can be easily checked, allowing immediate feedback for the developer in case of modeling errors.

In addition to the metamodel, the following aspects of the specification technique had to be defined during the development of the tool:

- how the models will be graphically represented (concrete syntax); for instance, a state is represented as a rounded rectangle, and
- the meaning of the different modeling constructs in a particular principle solution (dynamic semantics); for instance, state transitions are triggered when the event at the transition is fired.

Using a precise definition of the dynamic semantics of a language, a model can be simulated, analyzed, and/or formally verified [23, 25].

The Mechatronic Modeller addresses all particularities of the specification technique. The very important aspect of usability can therefore be appropriately addressed. The aim is to hide the complexity of the model from the developer. Thus, several functions have been incorporated into the Mechatronic Modeller which support working with the specification technique and make the tool more comfortable and enables an intuitive use. In particular, complex manipulations such as partial model reorganization by incorporating or deleting of hierarchy levels, are provided by the tool. Furthermore, cross-references between elements of different partial models are stored in the data model. Thus, Mechatronic Modeller is capable of handling complex dependencies between elements of different partial models within the principle solution. For example, it is possible to check which requirements have not yet been realized by functions or system elements (static semantics checks). In particular, requirements traceability is possible, e.g. if a particular system element needs to be exchanged, then the developer can examine which requirements had to be originally met by it [23].

4.3 Consistency Analysis of Application Scenarios

Christian Brenner and Wilhelm Schäfer

During the development of a self-optimizing system, the definition of the system behavior is highly important. Application scenarios usually form the earliest description of the system behavior (cf. Sect. 4.1). In the course of the progressing conceptual design this description is further refined; the partial models behavior–states and behavior–activities are used for this purpose. Eventually, these partial descriptions of the behavior are combined into one scenario-spanning model of the overall system behavior. It is a challenge to ensure the consistency of the aforementioned partial descriptions of behavior. In order to support this difficult task, a method to ensure the consistency of application scenarios has been developed. This method is based on a formal description of application scenarios [29]. It allows for the early detection and correction of inconsistencies between partial descriptions of the behavior.

Figure 4.10 shows the procedure model of the method. The starting point is the definition of the discrete system behavior by using application scenarios. The method consists of the following three phases:

Phase 1 - formalization of the discrete behavior: In order to automatically detect inconsistencies in the modeled system behavior, the models of interest have to

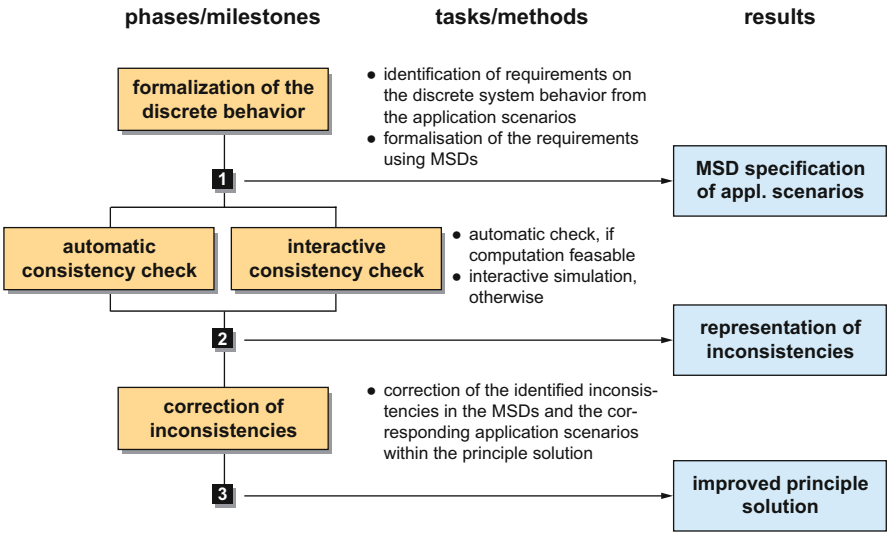


Fig. 4.10 The procedure model of the method for ensuring the consistency of application scenarios [29]

be specified in a formal way. We will demonstrate this by using the example of the application scenario "AS12: Drive onto next track section" shown in Fig. 4.4 (Sect. 4.1, p. 122). It defines how the RailCab and the track section control interact, when the RailCab is about to enter a track section. In particular, it defines requirements regarding the system behavior using text (e.g. in the principle solution of AS12: "[...] the RailCab, when reaching the end of the track section, sends a request [...]") and illustrations. Also, it may contain assumptions about the system environment (e.g. in the description of AS12: "[the RailCab] is at some point notified that it approaches the end of the track section."). However, these requirements and assumptions are at first specified only informally using natural language. In order to process the application scenario automatically, we first need to formalize them. We use **Modal Sequence Diagrams** (MSDs) for this purpose [30]. They have been adapted for mechatronic systems [29] by taking into account real-time behavior and assumptions about the system environment. We distinguish requirement MSDs and assumption MSDs. Requirement MSDs model requirements on the system based on the respective application scenario. Assumption MSDs specify assumptions about the environment of the system.

Figure 4.11 shows the MSD specification formalizing the application scenario "AS12" from Fig. 4.4. It consists of three MSDs. The topmost MSD and the one in the middle are both requirement MSDs. The bottom MSD is an assumption MSD, which is indicated by the label *«EnvironmentAssumption»*.

In each MSD, the vertical dashed lines, called *lifelines*, represent the participants of the respective scenario. At the top of each lifeline, a label in a hexagon defines the corresponding system element or environment element. The application

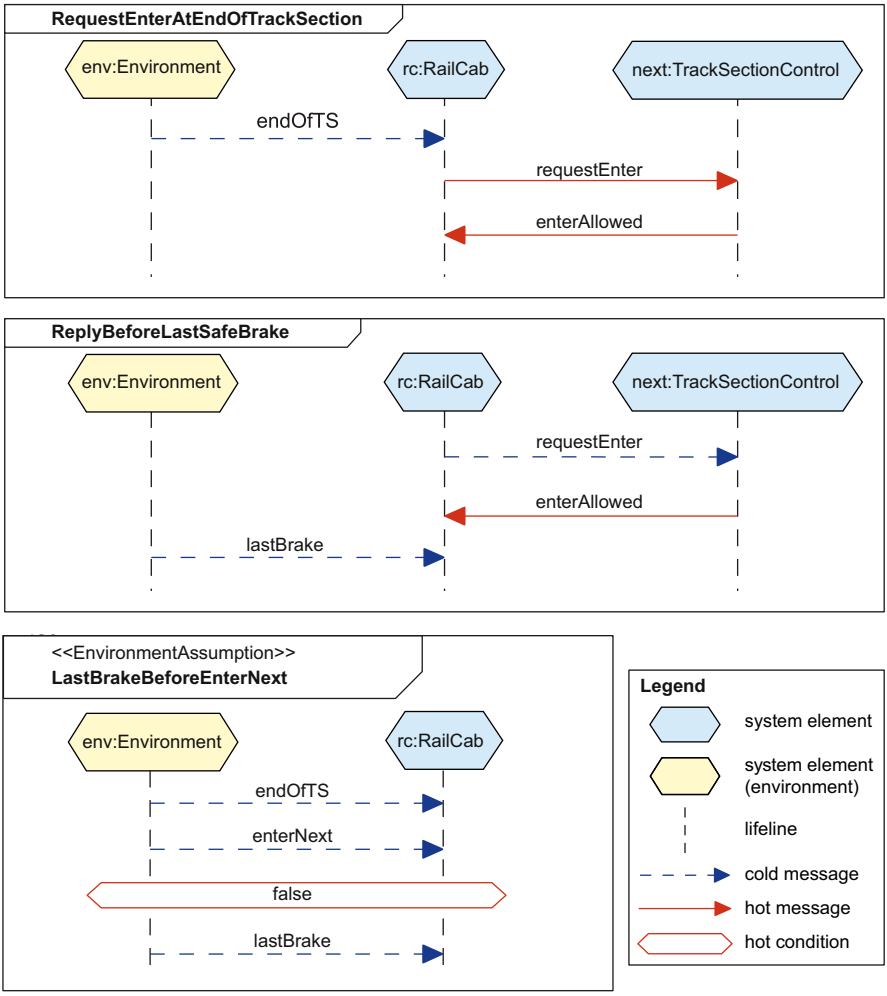


Fig. 4.11 Modal Sequence Diagrams for the application scenario of Fig. 4.4 [29]

scenario in Fig. 4.4 explicitly mentions a RailCab and a section control of the next track section. Consequently, both requirement MSDs contain a system element "rc" (representing a RailCab) and a system element "next" (representing the upcoming track section control). In addition, the environment is represented by the lifeline "env". The horizontal arrows in the MSDs are messages that are exchanged between system elements. Each arrow starts at the lifeline of the sender and ends at the lifeline of the receiver. The label at the arrow defines the type of message that is exchanged. Dashed arrows represent messages that may occur, but do not have to (e.g. *endOfTS* in the topmost MSD). These are called *cold messages*. Solid arrows represent messages that are required to occur (e.g. *requestEnter* in the topmost MSD). We refer to them as *hot messages*. The vertical position of the arrows in the MSD

defines the chronological order of the corresponding messages: the topmost message is expected to occur first, then the one below it, and so on.

As mentioned before, the MSDs in Fig. 4.11 formalize the requirements and assumptions that are informally expressed in the description of the application scenario in Fig. 4.4. For example, in the topmost MSD, the cold message *endOfTS* models the notification by the environment mentioned in the first sentence of the description of the development task of AS12 ("[...] the RailCab [...] is at some point notified that it approaches the end of the track section"). The system has to realize the requirements specified by the hot messages *requestEnter* and *enterAllowed* only after receiving the message *endOfTS*. According to the MSD, both messages have to be sent in this particular order. They model the requirements stated in the first sentence in the principle solution part of the description of the AS12 in Fig. 4.4 ("[...] the RailCab, when reaching the end of the track section, sends a request to enter the next track section to the section control [...]", "Then the section control replies [...]"). The second MSD formalizes the remaining requirement in a similar way: the reply must be sent on time, i.e. as long as braking is still allowed.

When the first message of an MSD is exchanged, this MSD becomes *active*. For example, the topmost MSD becomes active when the environment sends the message *endOfTS* to the RailCab "rc". Then, the message *requestEnter* is expected to occur next. The next expected message of an active MSD is referred to as an *enabled message*. After an enabled message has been sent, the next enabled message is the subsequent message in the order defined by the MSD. If, for example, the message *requestEnter* in the topmost MSD is enabled and is actually sent, *enterAllowed* is the next enabled message.

If a message occurs that is included in an active MSD, but is not currently enabled, then this is a *violation* of the MSD. Messages that are not included in an MSD can never violate it (e.g. *lastBrake* can never violate the topmost MSD). A violation of an MSD is a *cold violation*, if the enabled message is a cold message. It is a *hot violation*, if the enabled message is hot. A cold violation of an MSD turns the MSD inactive again. Assume, for example, that the MSD at the bottom of Fig. 4.11 is active and the message *enterNext* is enabled. Then, the message *lastBrake*, if it is sent, causes a cold violation and turns the MSD inactive. Note, that a cold violation does not indicate incorrect behavior. It only means that there is an allowed deviation from the scenario modeled by the MSD. A hot violation, on the contrary, models forbidden behavior of the system or unexpected behavior of the environment. If, for instance, the topmost MSD is active because *endOfTS* was sent, the next enabled message is the hot message *requestEnter*. If *requestEnter* is never sent, or if *enterAllowed* or *endOfTS* is sent instead, a hot violation occurs. Unlike a cold violation, a hot violation of a requirement MSD may never occur in the real system. A hot violation of an assumption MSD means that the environment does not behave as assumed. The system may not be used in such an environment.

After modeling the requirement MSDs and assumption MSDs for all application scenarios, they are combined into one complete MSD specification for the whole system. In Phase 2 this MSD specification is analyzed for inconsistencies.

Phase 2b - interactive consistency check: The *automatic consistency check* is computationally very expensive and, hence, not applicable for very complex MSD specifications. In case of such complex specifications, the engineer can instead perform an *interactive simulation*. This simulation does a stepwise evaluation of the specified behavior. The simulation can be influenced by the engineer by selecting the actions of the system, if the specification allows several alternatives. The engineer performing the simulation can also influence the simulated behavior of the environment. However, the interactive simulation only allows to consider individual simulation runs. For large systems, it is usually not possible to cover all possible executions. Therefore, the interactive simulation alone can not prove that the discrete behavior of the system described with application scenarios is completely consistent.

Phase 3 - correction of inconsistencies: Here the inconsistencies found during Phase 2, if any, are corrected. This happens in two ways. On the one hand, contradicting requirements are directly corrected by modification of the respective requirement MSDs. On the other hand, contradictions in the requirements can often be resolved by modeling additional environment assumptions using assumption MSDs. In both cases, the engineers also adapt the textual descriptions of the application scenarios in accordance with the changes in the MSD specification.

Tool support: For our method a software tool, called Scenario Tools, has been developed. It allows the engineer to create and edit MSD specifications as well as to validate the specification by using either the automatic consistency check or the interactive simulation, as described above.

All in all, with the presented method and tool the overall consistency of the behavior described with application scenarios is improved. This is very important, as potential inconsistencies in the behavior specification would otherwise be difficult to detect in later product development phases and could lead to problems regarding reliability, safety or availability.

4.4 Design of the System of Objectives

Rafal Dorociak and Jürgen Gausemeier

Sections 4.4 - 4.5 introduce methods, which support the development of the principle solution. We will begin with the specification of the system of objectives. The partial model **system of objectives** describes the objectives of the system which are subject to self-optimization and are therefore of particular importance for the design of self-optimizing systems. For the specification of the system of objectives, a method has been developed by Pook (2011) [45]. Using the method objectives of the system, their relationships to each other and potential conflicts are identified based on the description of the principle solution for the system. Altogether the method assists developers with the design of the information processing of the system, which then realizes self-optimization. During the development of the method some ideas and concepts from the Fault Tree Analysis (FTA) [4, 6, 32] and Failure Mode and

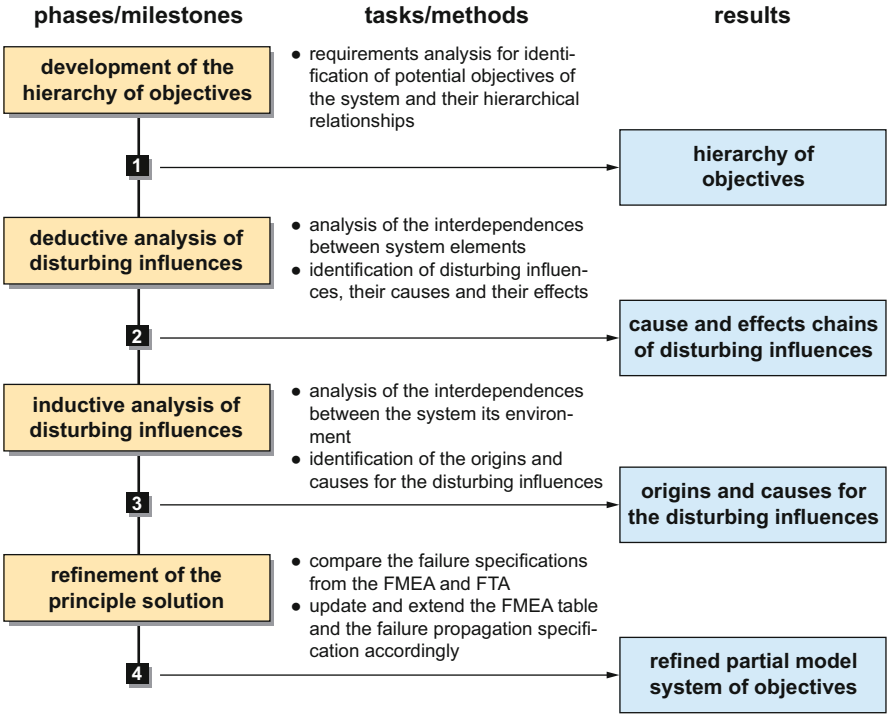


Fig. 4.12 The procedure model of the method for the design of the system of objectives

Effects Analysis (FMEA) [4, 6, 31] were adapted. Figure 4.12 shows the constituent phases of the method and the corresponding milestones:

Phase 1 - development of the hierarchy of objectives: The starting point is the identification of the possible objectives of the system. The objectives are contained in the list of requirements. The list of requirements of a complex mechatronic system usually contains a great number of entries, e.g. up to several thousands of requirements. In order to extract the objectives of the system from the list of requirements, the cross-references between the partial models of the principle solution are used (e.g. cross-references between functions and functional requirements and between system elements and functions). The starting points are the system elements which realize the information processing. The hierarchical dependencies between the identified objectives are found, analogously. The first phase is performed only once. The result is the hierarchy of the objectives of the system. Figure 4.13 shows a cut-out of the principle solution for the trace guidance module of the RailCab system. System elements, which realize the information processing, are identified first, e.g. the system element "data processing of the trace guidance module" (1). The corresponding requirements are then traced using the respective cross-references. One requirement corresponding to the system element "data processing of the trace guidance module" is the requirement "2.1.1. The vehicle determines autonomously

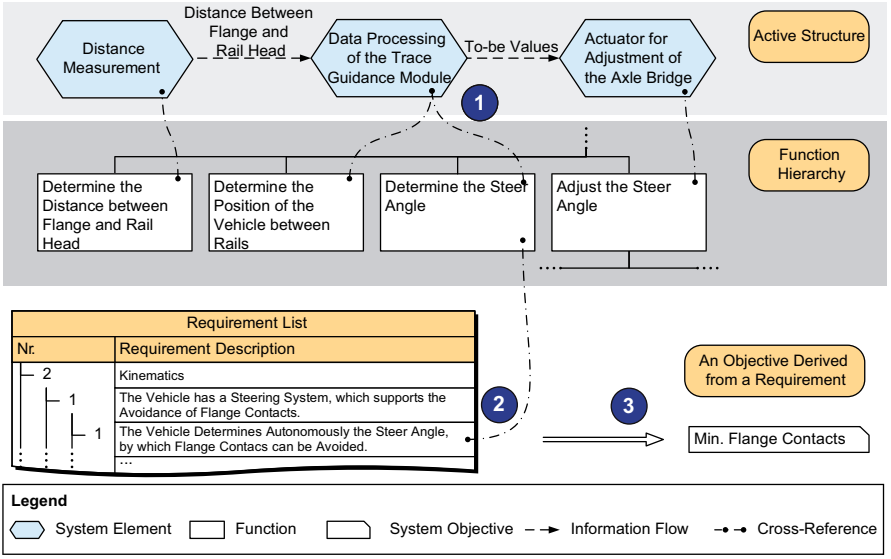


Fig. 4.13 Development of the hierarchy of objectives of the RailCab System (cut-out) [45]

the steering angle, where flange contacts can be avoided" (2). From this requirement the system objective "minimize flange contacts" is derived (3).

Further requirements and objectives are found, analogously. For instance, the requirement ("13.1. The vehicle has facilities for increasing the comfort during the transport of people"). From this requirement the objective "maximize traveling comfort" is derived. In the context of the RailCab system the objective "maximize traveling comfort" is subordinated to the previously identified objective "minimize flange contacts". In the partial model system of objectives this fact is modeled using the "is part of" relationship; a hierarchy of objective is constructed in this manner.

Phase 2 - deductive analysis of disturbing influences: Each of the objectives of the system found in Phase 1 is further examined. The disturbing influences are identified, which may occur during the operation of the system and have negative influence on the objective of the examined system. In order to identify these disturbing influences a Fault Tree [4] is built for each objective of the system. The top event of the Fault Tree is first postulated. It usually has the form: "the system is being disturbed while pursuing the objective x". The deductive analysis follows. It can be thought of as a "how-can" analysis [15]. The developer works in a top-down manner to find specific combinations of events, which could have occurred for the top-event to have taken place. We propose the following schema: The left branch of the tree describes cases, in which the system is disturbed by provision of premises for the pursuit of the objective under consideration; the inputs of the system element are examined here. The right branch describes cases, in which the premises are given, but the system is disturbed during the execution of activities for the pursuit of the

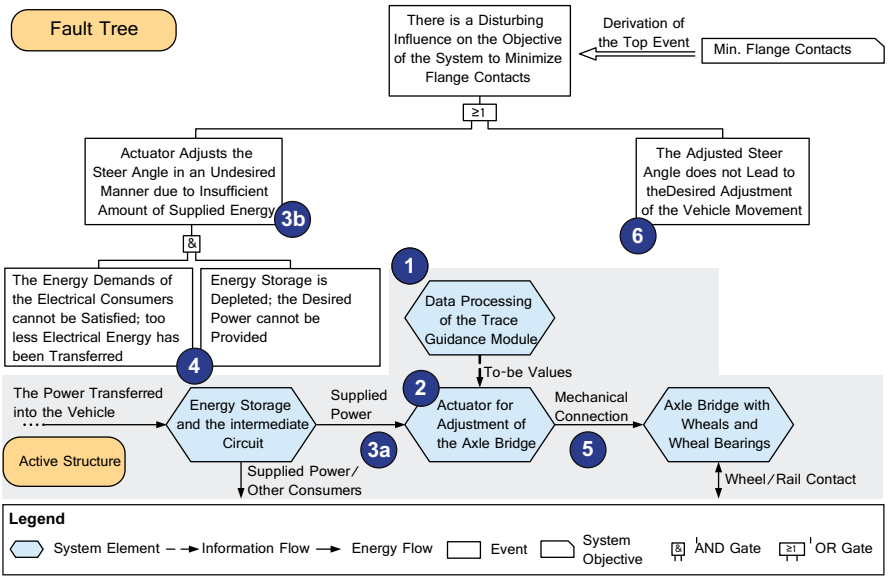


Fig. 4.14 Deductive analysis of disturbing influences of the RailCab system (cut-out) [45]

objective; here, mainly the outputs of the system element under consideration are examined.

Let us consider the objective "minimize flange contacts". The respective cut-out of the active structure for the RailCab system and the corresponding Fault Tree are shown in Fig. 4.14. From Phase 1 we know, that the objective originated from the system element "data processing of the trace guidance module" (Fig. 4.14 (1)). The "desired values" are sent to the system element "actuator for adjustment of the axle bridge" (2). We construct the left branch of the tree by following the aforementioned schema. The system element "actuator for adjustment of the axle bridge" receives power from the "energy storage and the intermediate circuit" (3a). Thus the event "actuator adjusts the steer angle in an undesired manner due to insufficient amount of supplied energy" (3b) is integrated in the left branch of the Fault Tree. This event is further refined in cooperation with the respective developers. In the course of this, two subordinated events are incorporated into the Fault Tree (4). We now proceed with the right branch of the tree. One of the outputs of the "actuator for adjustment of the axle bridge" is the flow "mechanical connection" (5). It is examined and the event "The adjusted steer angle does not lead to the desired adjustment of the vehicle movement" is integrated into the right branch of the Fault Tree (6). The newly incorporated event is then further refined and the procedure continues.

Phase 3 - inductive analysis of disturbing influences: Usually not all of the disturbing influences are found in Phase 2. Therefore an inductive analysis similar to the FMEA is performed as well. An inductive analysis can be thought of as a "what-if" analysis [15]. The developer asks: What if this system element failed, what are the consequences, what are the possible causes etc.? The developer starts

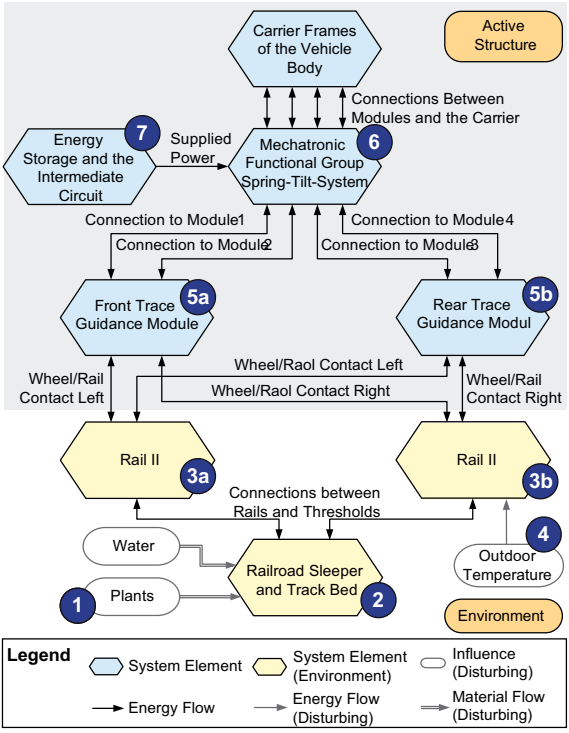


Fig. 4.15 Inductive analysis of disturbing influences of the RailCab (cut-out) [45]

with the influences from the environment of the system and investigates how they propagate through the active structure of the system. Fault Trees from Phase 2 are extended with regard to the newly gathered information.

Figure 4.15 shows a cut-out of the partial models environment and active structure of the RailCab. The track bed changes due to growing plants (Fig. 4.15 (1)). As a result the position of the sleeper in the track changes (2), as well as the position of the rails (3a and 3b). In combination with environmental influences (4) deformations of the rail occur. These lead to movements of the vehicle body (5a and 5b) which are transferred to the Active Suspension Module through mechanical connections (6). In order to dampen vibrations and tilt the vehicle body during curves, more energy has to be provided to the active suspension (7). The increased energy demands of the Active Suspension Module can eventually lead to the depletion of the stored energy. The newly gathered information is incorporated into the Fault Trees, which have been constructed in Phase 2.

Phase 4 - refinement of the principle solution: The Fault Trees are evaluated and the principle solution is extended accordingly. In particular, the conflicts between objectives of the system are identified. Objectives of the system which do not exhibit any conflicts with other objectives are marked as not relevant for the self-optimization and removed from the partial model system of objectives. The description of the identified conflicts is incorporated into the system of objectives.

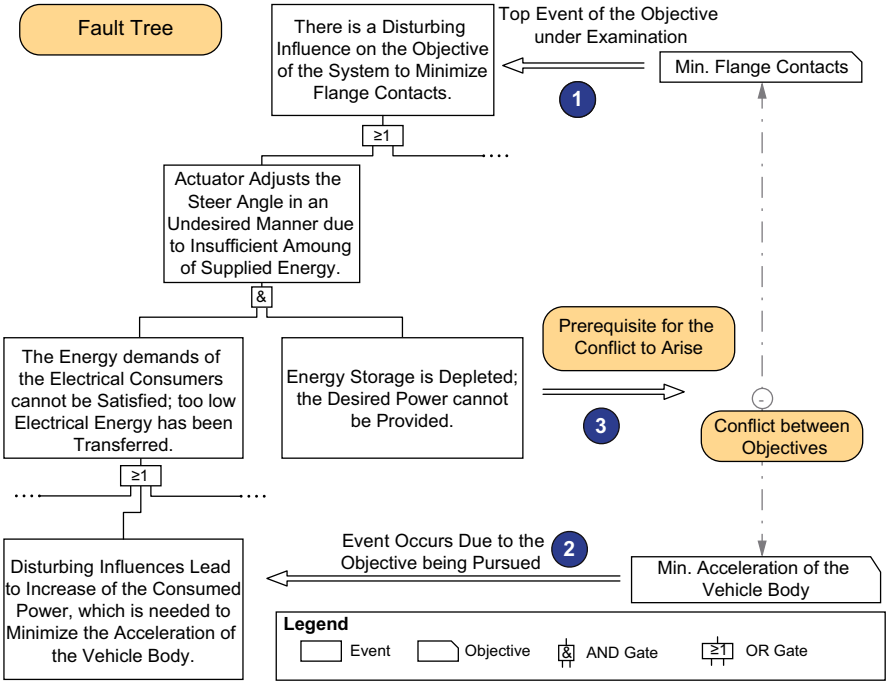


Fig. 4.16 Identification of conflicts between objectives based on the Fault Tree (cut-out) [45]

The result of this phase is the extended system of objectives, showing all objectives relevant for the self-optimization and the possible conflicts between them.

A cut-out of the refined Fault Tree for the RailCab is shown in Fig. 4.16. The event "There is a disturbing influence on the objective of the system to minimize flange contacts" (1) was derived from the objective "minimize flange contacts" in Phase 2. Starting with this event the Fault Tree is further examined. The event "Disturbing influences lead to increase of the consumed power, which is needed to minimize the acceleration of the vehicle body" (2) is found. It occurs if the objective "minimize acceleration of the vehicle body" is being pursued. According to the Fault Tree there is a potential conflict between both objectives. The prerequisite for the occurrence of conflict is also derived from the Fault Tree (3). Such information is very relevant for the further realization of the self-optimization process.

Phases 2 - 4 are conducted for each objective identified in Phase 1. Possible conflicts as well as prerequisites for their occurrence are recorded in the partial model system of objectives (for an example see Fig. 4.8 in Sect. 4.1). In particular, objectives, which are in conflict with the objective "maximize reliability", and the respective prerequisites are identified. The gathered information forms a basis for the further improvement and extension of the principle solution. In particular, new measuring system elements and corresponding information flows have to be incorporated into the active structure. Furthermore, activities for gathering information

about disturbing influences and conflicts between objectives as well as for recognizing and mitigating these are integrated into the partial model behavior–activities. In particular, condition monitoring and performance assessment are implemented [37, 51]. Altogether, the system under development is made more reliable. The process of the improvement of the principle solution can be supported by solution patterns, which we will explain in Sect. 4.5.

4.5 Design Framework for the Integration of Cognitive Functions Based on Solution Patterns

Harald Anacker, Roman Dumitrescu, and Jürgen Gausemeier

The following approach by Dumitrescu (2011) defines a design framework for the development of cognitive functions based on solution pattern during early design phases – conceptual design or system design. By using this framework developers can systematically integrate those functions within the principle solution of a self-optimizing system. The result is the early specification of the information processing within the architecture of the Operator-Controller-Module. Later on, during the concretization, this enables the final implementation of the cognitive functions [10]. The basis for this approach are the research fields of mechatronics, that cover the technical demand, and cognitive science, from which many results about intelligent behavior and structures have to be considered. The design framework itself covers four basic steps, which will be introduced in the following four sections.

The core of the framework is a procedure model (4.17). It gives an overview of the steps that have to be carried out during the conceptual design to integrate cognitive functions in the principle solution and the concrete results of the correlative steps.

Moreover, the procedure model defines what methods or tools should be used during which step. The procedure model connects all the parts of the design systematics in a logical sequence for their application. The different phases will be explained in the following subsections.

4.5.1 Systems Analysis

The objective of the first phase is to create a statement if the significant improvement of system performance can be expected through the integration of cognitive functions. Consequently, this phase clarifies the requirements for using cognition and its respective methods. In order to detect the potential use of cognitive functions, the *method for objective function analysis* is used. With this method, the necessity of using active paradigms for self-optimization can be established. This is divided into four successive steps:

1. **Identification of relevant influential specifications:** In the first phase, if not already done, the objectives of the system are identified (e.g. low energy

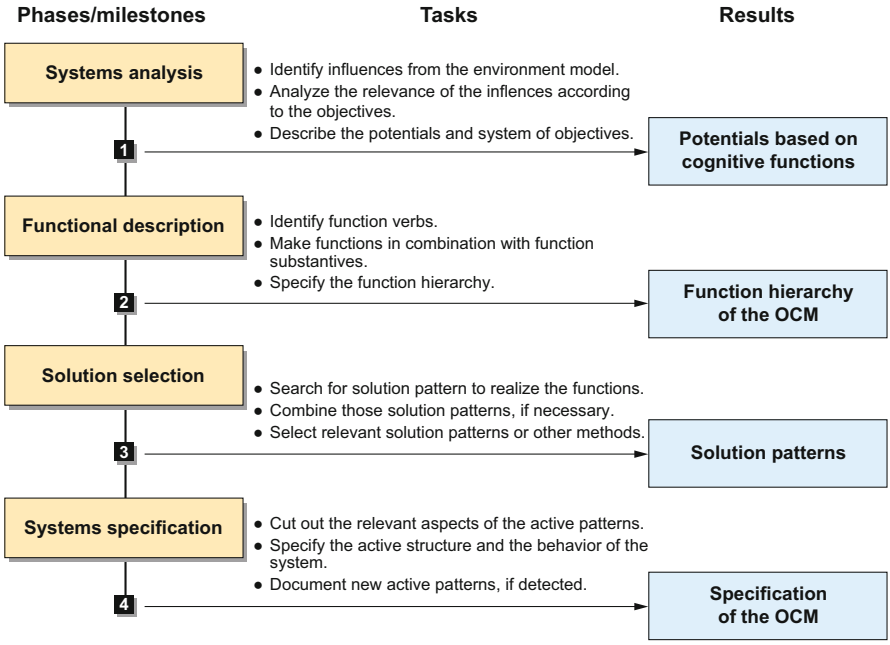


Fig. 4.17 Procedure model for the specification of the OCM with cognitive functions [10]

consumption or the level of safety of a vehicle.) In addition, a context analysis has to determine the influences that in principle the objectives can influence. It is important that potential forms of the influences are determined.

2. **Illustration of the effect of the influential specifications:** The next step, the influential specifications and the objectives of the system are placed in relationship to each other. It is interesting to note how the influential specifications affect the objective priority. The objective priority characterizes the importance of a objective under the given influence. Accordingly, an increase in the objective priority should be accompanied by an increase in the objective weighing. Here the **Objective Priority Matrix** lists the influence like its specifications in rows and in the columns the rows. The matrix can then change the priority objective due to the influential specifications (strong decrease, decrease, no effect, increase, and strong increase). 4.17 shows a portion of the objective priority matrix of the flexible road vehicle Chameleon 2.3.
3. **Educational relevant situations:** In the third phase non-relevant objectives and influential specifications are stricken from the Objective Priority Matrix of operation. It is evident that a column (irrelevant objective) or a row (irrelevant influential specification) is evaluated neutrally with a "0". Afterwards situations are formed from the leftover influential specifications. Situations are consistent combinations of influences. Combinations of influential specifications that cannot happen in reality should be excluded.

Objective Priority Matrix **question: „How does the influence expression i (line) affect the priority of system objectives j (column)?“** scale: -- = strong reduction of the objective priority - = reduction of the objective priority 0 = no change of the objective priority + = inrease of the objective priority ++ = strong inrease of the objective priority		**systems-objectives**	low energy consumption	high driving saftey	high comfort	low tyre wear	compliance of driver request´s
influences	**expression**	**no.**	**1**	**2**	**3**	**4**	**5**
unevenness	available	**1**	0	+	+	0	0
	not available	**2**	0	0	0	0	0
driver mass	high	**3**	+	0	0	0	0
	low	**4**	0	0	0	0	0
steering strategy	override	**5**	0	+	0	0	0
	understeer	**6**	0	0	0	0	0
⋮							
energy consumption	high	**23**	+	0	0	0	0
	low	**24**	0	0	0	0	0
negative acceleration	high	**25**	0	0	0	-	0
	low	**26**	0	0	0	0	0

Fig. 4.18 Objective Priority Matrix (example: demonstrator Chameleon) [10]

4. **Evaluation of the situation-dependency of the objective:** In the final step, the objective priorities are counted from the Objective Priority Matrix for each situation and a situation-dependent priority of each line determined. The outcome of this is the degree of dependence of the objective. Thus, an objective of single priority in a given situation can stand on its own or distribute itself to one or many other objectives. If the objectives of situations across other situations prioritize differently, then this is the first criteria for the integration of cognitive functions. Furthermore, the proportion of shared objectives is of importance. With a high proportion of shared objectives this suggests that this objective is closely related to other objectives and its weight should not be from the outset determined and isolated by the developer.

4.5.2 Functional Description

In order to realize a self-optimization process, optimization systems have to perform information processing functions such as to communicate, to share knowledge or to extract information. These functions are known as **cognitive functions**. Even though there is no common accepted definition of cognition, there is a common sense that cognition intervenes between the perception and the behavior of a system in the way, that certain stimuli does not always result in the same reaction. Therefore, cognition can be characterized as the ability that does not only enable autonomy and adaptability, but also a more reliable, effective ,and viable system with regard

Table 4.2 Examples of basic information processing functions [10]

basic functions	specific functions for concretization
to acquire	to (call, update, ask, receive, measure)
to process	to (prepare, contain, divide, compensate, choose, filter, convert, delete, save, compare, evaluate)
to transfer	to (command, deactivate, activate, provide, set, inform, send, allocate, transmit)

Table 4.3 Examples of cognitive functions [10]

complex functions (abstract)	declaration - exemplary combination of basic functions
to analyze	A system element receives information or makes an explicit request. In addition, the information is analyzed based on existing and additionally requested information. In conclusion, the results are transmitted to one or more system elements. to call - to compare - to transmit
to classify	A system element receives information or makes an explicit request. The element compares the information with already existing information. Depending on the comparison; the information gets a new evaluation and classified. This classification is saved and transmitted to other system elements. to receive - to compare/prepare/save - to allocate

to its purpose. Strube (1996) distinguishes the following cognitive functions on a psychological level [54]: to observe, to recognize, to encode, to store, to remember, to think, to solve problems, to control motor function and to use language. Thus, cognitive functions are basically information processing functions which not only formalize new information, but also connect new information with existing internal information. Since cognitive functions process information – and this is the main assumption of cognitive science – they are calculation processes and can therefore also be implemented in technical systems.

To name the functions of technical systems different noun-verb-catalogs for mechanical engineering have been developed [34, 35, 43, 54]. Due to an outstanding overall catalog of information processing functions, we have developed one as the first step to integrate software specific aspects in the principle solution. According to the IPO-Model (Input-Processing-Output), we have distinguished between three basic functions: the functions of acquiring, processing and transferring information. In respect to several other functions were identified to concretize those basic functions. For example, the acquisition of information can be done in a passive (e.g. to receive information) or an active way (e.g. to retrieve information). All in all, 24 functions have been documented Table 4.2. Furthermore, other types of functions have occurred: functions, which were a combination of the basic functions (e.g. to analyze or to classify). These functions are named complex cognitive functions of information processing Table 4.3.

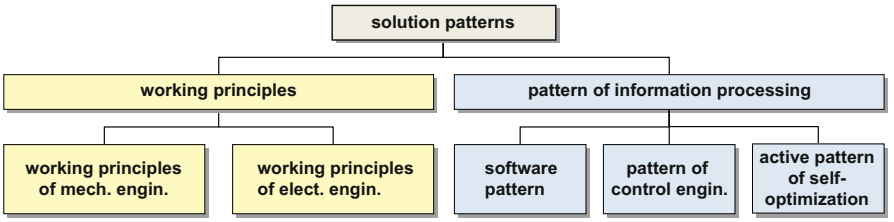


Fig. 4.19 Classification of solution patterns [10]

4.5.3 Solution Selection

In this phase, the potential solutions for the functions are identified. In order to achieve this, the sub-functions must be allocated at the lowest level of the function hierarchy of partial solutions and successively fill the higher-level functions up until the complete function. The prerequisite for finding solutions for the integration of cognitive functions is an adequate representation. For this purpose, recurring solutions in the form of **solution patterns** are prepared.

In the architecture the idea was formulated in this context that the core of a solution can be described as a pattern for a specific problem that can be drawn upon in this analogous problem situations [1]; recurring problems are not to be solved from the ground up every time. This is valid in an analogous way in mechanical and electrical engineering as well as in control and software engineering and also for the conception of intelligent behavior of self-optimizing systems. In this respect, the structuring of the solution patterns during the development processes presents an important foundation for the development of these systems.

Generally a pattern describes a recurrent problem in a definite context and the core for this problem, i.e. the structure and behavior of the characteristic elements of possible solutions in a generalized form. Based on this assumption, 4.19 presents a depicted classification of the solution patterns.

We differentiate solution patterns that contact physical effects and patterns that serve exclusively for processing information. The construction doctrine of mechanical and electrical engineering identifies the first group as working principles [43]. Working principles create the connections between the physical effect, material, and geometric structure. An example for this is the working principles of the electric motor that can be used as a solution of the function to convert electrical energy into mechanical energy.

This similarly applies for the area of information processing. In the area of software engineering, software patterns are utilized in order to save cooperating objects and classes in case these solve a general design problem. The patterns contain information on how they can be used and implemented in new situations. The solution description is made up of a structure and the partial behavior of each structural element. The domain of control engineering describes solution patterns on how a control loop is created, influenced or how the size of a path is measured or observed. Patterns of control engineering are primarily assigned to patterns of

information processing. In doing so it must be observed that the patterns of control engineering can be concretized as patterns of software technique or as working principles. Besides the mentioned specifications of solution patterns, in the context of self-optimizing systems, working patterns for self-optimization come into play [50]. They are used for the implementation of self-optimization processes. Working patterns for self-optimization fulfill functions for self-optimization like autonomous planning, cooperating, acting and learning. The spectrum of the working patterns for self-optimization envelops the complete self-optimization process (1. Analysis of the actual situation, 2. Designation of system objectives and 3. Adaption of the System Behavior) [40].

As already mentioned the early design phases of self-optimizing systems require an effective cooperation and communication between all developers involved. To come up with such requirements we developed a uniform specification of solution patterns that is similar for all the disciplines involved. We structured the specification in six aspects with respect to the categories according to Alexander (1977), which is presented in Fig. 4.20 and will be explained in the following:

Characteristics: This Aspect describes the characteristics of the pattern. Because of the significant differences between the characteristics of the basic system and the information processing we will distinguish two subcategories of this Aspect. Examples for relevant characteristics of the basic system are usage conditions, geometry or material. In this context it is useful to note the environmental conditions, where the physical system elements can be used. The geometrical aspect implies the description of the approximate dimensions of the elements. Material characteristics in particular have to be specified for a compatible combination of solution patterns for the basic system. This aspect refers to similarities of the system elements and the working medium like fluid in hydraulic systems.

Similar to the patterns for the basic system, the information processing of the OCM is linked to some kind of usage conditions. More examples are processing speed or the type of calculation. So by generating a new SP for the information processing, the developer has to differentiate between hard real-time and soft real-time. The kind of calculations is based on different information methods. Generally we distinguish between mathematic relations and software code that represent an application's flow of commands.

Functions: This Aspect contains all those functions that the SP can realize. Thus this aspect expresses the problem description. The hierarchical structure facilitates the developers to assign a suitable SP for the underlying problem.

Active Structure: The aspect active structure is the core of the solution-description. This Aspect specifies which system elements are necessary in order to implement the SP and how those system elements are interrelated. To support the developer to handle the complexity of a self-optimizing system we designed a general structure according to the OCM. This structure shows the basic elements of a self-optimizing system and has to be modified and concretized for each problem. This general structure also clarifies the interface between all involved disciplines on the different levels of the OCM.

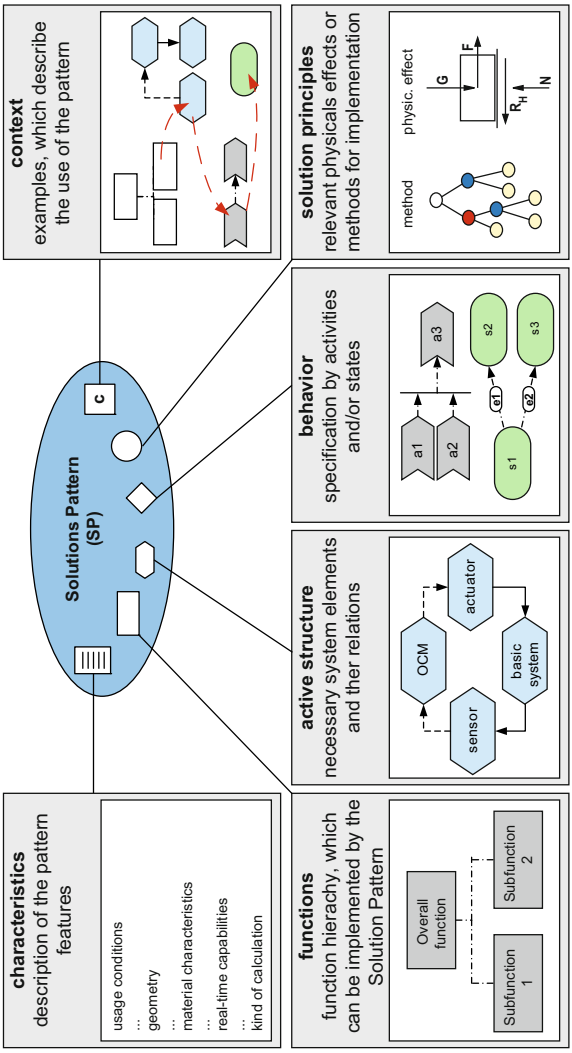


Fig. 4.20 Uniform specification of solutipPatterns [10]

Behavior: This Aspect describes the behavior of the system elements. The aspect behavior is split into two sub-aspects. Behavior–activity describes the activities that are performed by the active elements during operations. So this sub-aspect has to be developed for patterns of the information processing. However, the behavior–state aspect models the possible sequences of states and states transitions of all system elements of the active structure. Thus this aspect has to be modified for each pattern to design self-optimizing systems.

Implementation: In the course of engineering, the implementation of solutions is generally based on certain method. However, an explicit definition that focuses the implementation of physical problems as well as information processing does not

exist. According to the definition by Sauer (2006) [48], methods describe sequences of physical, chemical, biological or information processing application flows that are necessary to realize the defining functionality. So a method concretizes the abstractly formulated process, which describes the transformation of the operand from the initial state to the final state.

Therefore patterns of the basic system are generally based on physical effects. The Controller algorithms are based on mathematic relations like elementary transfer elements (e.g. P-element). However, the SPs for the RO are based on software algorithms. These are implemented in the CO model-based and behavior-based methods.

Context: A solution pattern is generally attached to a specific context, so this Aspect completes the uniform specification. Examples, which clarify the successful use of the pattern, have to be declared. The core of each pattern is the description of the underlying problem and the related solution. Therefore the aspects functions, active structure and behavior have to be modified for each example.

The presented specification is similar for all different problems within the domain-spanning conceptual design. Because of the increasing complexity of self-optimizing systems the following question is asked: Is it possible and advantageous to categorize solution patterns?

In order to design self-optimizing systems, we propose a categorization according to the generic composition of mechatronic systems that adjusts to the subdivision of information processing into several hierarchical levels, see Fig. 4.21. This division is well-grounded in cognitive science and enables the illustration of information processes which implement intelligence (cf. Chap. 1).

Tool Support: The structure of the above presented solution patterns allow the externalization and documentation of reusable solution knowledge. However, an efficient use of these patterns requires an appropriate computer support. Developers need a way to create new solution patterns, to store them in a repository, as well as an opportunity to integrate existing solution patterns in their current development process. A need for some kind of database, in which solution patterns and thus the knowledge of the experts can be stored, is apparent. Therefore, we developed a knowledge base for the systematic management of solution patterns, called "Solution Pattern Knowledge Base". The basic functionality is shown Fig. 4.22.

The solution pattern knowledge base is a central repository for all developers. Apparently, it is necessary to extend the functional range of a standard database, which only stores the information. In a standard database all users need detailed knowledge about the structure of the information and possible search methods. Instead, the Knowledge Base has to support in such a manner, that developers from different domains are able to recognize an appropriate solution pattern.

An information system that implements the pattern repository must support the collaboration of the solution patterns developers, which are e.g. experts within the field of artificial intelligence or mathematical optimization, and the engineers. Our aim is the storage of all required information in one single repository to enable the developers an access to domain-spanning solutions during the system design. The Knowledge Base is composed by several parts: database, which stores solution

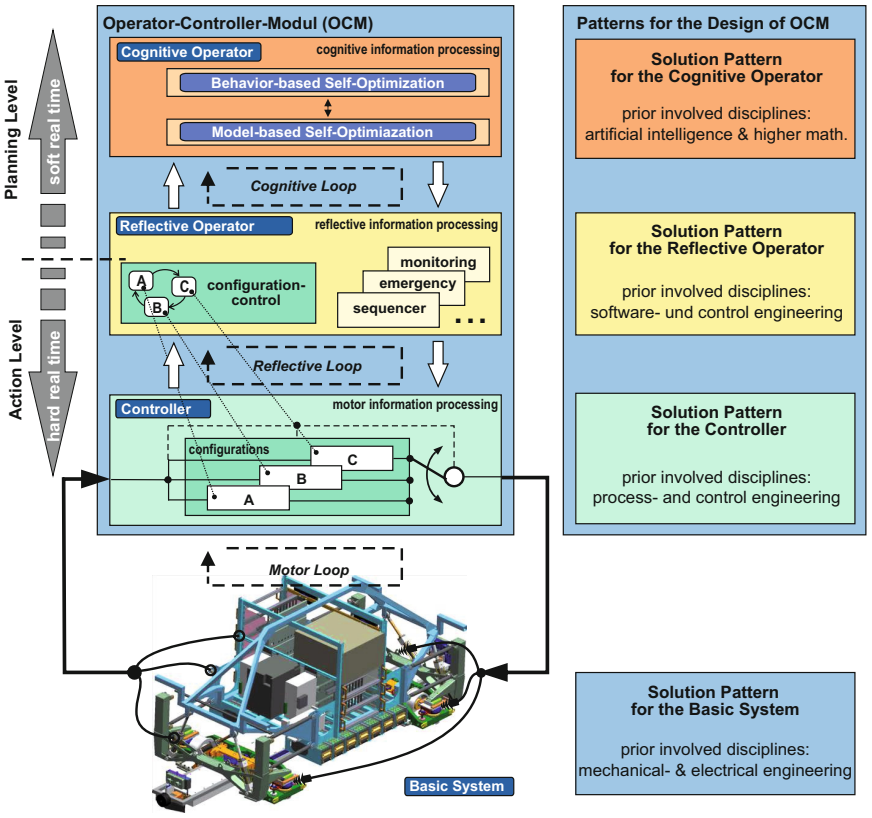


Fig. 4.21 Classification of solution patterns according to the Operator-Controller-Module [10]

patterns and a catalog of functions; ontologies for the consideration of the semantic; inference-rules to combine solution patterns.

4.5.4 Systems Specification

In order to support the engineers during the conceptual design of self-optimizing systems, especially the design of the information processing, we developed a design template for the active structure. Therefore it was necessary to identify the interrelationship between the cognitive functions and elements of the technical systems. In this context, the scientific cognitive view is transferred to the technically oriented OCM-architecture in the following. Figure 4.23 presents the transformation of the point of view from cognitive science to the technical OCM-architecture. It shows a general active structure that can be concretized by different flows. This general

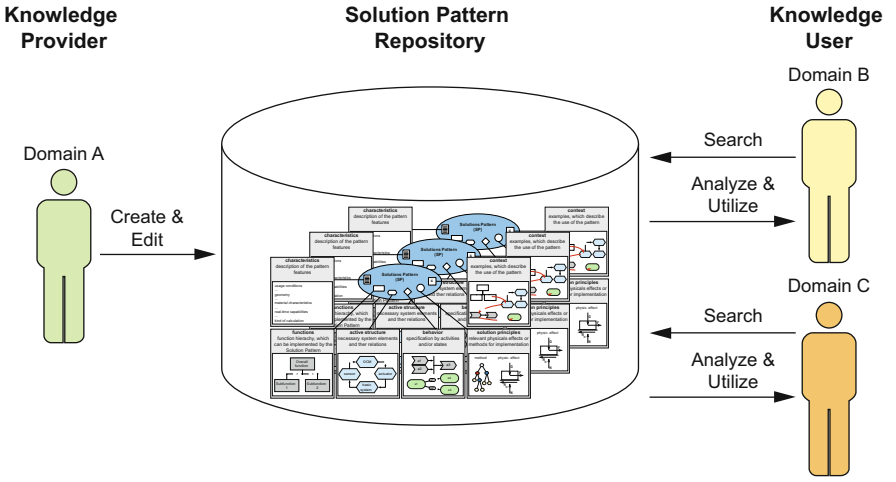


Fig. 4.22 Basic concept of a knowledge base for the domain-spanning reuse of solution patterns (in accordance to [10])

structure also clarifies the interface between all involved disciplines of the different levels of the OCM.

Basic System: The physical system elements are located in the basic system. This can be subdivided into the following system elements: passive basic structure, hardware for data processing and energy supply. All energy flows of the basic system represent the attachment of single elements at the passive basic structure. The actuating and sensor elements are the interface for data processing.

Controller: The Controller (CO) realizes the non-cognitive control. The most important task of the Controller is to improve the system performance. This is implemented by adjusting disturbance values, which influence the basic system. From a process control point of view, the actuator elements are an integrative part of the setting device. It fulfills functions like e.g. "to balance system deviation". The Controller realizes a rigid interface between the actuating and sensor elements, whose action flow is known as a motoric loop

Reflective Operator: In the Reflective Operator (RO) the associative regulation, such as classical or operant conditioning, can be implemented by its corresponding algorithms. The RO can thus be used for learning control superimposed by the CO. The processing speed through the RO is subdivided in soft and hard real-time. The studies of existing systems have shown that associative functions run in soft real-time. Basically, the majority of the calculations in the RO run under hard real-time conditions. Its primary task is to generate reference values based on sensory input for regulation in the Controller. The data processing of the OCM has to split into several system elements that fulfill different functions (e.g. configuration management, monitoring or corrective element). Furthermore, a communication module is necessary to share important data or information with external systems.

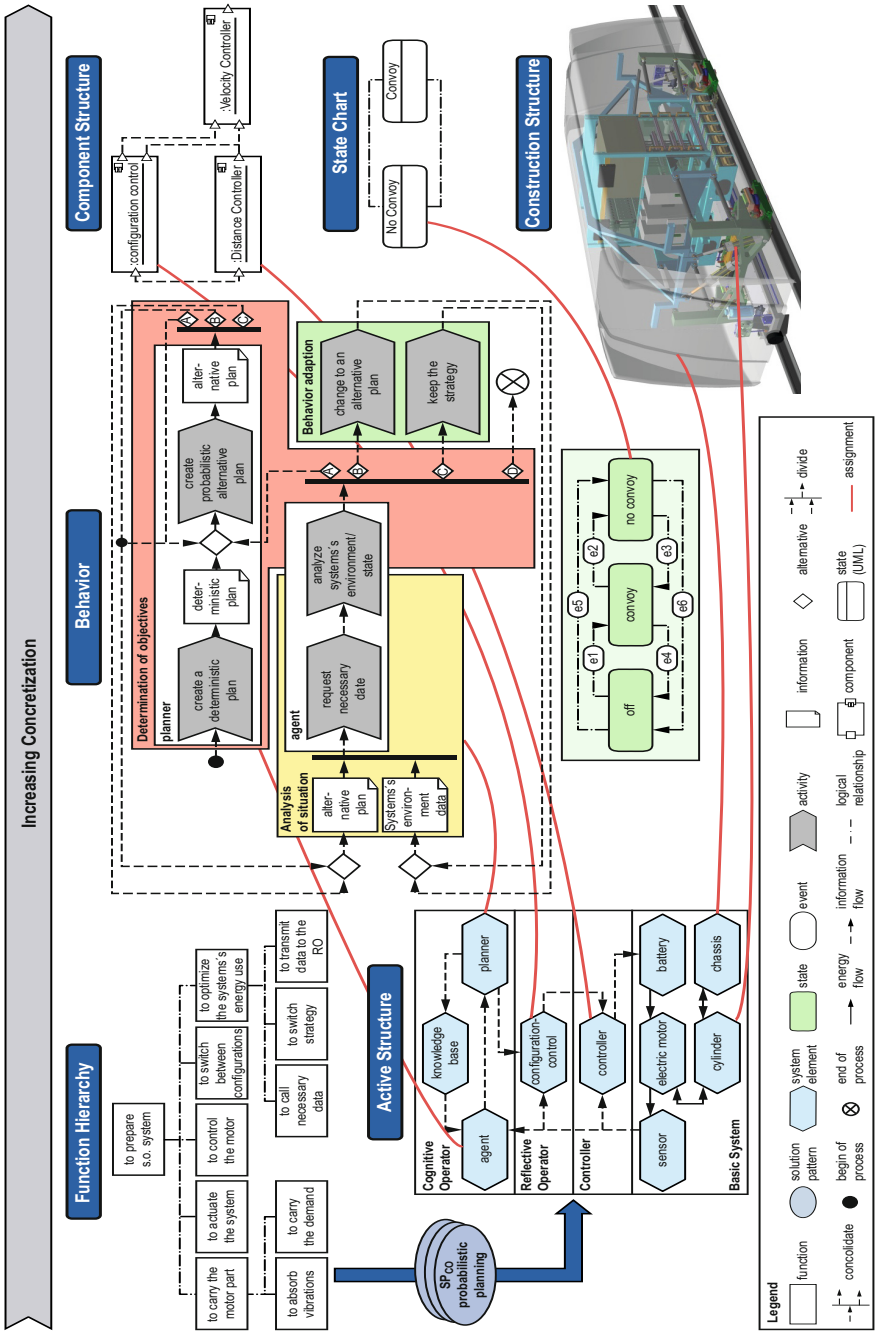


Fig. 4.23 Generic structure of intelligent technical systems based on the OCM-structure [10]

Cognitive Operator: The cognitive regulation is realized in the Cognitive Operator (CO). The calculation of the mathematical problems can run in offline or online mode. The system elements that are involved in the implementation of calculation methods are situation analysis, system of objectives and adapting system's behavior. The needed information to optimize are identified, received, tested and classified by the situation analysis. The keyelement of an autonomous intelligent system is a knowledgebase that stores the relevant information and generates internal knowledge. The system of objectives is the internal target system that plans the expected system behavior in certain situations. The results are transmitted to the element "adapting system's behavior". This system element does not directly access the regulation or the basic system with the appropriate actuators. Rather it adjusts the planned and optimized strategy in the RO. This coupling between RO and CO is called cognitive loop.

Conceptual Design of Self-optimizing Systems with Solution Patterns Exemplified by Probabilistic Planning

According to Fig. 4.24 the use of patterns to develop self-optimizing systems starts on the left with the successful implementation of eligible solution patterns for specific sub-functions. In the next step the developer has to feed the aspects of the active structure and behavior to specify the complete system. Therefore it is generally necessary to modify these aspects of the basic pattern to achieve the different fundamental problem. The developed principle solution is the second intermediate result. At the beginning of concretization the aspects have to be transformed in domain-specific terminologies. So the process ends in an early specification of the information processing as a component structure and a state chart. Simultaneously the construction structure, based on the principle solution of the basic system, is developed.

In the following we will explain the solution pattern "**Probabilistic Planning**". The topic of the patterns, which is shown in Fig. 4.25, is the forward planning of possible situations when considering insecurity. The core of the planning is a decision tree consisting of an ideal path and several possible intersections, which lead to the same result. A condition is defined as the amount of the critical value of one or more state variables (conditional planning). The probabilistic planning contains the aforementioned limited planning and the execution monitoring. Furthermore, a new planning can be considered. The basic idea is to use the conditional planning as a standard practice. The new planning however represents a backup level for unpredictable and implausible events. In this case, more intersections have to be added to the decision tree. The requirements for the planning speed become more diverse depending on the situation, in which the new planning is necessary. Because of the fact, that the planning quality decreases if an alternative solution is found, the use of the backup level should be avoided as much as possible. In the following, the different aspects of the solution pattern "Probabilistic Planning" will be explained. The uniform specification characteristics are:

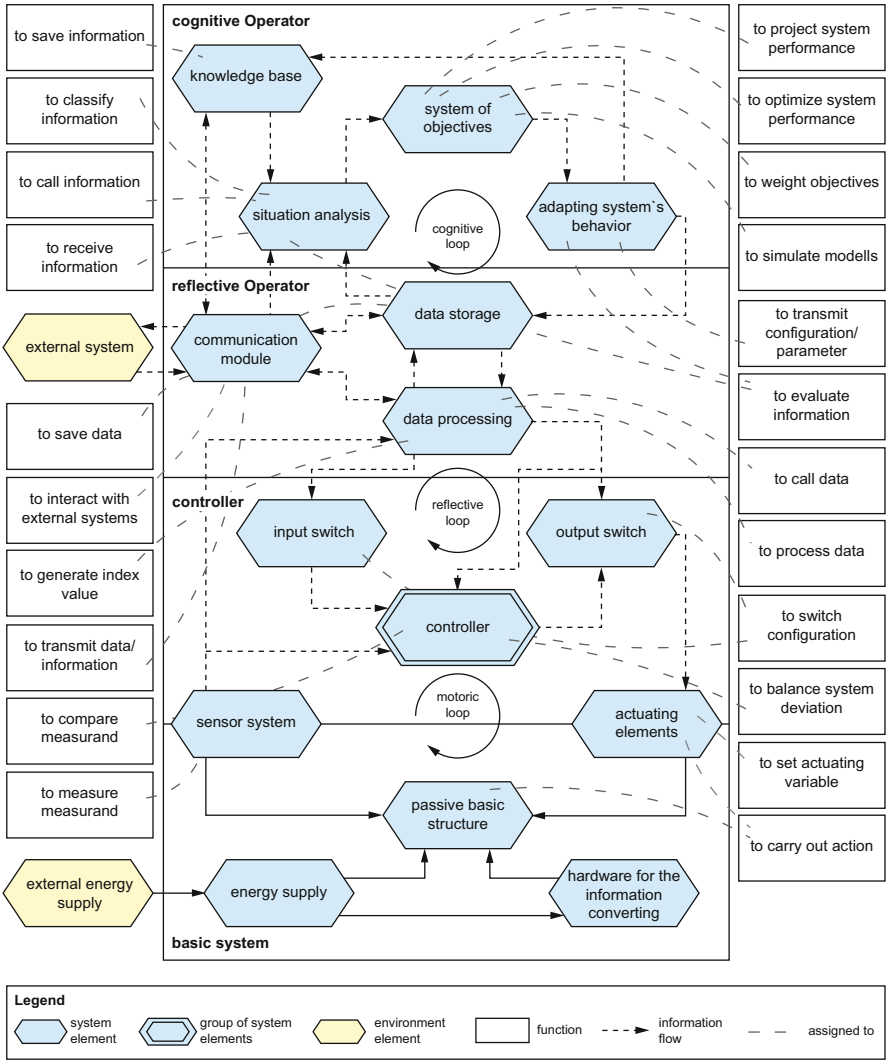


Fig. 4.24 Design of self-optimizing systems with solution patterns [10]

Working conditions: The probabilistic planning is designed for mechatronic systems that don't always start from the same position and constantly change their positions, e.g. vehicle mobile robotics.

Mode of calculation: The processed values are discrete. This applies to both the conditional monitoring as well as for the execution monitoring.

Real time capability: The requirements for the planning time are diverse. The entry of an unexpected event has an effect on the quality of the new planning. The creation of the decision tree including the necessary intersections happens in soft real-time with a significant variation of the time limits.

Modeling: A holistic physical representation of the system‘s behavior is not necessary.

Entity: The system element‘s planner and agent work in a collective way.

Modeling language: The specification of this pattern is based on the detailed terminology PDDL (Planning Domain Definition Language).

A functional description of the SP probabilistic planning divides the overall function "evaluate probabilistic" into three sub-functions to analyze the situation, to determine objectives and to adapt behavior. The analysis of the situation calls all essential information of the knowledge base and preprocessed data of the Reflective Operator. The sub function to determine objectives is to predict using probabilistic planning and to compare situations in one step in the plan. The adaption of the behavior allocates a configuration in the next step in the plan. For each subsystem, one configuration exists. The configuration is then sent to the Reflective Operator.

Active Structure: The two central elements of the solution pattern "Probabilistic planning" are the agent and the planner. The tasks of the agent cannot be clearly assigned to a phase of the self-optimization process. The agent’s participation is significant in the analysis of the situation based on the analysis of the environment as well as the analysis of the system state. The agent is active in the objective determination because the agent can evoke a new plan through his evaluation of a corresponding step in the plan. The planner on the contrary is assigned to determine objectives. The necessary information for the planner – which normally relates to models – resides in the knowledge base. With the help of this information, a planner can determine the probability distribution for all the discrete system states. The information is either generated in real-time or saved externally in the knowledge base. Additional data that is necessary is provided by the data call system element. It receives data from the data storage/memory of the RO and executes the system analysis along with the agent. The pattern group is complemented through the behavior adjustment.

Behavior: Directly after the beginning of the process, a deterministic plan is triggered. This occurs offline, because the system is incapable of action without an established strategy. The necessary decision tree contains a finite number of alternative branches that are created in a repeating loop. After the completion of the loop, it is verified if the plan is mature enough so that the system can begin the operation. At this point three deciding criteria are differentiated. If the conditions for the execution of an operation are not fulfilled, other branches are created. This way the normal as well as the new planning can be executed. However, if the condition is fulfilled, the system can begin the targeted aim. If the plan is not completed at this point the planning is resumed parallel to the executing system.

An essential task of the agent is to compare the next possible controlling point in the plan with the current environmental condition as well as with the system state. An evaluation is carried out after an analysis of the actual situation with the four deciding factors. If an unexpected event occurs that the existing branches do not consider the agent initializes a new plan through the planner. If however an alternative branch already exists for the event, the subsequent behavior adjustment will react correspondingly and an alternative will be set up accordingly to the configuration. If

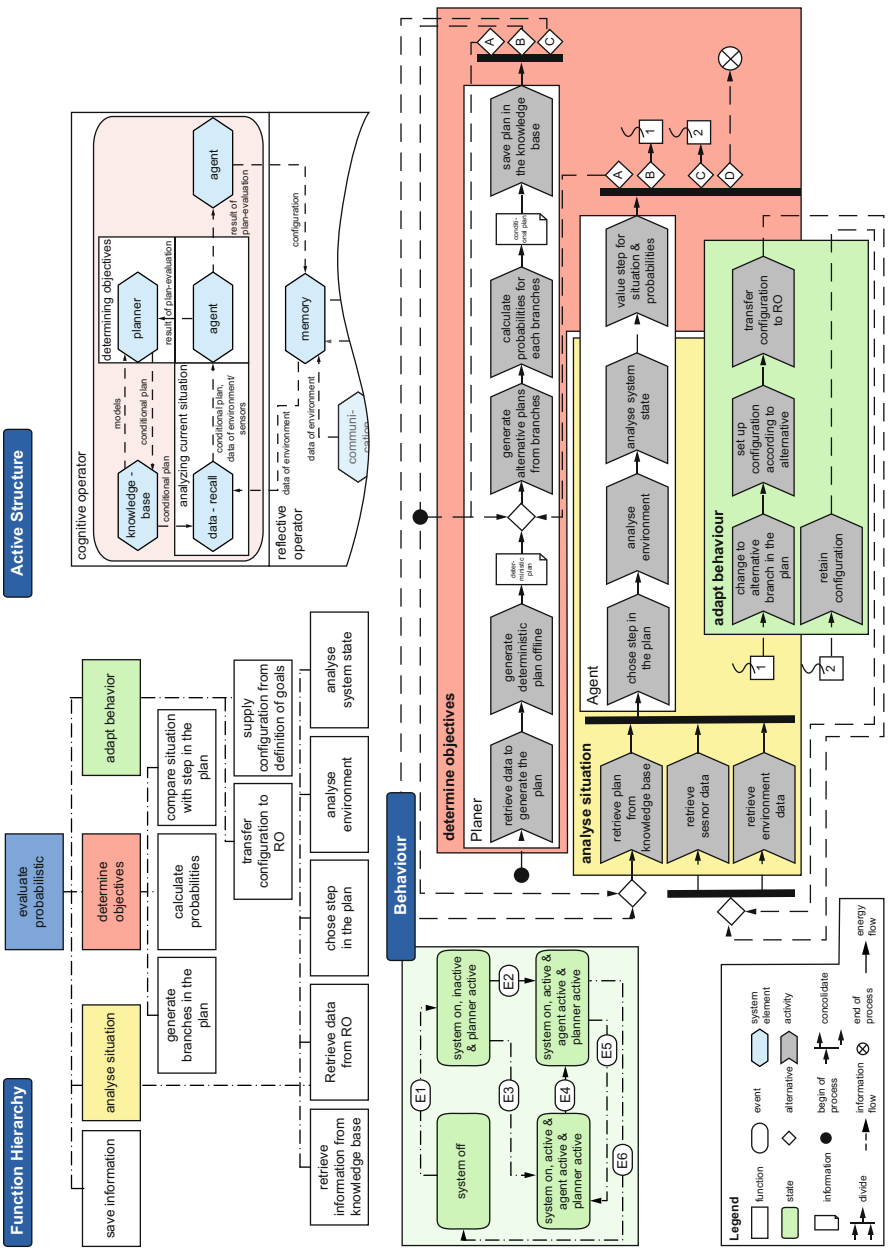


Fig. 4.25 Main aspects of the solution pattern "Probabilistic Planning" [10]

the evaluation yields that the complete behavior proceeds according to plan, a check is performed to clarify if the targeted objective has already been reached or not. If the objective has been reached, the system is turned off. It is however necessary to finalize further steps in the plan to communicate to the behavior adaptation that the configuration should stay the same. The whole process runs recurrently until the objective of the plan is reached.

4.6 Product Structuring for Self-optimizing Systems

Rafal Dorociak and Jürgen Gausemeier

Another important method that supports the creation of the principle solution is the product structuring. Product structuring is an important mean to handle the complexity of a technical system [38]. The aim is to identify modules that form logical and functional units, which can be developed, tested, maintained and, if necessary, be exchanged autonomously by different teams. Thus, the product structure affects the whole product life-cycle.

Before we introduce the method by Steffen (2006) itself, two basic concepts have to be explained [53]. These are 1) the basic types of a development task with regard to product structuring and 2) design rules for product structuring.

Basic types of development task: In general, the product structure can be either modular, integral or a combination of both. Which product structure mechatronic systems and especially self-optimizing systems have, depends on a number of factors. These factors are, in particular, requirements on the product, on the product program and on the product development process. The analysis of several development tasks and their respective requirements has shown, that there is a number of criteria, according to which a development task can be described [27]. These criteria can be divided into three groups:

- **criteria with regard to the product:** These are: the size of the system, installation space, weight, performance data, recyclability, quality/reliability, availability, expandability and reconfigurability.
- **criteria with regard to the product program:** These are: number of market segments, planned product generations, quality of differentiation and variance of costs.
- **criteria with regard to the product development process:** These are: development effort, depth of the development and time of delivery.

Another result of the analysis of the development tasks is that, using the consistency analysis [21], they can be clustered in nine consistent combinations, which we call profiles [27]. As a consequence there are nine basic types of development tasks. As shown in Fig. 4.26 these are: 1) miniaturized product, 2) cost optimized mass product, 3) performance optimized single product, 4) complex miniaturized system, 5) system with numerous variants, 6) complex system with specialized modules, 7) mechatronic function module, 8) safety-intensive system and 9) reconfigurable system.

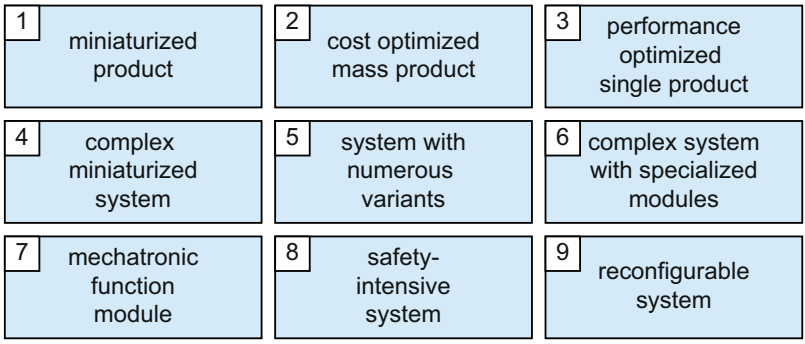


Fig. 4.26 Nine basic types of development tasks [27]

Figure 4.27 shows the profile of the basic type "9) reconfigurable system". An example of a system that complies to this basic type of development task is the RailCab. The profile is represented in a tabular form. The rows of the table are the aforementioned criteria. The cells describe the values of the respective criteria and whether they indicate a integral, modular or neutral type of the product structure. It is shown that reconfigurable systems are of relative high size and have high requirements when it comes to quality/reliability and availability as well as on expendability and reconfigurability. In addition, the breadth of the product program for such systems is high (i.e. a high number of markets has to be addressed and several product generations have to be planned). All in all, the values of the criteria indicate, that in most cases a modular product structure is recommended. Although, particular subsystems (modules) can certainly have an integral product structure, when some particular technical requirements have to be met.

Design rules for the product structuring: The basic types of a development tasks provide only a guideline. A direct adoption of the product structure for a whole class of systems is usually not possible (e.g. one product structure which can not be universally applied to all types of reconfigurable systems). Therefore, a number of design rules were defined, which support the developer by decision making in the conceptual design with regard to product structure relevant issues. In [27] 27 design rules were defined, which build eight categories. These are design rules for 1) product functionality, 2) disassembly/recyclability, 3) quality/reliability, 4) expandability, 5) standardization, 6) costs, 7) development and 8) manufacturing. The design rules address a number of properties of the product (disassembly/recyclability, expandability) and boundary conditions of the product development (standardization, manufacturing) into account. They are applied for the development of the partial models active structure, shape, etc. An example of a design rule is shown in Fig. 4.28. The main goal of this particular rule is to support reusability of the modules, which result from the product structuring.

For each basic type of development task a selection of such design rules has been defined [27]. Figure 4.26 shows a number of design rules, which are used for systems of the basic type "9) reconfigurable system". These are function fulfillment,

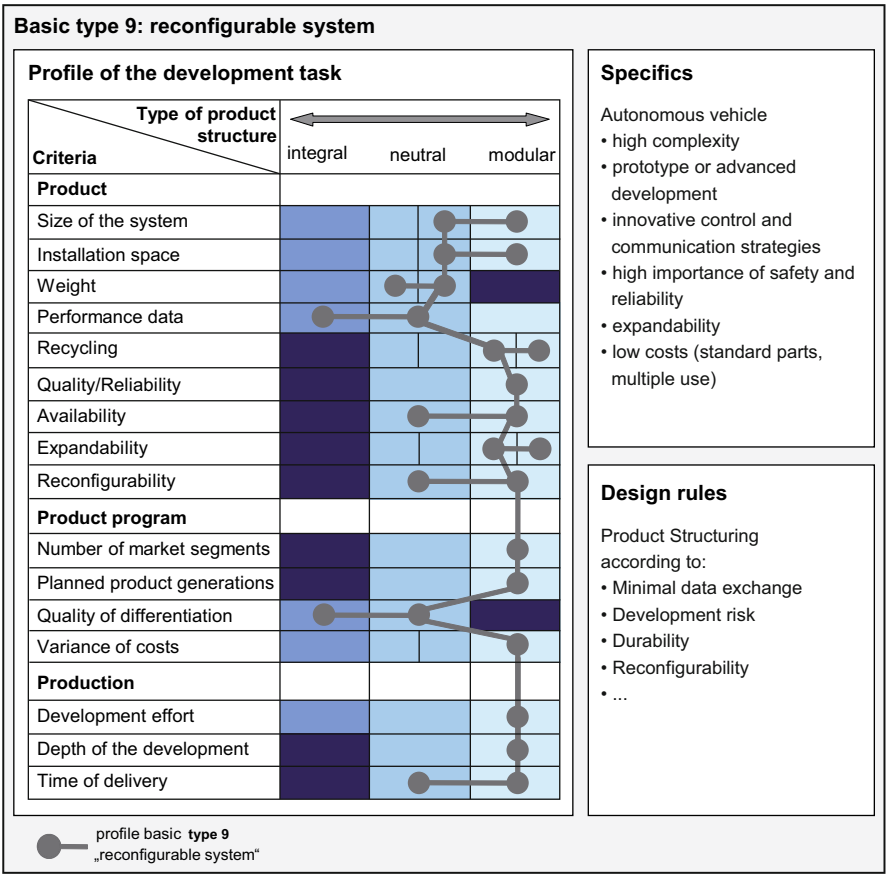


Fig. 4.27 Description of the characteristics of the basic development task "9) reconfigurable system" [27]

Fig. 4.28 Example of a design rule: "product structuring in terms of reusability" [53]

19	F G W	product structuring in terms of reusability
standardization		Sum up systems elements in a way that they can be used several times in the same product or other series. The aim is reduction of development costs and the realization of economies of scale.
• time between innovation • rate of reuse		
[Mül00, S. 31] [Woh98, S. 56]		*example: automotive – plattform concept of the VW group*

minimal data exchange, ability of testing and validation, durability, reconfigurability, user aspects, independence during further development, and development risk [27].

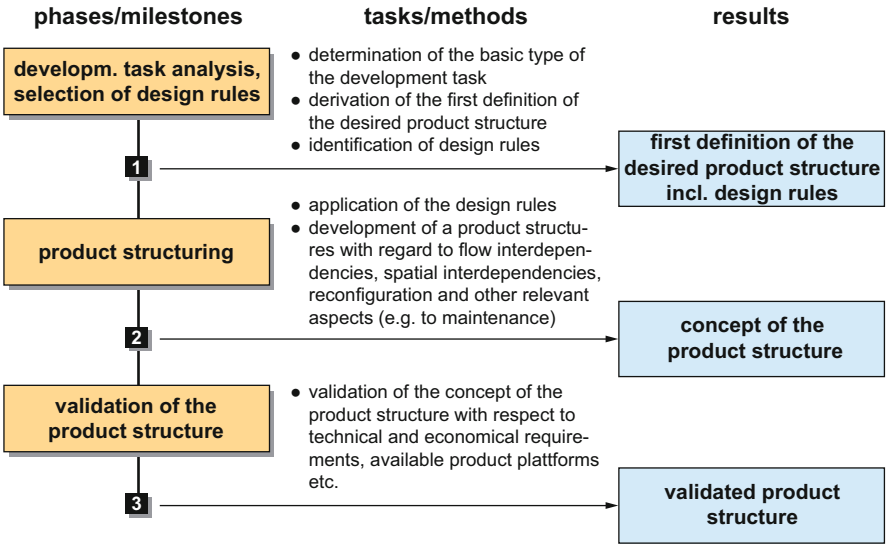


Fig. 4.29 The procedure model of the method for the product structuring of self-optimizing systems based on the principle solution [53]

The method for the product structuring of self-optimizing systems consists of the following three essential phases [27] (Fig. 4.29):

Phase 1 – development task analysis and selection of design rules: First, the underlying development task is analyzed. The goal is to define the desired product structure for the system of interest. For this purpose, requirements on the product, the product program and the product development process are gathered according to the criteria described before. These are compared then with the profiles of the nine basic types of development tasks. Based on this comparison, one particular basic type of development task is chosen, which corresponds best to the development task. The result of this phase is a first definition of the desired product structure based on the profile of the chosen basic type. It serves as an orientation aid or "light house" during the further development (it can be compared to the "ideal concept" by Altschuller (2000) [28]). As explained in Sect. 4.6 there is a number of design rules assigned to each basic type of development task. For the system of interest the design rules are used, which correspond to the chosen basic type of development task.

The described approach has been validated on the RailCab. The analysis of the development task has shown that the development of the RailCab focuses on the validation of the applied technologies and the newly developed information technological processes (self-optimization). Design and efficiency of the prototype are less important. It is important that the drive and the active spring technology are accessible and modifiable during later test cases. For the validation of new processes in the field of information technology, additional properties are important. These are: autonomy of the included modules and system elements, learning ability, high

control performance, and high safety requirements. In addition, the prototype has to be updatable. With respect to a later serial production, the mechanical components have to be reusable. Altogether the development task has the characteristics of basic development task "9) reconfigurable system" (Fig. 4.27). The corresponding design rules are: function fulfillment, minimal data exchange, ability of testing and validation, durability, reconfigurability, user aspects, independence during further development, and development risk.

Phase 2 – product structuring: The design rules selected in Phase 1 are applied in the course of the specification of the principle solution throughout the whole conceptual design, when design decisions with regard to product structuring are made. The design rules are used especially at the end of the conceptual design on the system level (before the beginning of the conceptual design on the subsystem level), as the subsystems (modules) of interest result from product structuring. In order to obtain modules, a number of established methods is used in combination. This is shown in Fig. 4.30. The starting point is the analysis of the specification of the principle solution (Fig. 4.30, (1)). Especially the partial models application scenarios, active structure and shape are concerned.

The information about system elements and their relationships (energy, material, information flows and spatial relationships) are extracted from the principle solution (mainly active structure and shape) and serve as input for the **Design Structure Matrix** (DSM) [14] (Fig. 4.30, (2)). With the resulting DSM the relationships between system elements are analyzed. The weighting of the different relationships is determined in accordance with the desired product structure and the chosen design rules. For product structuring, two particular views on the system are created. One focuses on its shape-oriented structure, the other one focuses on its function-oriented structure. Both views are then superimposed. Using DSM algorithms clusters of strongly dependent system elements are built. This happens semi-automatically. Some of the weights (values of the matrix) have to be modified manually in accordance with the underlying development task. The results are a product structure with regard to flow interdependencies and a product structure with regard to spatial interdependencies.

For self-optimizing systems, one aspect has to be particularly taken into account: self-optimizing systems have the ability to reconfigure. Hence, autonomous modules with disjoint functions and homogeneous interfaces have to be identified. For this purpose, the so called aggregation DSM and the **Reconfiguration Structure Matrix** (RSM) have been developed (Fig. 4.30, (3)). Both extend the DSM concept and use application scenarios of the system as input. For each application scenario a separate DSM is set up. Afterwards, the different DSMs are superimposed in two ways. Firstly, the aggregation of all DSMs is generated in a way, that all interrelationships are taken into account only once. The resulting aggregation DSM shows all possible connections within the system and allows the formulation of an adequate product structure. Secondly, the RSM is built. For this purpose, the frequency of the connections is taken into account by summing up the interrelations over all application scenarios. The resulting RSM allows the identification of system elements, which are only active in few application scenarios. This is a hint for reconfiguration potential. Such system elements could be integrated into autonomous additional modules.

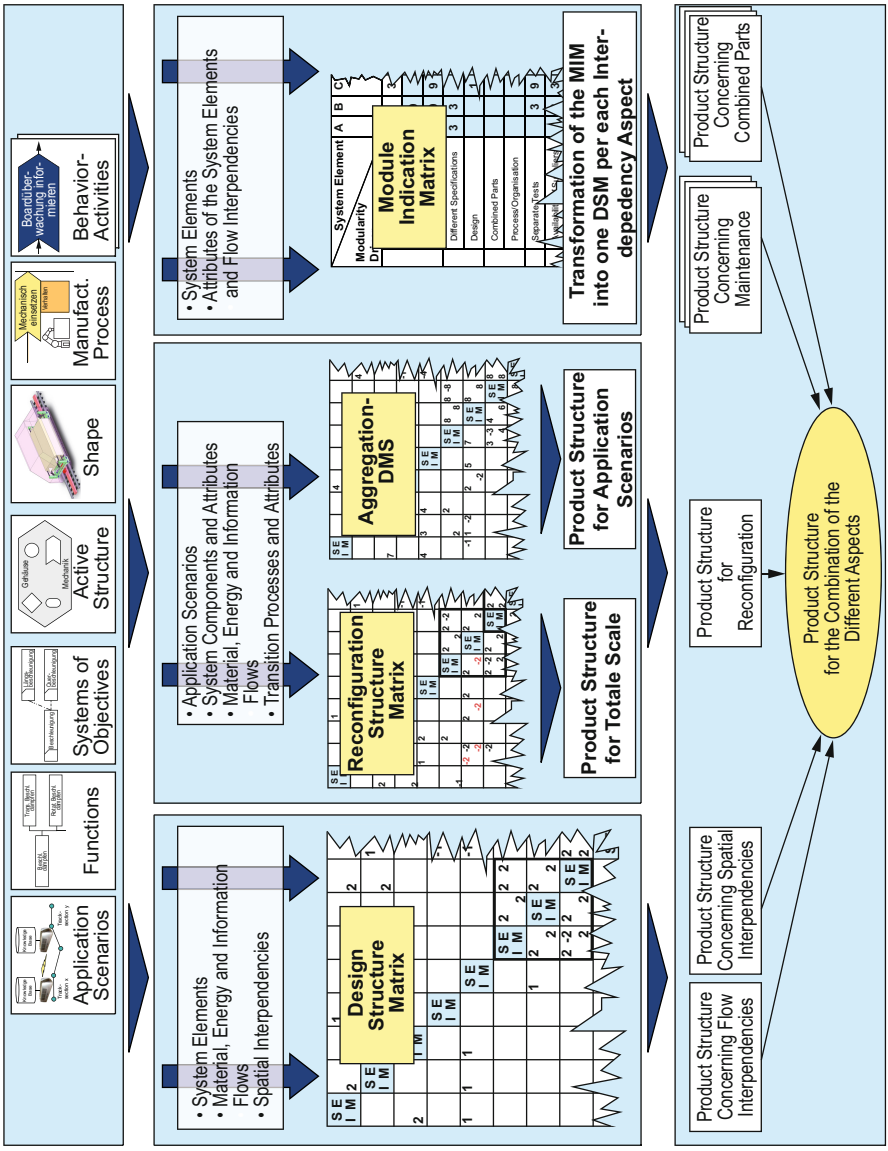


Fig. 4.30 Interaction of the partial models with DSM, RSM, Aggregation-DSM and MIM [27]

System elements which are active in all application scenarios are integrated into basic modules of the system. The result of the application of the aggregation DSM and RSM is a product structure with regard to reconfiguration.

With the aggregation DSM and the RSM only flow and spatial interdependencies between system elements were taken into account. For the definition of the product

structure, other aspects such as maintenance have to be considered, as well. A further refinement of the product structure is therefore necessary. We use the **Module Indication Matrix** (MIM) by Erixon (1998) [16] and its extension by Blackenfeld (1999) [39] for this purpose (Fig. 4.30, (4)). The MIM makes it possible to consider different properties of the system elements (e.g. maintenance intervals of a systems element) as well as partial model spanning cross-references (e.g. functions that a particular system elements concretizes). Which information is taken into account depends on the underlying development task and the selected design rules. The result of the analysis with a MIM with regard to the additional relevant aspects (e.g. with regard to maintenance).

Finally, the previously developed product structures are combined to the final product structure, which addresses all the aforementioned aspects (flow and spatial interdependences, reconfiguration, maintenance etc.) (Fig. 4.30, (5)) [27]. It forms the basis for planning of the activities in the further design and development phase. In particular, the resulting product structure integrates the two basic and mostly contradictory views of a shape- and function-oriented product structure. This is necessary, as both aspects are equally relevant for the development of mechatronic and especially self-optimizing systems.

During the conceptual design on the system level of the RailCab the design rules identified in Phase 1 were applied in an implicit way. The result is a first principle solution specified with the specification technique CONSENS presented in Sect. 4.1. At this stage the active structure for the RailCab consists of about 150 system elements. An explicit application of the design rules takes place at the beginning of the conceptual design on the subsystem level. For this purpose, flows (representing functional interdependencies) and spatial interdependencies are taken into account. Additionally the multiple usability of system elements is relevant. Two product structures are generated by using DSM. One with regard to flow interdependencies (function-oriented) and one with regard to spatial interdependencies (shape-oriented). Figure 4.31 shows these two structures (the function-oriented and the shape-oriented ones) and their relationship to the specification of the active structure for the RailCab. It is shown, that the two driving modules (front and rear) result from the shape-oriented product structure. They consist of one drive and one brake module and one axle that includes a Tracking Module as well as an Spring and Tilt Module. The Active Suspension Module, the Active Guidance Module and the Actuation Module are derived, from the function-oriented product structure.

As already explained, the initial product structure of the RailCab was refined by taking into account additional aspects. RSM and aggregation DSM are used to refine the Active Suspension Module. The MIM is used to analyze the aspects reusability and extensibility.

Phase 3 – validation of the product structure: Finally the developed product structure is validated. The core criteria of the validation are the level of compliance of the desired product structure to the underlying development task, technical and economical requirements as well as, if applicable, the available product platforms. If a need for improvement is identified, revisions of the product concept that support

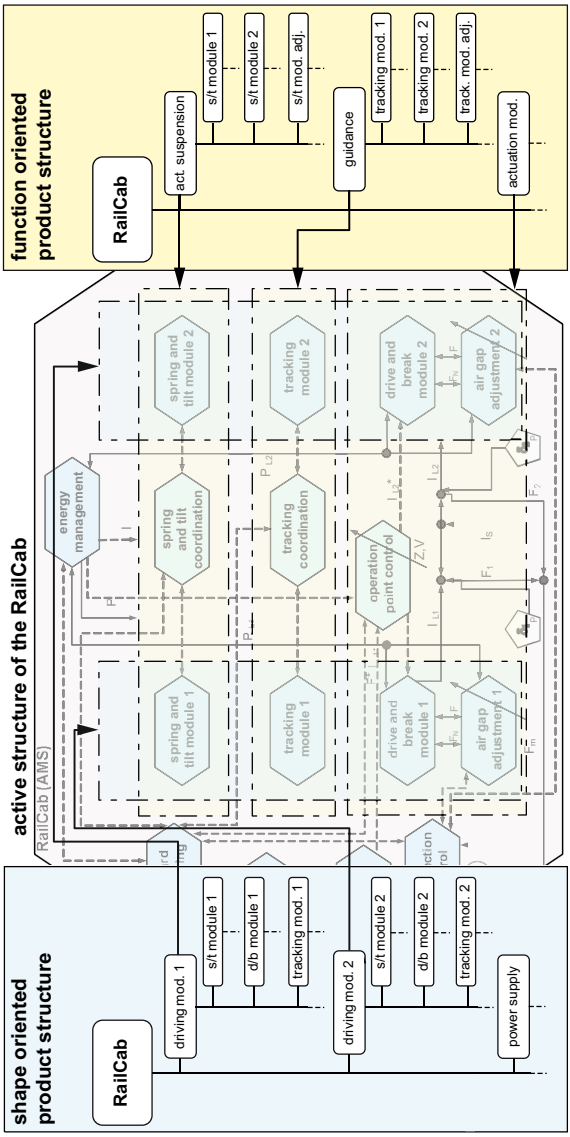


Fig. 4.31 Comparison of initial shape- and function-oriented product structure [27]

a consequent realization of the desired product structure are initiated (e.g. modifications of the interfaces). Afterwards, the parallel domain-spanning design and development of the subsystems begins. The validated, development-oriented product structure of the RailCab is shown in Fig. 4.32 [27].

Product structuring is an important step in the development process for modern mechatronic and self-optimizing systems. It helps reduce the complexity and increase the quality of a system, but it also requires additional effort. A success factor is an adequate integration in the development process, by using established

specification techniques, methods and tools. The presented approach shows, how this could be realized for mechanical engineering systems of tomorrow that may possess a high amount of information technology. The additional effort for product structuring during the conceptual design is profitable, compared to the costs of typically sub-optimal interfaces and high synchronization efforts which lead to time-intensive and costly iteration loops during the further design and development.

4.7 Early Probabilistic Reliability Analysis Based on the Principle Solution

Rafal Dorociak and Jürgen Gausemeier

Based on the domain-spanning description of the principle solution a number of analysis methods can be conducted. In this and the following section examples of such analysis methods will be shown. We begin with the method for the early probabilistic analysis of the reliability of a self-optimizing system based on its principle solution. It allows for first statements with regard to the reliability of the system in the early engineering phase of conceptual design. In particular, the weak points of the system with respect to reliability are found. For those weak points, detection measures and countermeasures are derived and implemented directly in the principle solution of the system. Altogether, the system under consideration is made more reliable in the early development stage.

The main input of our method is the domain-spanning specification of the principle solution (cf. Sect. 4.1). Following the recommendation of the CENELEC EN 50129 norm [13], our method uses two complementary reliability assurance methods FMEA (Failure Mode and Effects Analysis) [6, 25, 31] and FTA (Fault Tree Analysis) [6, 8, 32] interdependent. Some concepts known from the FHA (Functional Hazard Analysis) [58] method have been adapted, as well. This is especially relevant in, the use of a failure taxonomy for the identification of possible failures. By using these complementary methods, the completeness of the list of possible failure modes, failure causes and failure effects as well as of the specification of failure propagation is increased; both failure specifications are held mutually consistent.

Figure 4.33 shows the procedure model of our method; iterations are not shown.

Phase 1 – specification of the principle solution: The starting point are moderated workshops, where the experts from the involved domains work together in order to specify the system with the specification technique CONSENS as well as to analyze and optimize the principle solution with regard to reliability. In particular, the aspects functions, active structure, and behavior are described.

Our method has been performed for the RailCab. We will show some of the results for its Active Suspension Module. Each Active Suspension Module consists of three servo cylinders which dampen vibrations and tilt the vehicle body in curves. Figure 4.34 shows a cut-out of the partial model active structure for the servo cylinder of the Active Suspension Module. Each servo cylinder consists of a hydraulic cylinder, a 4/4-way valve, a servo cylinder regulation and a hydraulic valve regulation [46].

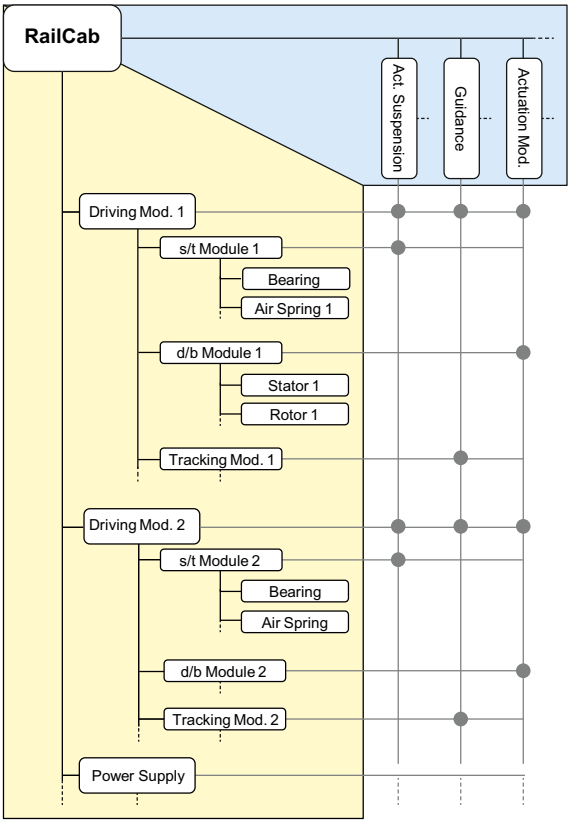


Fig. 4.32 The product structure of the RailCab [27]

Phase 2 – early FMEA based on the principle solution: The system structure and the corresponding functions are automatically derived from the description of the partial models functions, active structure, and behavior which are recorded in the FMEA table. Failure modes, failure causes and failure effects are identified then. Checklists and failure taxonomies (e.g. one shown in Fig. 4.35) [17, 56] support the failure identification process. In addition, combinations of failure modes are identified, which can possibly occur together and have a negative impact on the system (pairs of failures, trios of failures, etc.). Failure modes and relevant failure mode combinations are recorded in the FMEA table. For each failure mode (and failure mode combination) the possible causes and effects are analyzed. Check lists can be used to accomplish this because they describe system elements known to be source of problems with regard to reliability [15]. A number of failure effects can be found by analyzing the principle solution for the system; this concerns the partial models active structure and behavior. A risk assessment of the failure modes, failure causes and failure effects takes place using the risk priority number (cf. the IEC 60812 norm [31]). Finally, detection measures and countermeasures are defined as

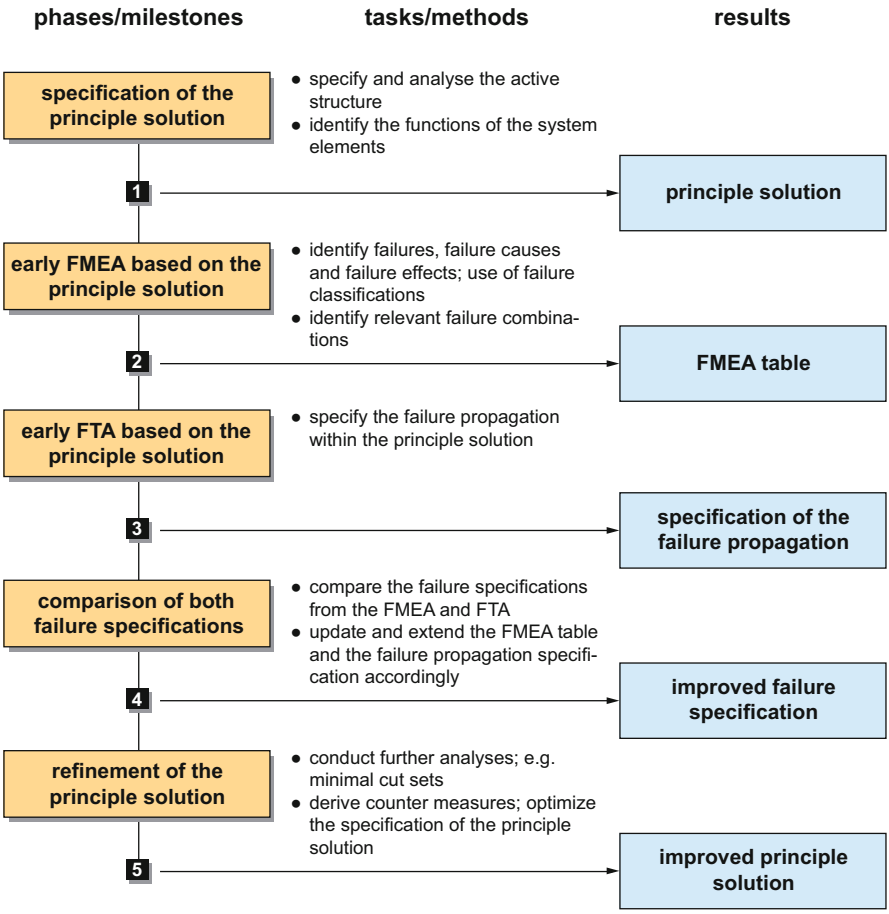


Fig. 4.33 The procedure model of the method for the early probabilistic analysis of the reliability of a self-optimizing mechatronic system

well as the corresponding responsibilities. This occurs similarly when compared to the classical FMEA. The FMEA table is updated accordingly.

The early FMEA method has been performed for the servo cylinder of the Active Suspension Module. A cut-out of the resulting FMEA table is shown in Fig. 4.36. Using the failure taxonomy by [17], the failure mode *hydraulic valve regulation provides no switch position for the 4/4-way valve* is found. This failure mode occurs, for instance, if the energy supply of the system element hydraulic valve regulation is interrupted. According to the FMEA the risk priority number for this case is 252. In order to eliminate or at least mitigate the failure mode, the energy supply of the hydraulic valve regulation should be monitored. One possible solution is to incorporate an additional monitoring system element into the principle solution. Then additional measures such as a redundant energy supply have to be implemented.

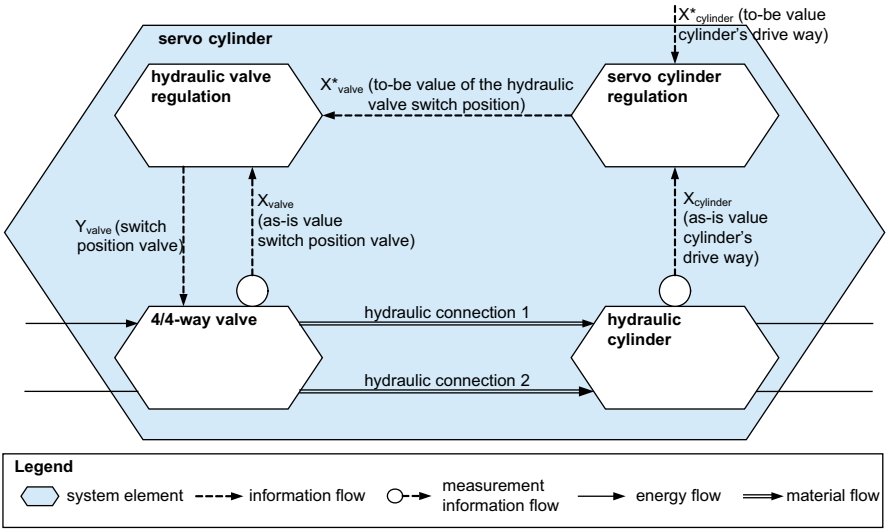


Fig. 4.34 Active structure of the Active Suspension Module (cut-out)

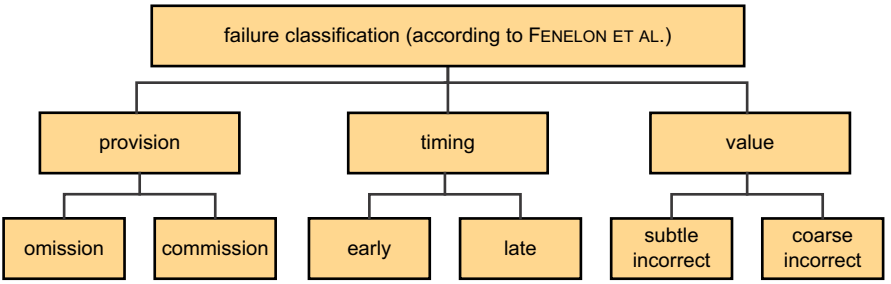
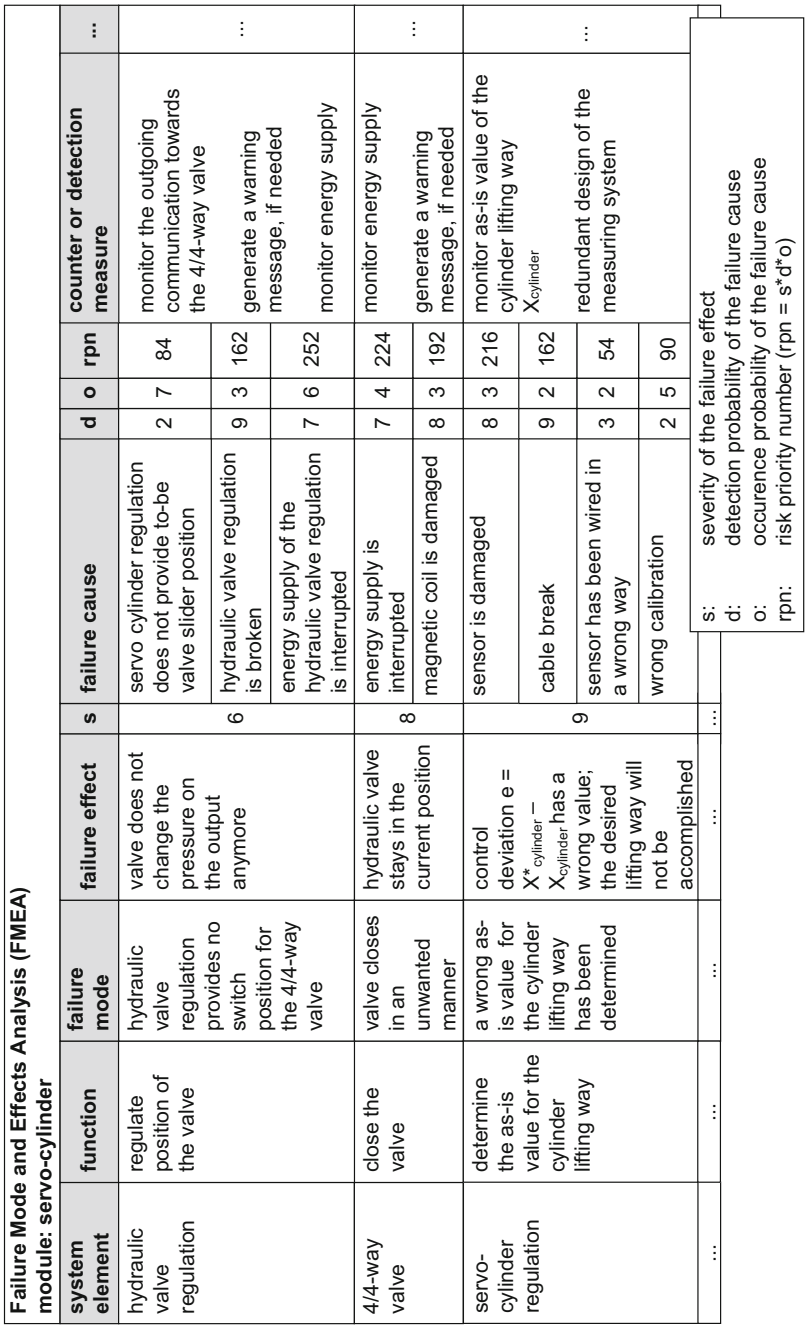


Fig. 4.35 Failure classification (according to FENELON ET AL.(1994)) [17]

Phase 3 – early FTA based on the principle solution: The specification of the failure propagation within the principle solution is performed. The process is very similar to the traditional FTA. For each system element, its internal failures as well as incoming and outgoing failures are specified and related to each other.

In our application example, the specification of the principle solution is extended by the specification of failure propagation (cf. Fig. 4.37). For each system element the relationship between incoming, local and outgoing failures is described. For instance, the output HV_2 exhibits an undesired system behavior, if the internal failure "F1" or "F2" occur or one the input HV_1 is faulty. Based on such a description of the failure propagation a fault tree can be generated (semi)-automatically [8].

Phase 4 – comparison of both failure specifications: The FMEA table and the specification of the failure propagation both contain information about causal relationships between failures. Following the recommendation of the CENELEC EN 50129 [13], we use both methods in combination, to ensure a completeness of the



Failure Mode and Effects Analysis (FMEA)
module: servo-cylinder

system element	function	failure mode	failure effect	s	failure cause	d	o	rpn	counter or detection measure	...
hydraulic valve regulation	regulate position of the valve	hydraulic valve regulation provides no switch position for the 4/4-way valve	valve does not change the pressure on the output anymore	6	servo cylinder regulation does not provide to-be valve slider position	2	7	84	monitor the outgoing communication towards the 4/4-way valve	...
					hydraulic valve regulation is broken	9	3	162	generate a warning message, if needed	
					energy supply of the hydraulic valve regulation is interrupted	7	6	252	monitor energy supply	
4/4-way valve	close the valve	valve closes in an unwanted manner	hydraulic valve stays in the current position	8	energy supply is interrupted	7	4	224	monitor energy supply	...
					magnetic coil is damaged	8	3	192	generate a warning message, if needed	
servo-cylinder regulation	determine the as-is value for the cylinder lifting way	a wrong as-is value for the cylinder lifting way has been determined	control deviation e = $X^*_{cylinder} - X_{cylinder}$ has a wrong value; the desired lifting way will not be accomplished	9	sensor is damaged	8	3	216	monitor as-is value of the cylinder lifting way $X_{cylinder}$	...
					cable break	9	2	162		
					sensor has been wired in a wrong way	3	2	54	redundant design of the measuring system	
					wrong calibration	2	5	90		
...	...	...	...	...						

s: severity of the failure effect
d: detection probability of the failure cause
o: occurence probability of the failure cause
rpn: risk priority number (rpn = s*d*o)

Fig. 4.36 FMEA table of the servo cylinder (cut-out)

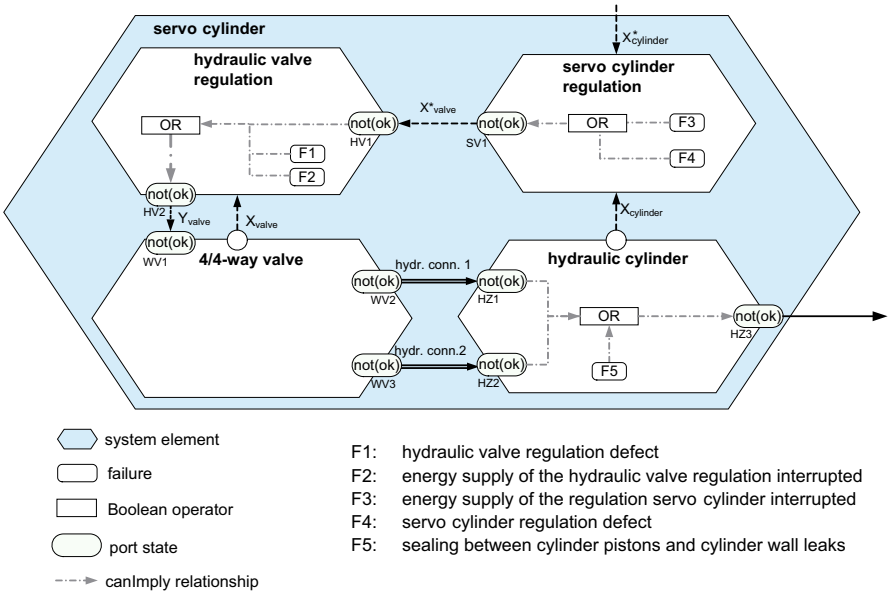


Fig. 4.37 Specification of the failure propagation of the servo cylinder (cut-out)

failure specification. This can be achieved by comparing the information content of the FMEA and the failure propagation specification: e.g. failures and causal relationships between failures can potentially be found in the failure propagation specification, which have not been found during the FMEA and are thus not documented in the FMEA table; the FMEA table is updated accordingly. This also applies for the other comparison direction: For example, if a causal relationship between two failures (e.g. between a failure mode and a failure effect) has been recorded in the FMEA table, there has to be a corresponding causal relationship in the failure propagation specification. If this is not the case, the causal relationship is incorporated into the failure propagation specification. In the process, some additional failures which have not been specified can be found. The completeness of the identified failure modes, failure effects, failure causes as well as of the failure propagation specification is improved.

Figure 4.38 depicts the interrelation between both failure representations for the servo cylinder. The failure cause *servo cylinder regulation does not provide to-be valve switch position* from the FMEA table (cf. Fig. 4.38, (1)) corresponds to the port state not(ok) of the input HV1 of the system element hydraulic valve regulation. The failure causes *hydraulic valve regulation is broken* (2) and *energy supply of the hydraulic valve regulation is interrupted* (3) correspond to the internal failures F1 and F2 of the hydraulic valve regulation. The aforementioned failure causes (2) and (3) may lead to the failure *hydraulic valve regulation provides no switch position for the 4/4-way valve* (4); it is recorded in the FMEA table as well as in the specification

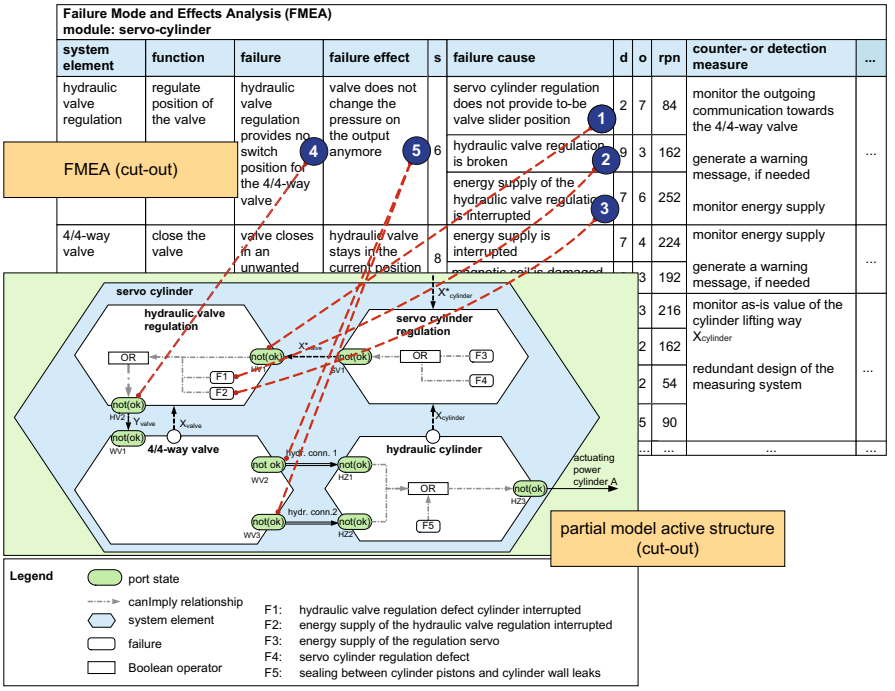


Fig. 4.38 Interrelation between the FMEA table and the specification of the failure propagation

of the failure propagation (port state not(ok) of the output HV2 of the hydraulic valve regulation).

According to the FMEA table there is a causal failure relationship between the failure *hydraulic valve regulation provides no switch position for the 4/4-way valve* (4) and the failure effect *valve does not change the pressure on the output anymore* (5). Although both failures were specified in the failure propagation model (input WV1 of the 4/4-way valve as well as inputs HZ1 and HZ2 of the hydraulic cylinder, respectively), the causal relationship between them has not been modeled. As a consequence, a thorough analysis was performed on the causal relationship. During this course the respective failure propagation path was modeled as well as an additional failure F6 (*valve position can no longer be changed mechanically; the valve slider stays in its current position*) (cf. Fig. 4.39).

Phase 5 – refinement of the principle solution: Both failure specifications are analyzed. For instance, the classical analyses known from the FTA field such as minimal cut sets are used [6]. In particular, the importance analysis is performed. For this purpose, the Bayesian network driven approach is used [9]; it enables the computation of the Fussell-Vesely importance measure. In this manner, the most critical system elements are identified. Detection measures and countermeasures are defined based on the analysis results. If possible, they are directly incorporated into

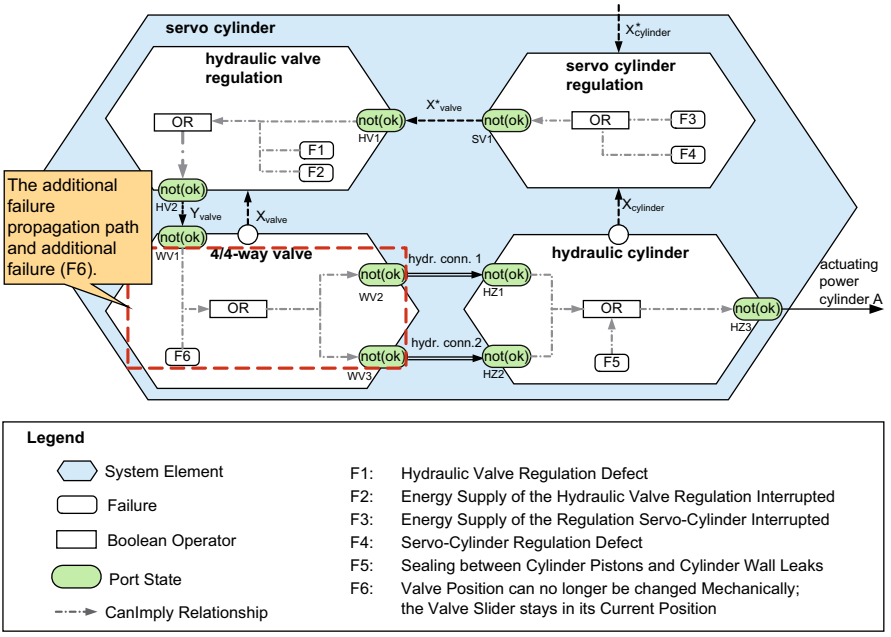


Fig. 4.39 The extended failure propagation specification of the servo cylinder

the principle solution (e.g. redundancy, condition monitoring [37], etc.). Otherwise, they are recorded for further domain-specific design and development (e.g. test and simulation measures, etc.).

The result of the method is an updated principle solution for the system which is improved with regard to reliability. As a consequence, the reliability of the system under consideration is improved during the early development stage and a great number of time-intensive and costly iteration loops during the further development phases is avoided. The failure specifications and analysis results from the conceptual design are used in the further development phase of domain-specific design and development. The reliability analyses such as FTA and FMEA are performed again along with the concretization of the system.

In our application example, the specification of the failure propagation of the servo cylinder from the Active Suspension Module is translated into a Bayesian network. The translation algorithm proceeds as follows: for each system element its internal failures and port states of its inputs and outputs are translated into nodes of the Bayesian network. The relationships between them are represented as edges in the Bayesian network. The Conditional Probability Table (CPT) of the Bayesian network is then populated: for each value of variables associated to a node or a node state, its conditional probabilities are described, with respect to all combination of values associated to variables of the parent nodes in the network. To support the translation, a dictionary of translation rules has been developed [9], [26, D.o.S.O.M.S. Sect. 3.1] .

The result is a comprehensive Bayesian network, which describes the part of the system that is relevant for the examination of the chosen top event. Based on the Bayesian network some further analyses are performed [36]. In particular, the Fussell-Vesely importance measure is computed, i.e. it is determined with what probability a particular system element (failure cause) had led to a particular failure (the so-called posterior probability). The top event that we examine is *valve does not change the pressure on the output anymore* (corresponds to the port state WV2.not(ok)). Let us consider the state of the failure specification before the additional failure F6 and the corresponding propagation path were incorporated into the specification. The failure rates of the failures are shown in Tab. 4.4. Let us further assume, that the output WV2 of system element hydraulic valve regulation is in state not(ok), as this is our top event. According to the specification of the failure propagation (cf. Fig. 4.37) failures F1, F2, F3 and F4 contribute to this. Table 4.4 reports the Fussell-Vesely importance measure of each failure, i.e. the posterior probability of the contributing failures given the occurrence of the aforementioned failure. Failures F1 and F4 are especially important with importance greater than 28 %.

Table 4.4 Failures, the failure rates and the Fussell-Vesely importance measure (before and after the failure specification had been extended) (Top-Event is WV2.not(ok))

Failure	Failure rate (per hour)	Fussel-Vesely importance (before)	Fussel-Vesely importance (after)
F1	5.11×10^{-7}	0.2897	0.2416
F2	4.02×10^{-7}	0.2279	0.1901
F3	3.28×10^{-7}	0.1859	0.1551
F4	5.23×10^{-7}	0.2965	0.2473
F6	3.51×10^{-7}	N/A	0.1660

Now let us consider the extended specification of the failure propagation (including failure F6). The failure rate of failure F6 and the respective importance measures are shown in Tab. 4.4. The failures F1, F2, and F4 are of highest importance with importance of approximately 25 %.

All in all, by using our method, the completeness of the failure specification has been improved. Especially, failures and failure relationships were identified, which could have been easily omitted otherwise. In our application example, the failure F6 has been identified, the importance of which is quite high (approximately 17 %). Based on the failure specification, further analyses are conducted. Detection measures and countermeasures are then derived and, if possible, implemented directly in the principle solution. Altogether, the reliability of the system under consideration is improved during the early development stage.

4.8 Evaluation of the Economic Efficiency

Mareen Vaßholz

The decision to design a self-optimizing system is made in the early development phase, when the potential for the optimization of contradictory objectives is identified. In this case the developer has to determine whether these contradictions will be resolved in the further development by compromising or by using self-optimization. The technical feasibility and economical aspects also contribute to this decision, since the use of self-optimization can result in changing resource requirements for the development, production and operation when compared to a conventional mechatronic solution.

The **economic efficiency** of a system is given by the ratio of its evaluated monetary benefit to costs. The result is a dimensionless number. If it is exactly 1, neither profit nor loss is made [55]. In particular, the evaluation of the benefits of a self-optimizing system is challenging because it occurs during run-time of the system. The high complexity and the dynamics of the system in operation make the evaluation in the conceptual design phase difficulty. Existing methods to evaluate the costs and benefit do not meet this complexity [57]. The aim of the presented method is therefore to provide proof of the economic efficiency of self-optimizing product concepts during the early stage of the conceptual design based on the principle solution. This includes the evaluation of the two aspects costs and benefit for the company as well as for the customer. Furthermore it enables a comparison with conventional mechatronic solutions in order to select the most economical one. On this basis, the decision can be made whether the solution variant will be developed further in the design and development and eventually brought to market.

Figure 4.40 provides an overview of the phases and milestones of the method for the early estimation of the economic efficiency of a self-optimizing system based on the principle solution. The starting point for the development of a technical system are different product ideas, that can be beneficial for the customer and the company. Before the development of one of these product ideas is initiated, it needs to be clarified whether this can strategically benefit the company. In phase 1 the market for the product ideas is analyzed. For the stakeholder the qualitative benefit is identified by experts of the company. Based on these benefits the changes of the market performance of the company can be derived. For example through a competitive advantage, shares of sales from the competitors can be tapped and a higher revenue growth for the company results. In the case of promising expected revenue growth, the conceptual design of the product is initiated in phase 2. To be able to estimate the production costs in the subsequent phase, the production system is designed as well. Before the solution variant is developed further in the design and development phase, its economic efficiency needs to be proven. In order to pursue this we need to distinguish the economic efficiency for the company and for the customer. The quotient of the expected benefit (expected revenue growth and market price) and the anticipated costs (development, production and investment costs) describe the economic efficiency for the company. For the customer the solution variant is

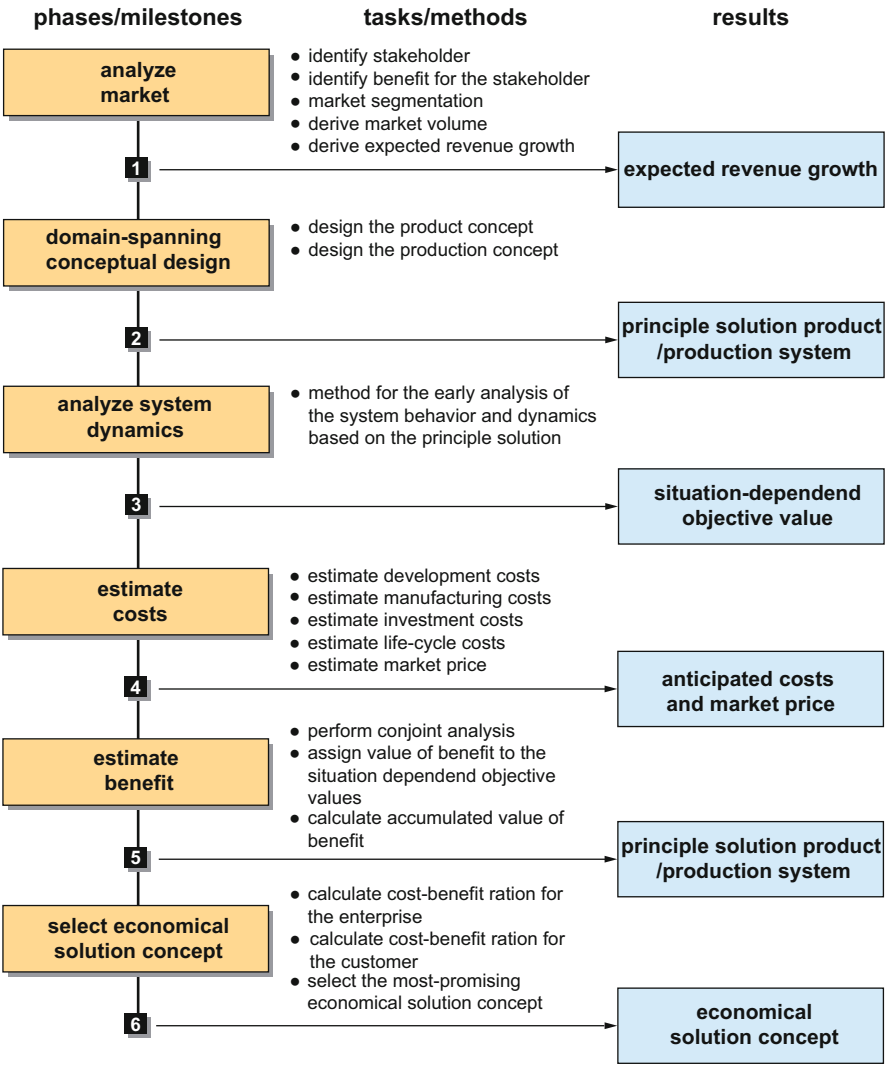


Fig. 4.40 Phase-milestone diagram for the early estimation of the economic efficiency of a self-optimizing system

economically efficient, if the quotient of the accumulated benefit in system operation to the life-cycle costs (purchase price, operation, maintenance and recycling costs) result in a value higher than one.

To be able to determine the benefit for the customer as well as the operation costs, the dynamics of the self-optimizing and the conventional mechatronic systems needs to be analyzed in phase 3. The result is the behavior of the systems in different operating situations. In the following phase the costs for the solution

variants are estimated in phase 4. In phase 5 the benefit for the customer is estimated based on a conjoint analysis. The result is annotated to the simulation results of phase 3 and by accumulation, the benefit for the customer over the life-cycle of the system results. Finally the economic efficiency of each solution variant is calculated in phase 6 and the most promising one is selected and developed further.

Phase 1 - analyze market: The first phase is carried out before the conceptual design of the system ideas. Before time and money are invested for the development of a technical system, it needs to be clarified whether the development is advantageous for the company. Therefore the potential for the new system on the market needs to be analyzed and the expected revenue growth identified. This is the case, when stakeholders derive benefit from the system and thus are willing to buy it. Based on the approach by Freeman (1984) [19], the stakeholders are identified and analyzed. In this case a stakeholder is a group or an individual, who can affect or is affected by the achievement of the system to be developed. In the next step the identified stakeholder are categorized by the three attributes power, legitimacy and urgency due to Mitchell et al. (1997). They distinguish between eight categories: dormant, discretionary, demanding, dominant, dangerous, dependent, definitive stakeholder as well as nonstakeholder [41]. From this distinction preliminary indications can be conducted on how the stakeholder will benefit from the new system. For example dormant stakeholders could be customers of the competitors. In the case that the new system is brought to the market, they can be a potential customer for the new system, because their expectations are met. This leads to a growth in revenue for the company at the expense of the competitor. The identification of the benefit for the stakeholder by causal chains is performed in the next step. The qualitative benefit is collected in a stakeholder-benefit-matrix. The business is structured into market segments according to the segmentation criteria by Backhaus (2003) [2]. The identified stakeholder can be assigned to the market segments. The segments are compared to the market performance of the system ideas in a matrix. For each combination, the respective market volume, the company revenue, the sales growth in the previous year and the expected revenue growth are evaluated. The expected revenue growth can be predicted by the benefit that is resulting for the respective stakeholder. The result of the first phase is the expected revenue growth resulting from the development of the system for the company. On this basis, a decision is made whether the development of the system is advantageous or disadvantageous. In the first case the conceptual design of the idea is triggered. Otherwise the idea is rejected.

Phase 2 - domain-spanning conceptual design: A promising revenue growth is the trigger for the development of the system. In phase 2, the principle solution for the system is developed. To be able to estimate the production costs of the system, the respective production system needs to be designed in the early phase as well. For the development of the principle solution for the product the approach in Sect. 3.2 is used. In case a self-optimizing solution is not expressively required, a mechatronic and a self-optimizing solution is made. To develop the principle solution for the production system, part dimensions and material data are derived during the first step of determining the manufacturing requirements. Afterwards the active

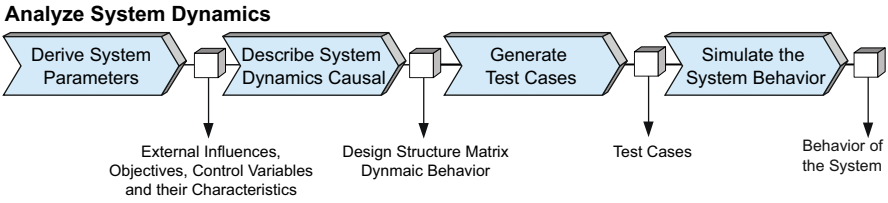


Fig. 4.41 Four steps to analyze the system dynamics

structure and the building structure of the self-optimizing/mechatronic system are analyzed, to identify all system elements that need to be manufactured. Based on the structural connections, the process sequence for the manufacturing and the assembly process are set up. Based on the initial assembly sequence, the system elements are linked by assembly processes. Next, parts to be purchased and parts that need to be manufactured are determined. The parts to be manufactured are completed by the necessary manufacturing processes to produce them from raw materials or prefabricated parts. In this context adequate manufacturing technologies are chosen with respect to the deployed product technologies then the resources of the production system are determined. The resources realize the specified processes and are allocated to them, based on the chosen production technologies. The selection depends on the production requirements, this way, alternative resource combinations are obtained [22]. In the following phases the developed principle solution for the self-optimizing and mechatronic system will be analyzed regarding the economic efficiency, whereupon the concept of the production systems gives evidence for the production costs.

Phase 3 - analyze system dynamics: Because the benefits and costs of self-optimizing systems during operation results from the dynamics of the system, it needs to be analyzed. A particular challenge for self-optimizing systems is, that their behavior can not be predicted easily due to its interdependencies and the resulting high complexity. To reduce the complexity, we use a matrix based approach and map the dependencies of the control loop between external influences, control variables of the system and the objectives of the system in a Multiple Domain Matrix. By matrix multiplication the continuous **self-optimization process** can be simulated and the respective system state detected. This results in having the objective characteristics of the system state for each operating condition over time. For this purpose four steps need to be performed (cf. Fig. 4.41):

Step 1 - derive system parameters: To be able to simulate the system dynamics the principle solution (cf. Sect. 4.1) for the self-optimizing system and of the conventional mechatronic system needs to be analyzed to derive the necessary parameter. These are the external influences, control variables and the objectives of the system. External influences result from the flows in the aspect environment. This can be for example the desired speed of the RailCab by the customer. The control variables can be identified based on the active structure of the system, e.g. the

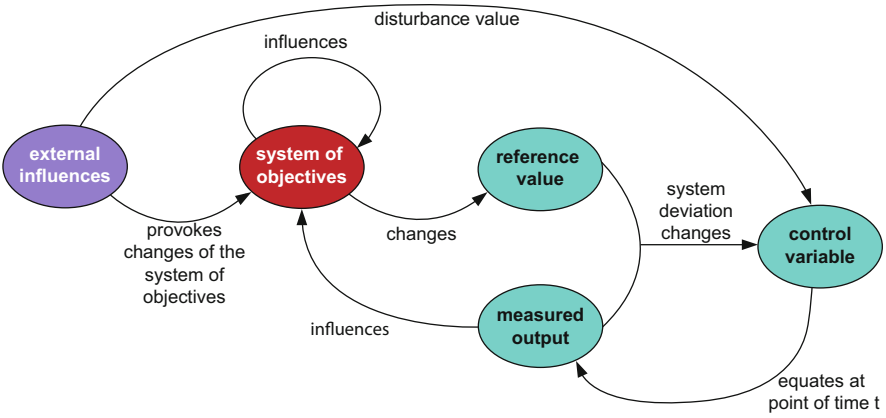


Fig. 4.42 Closed-loop representing the dynamic behavior of a self-optimizing system

charge rate of the capacitor of the Hybrid Energy Storage System. An example for an objective of the RailCab is "minimize energy losses". For each parameter characteristics are determined. For instance the priority of the objective "minimize energy losses" can be qualitative low, high or very high.

Step 2 - describe causal system dynamics: The dependencies of the parameters in the closed-loop of the self-optimizing system derived in step 1 are presented in a causal diagram in Fig. 4.42. In the case that the external influences of the environment on the system changes a changed priority of the objectives of the system can be provoked. The objectives of the system of objectives are also influenced by itself and the current system state. The system state can be influenced by the environment by disturbing values. The self-optimizing system determines its objectives, if necessary. This leads to a changed reference value for the control strategy of the system. The deviation between reference value and measured system state can change the control variable and therefore the behavior of the system. This circumstance is described in the Multiple Domain Matrix in Fig. 4.43.

Step 3 - generate test cases: In this step the life-cycle of the system is represented by test cases consisting of different operating situations. For this, the situation of the aspect application scenarios is formalized. In the Design Structure Matrix (cf. Fig. 4.43, matrix No. 1) the consistency of the characteristics of the external influences were analyzed [21]. Different operating situations are derived from this matrix, consisting of a combination of characteristics of the external influences. These are clustered by similarity and assigned to the application scenarios, which describe one situation during the life-cycle of the system. Afterwards a sequence of application scenarios is generated and each one is provided with a time stamp. Thus test cases that represent the life-cycle result.

Step 4 - simulate the system behavior: For these test cases the dynamical behavior of the systems is simulated by matrix multiplication. For the self-optimizing system the self-optimization process is conducted as follows:

1. **Analyzing the current situation:** The self-optimization process is initiated, in case either the situation and therefore the external influences or the system state changes. The first case results by the sequence of the application scenarios in the test cases. The associated state vector for the external influences can be derived from matrix No. 1 (cf. Fig. 4.43). Furthermore it is examined whether this change leads to a changing system state because of disturbance variables (Fig. 4.43, matrix No. 3). The other case occurs for example when the capacity of the battery of the RailCab switches from one state to another, due to energy consumption of the system, as described in "4. Continuous system operation". The state vector for the control variable is derived.
2. **Determining the system's objectives:** The next step is to determine the objective of the system. The priority of the objectives is dependent on external influences (Fig. 4.43, matrix No. 2), the system state (Fig. 4.43, matrix No.6) and the current priority of the objectives (matrix No. 4). Each characteristic in the matrices demand a certain priority of the objectives. Out of these demands the Pareto optima for every objective prioritization is chosen and results in the state vector for the new objectives.
3. **Adapting the system behavior:** The alteration of the system of objectives for the system, demands a change of the systems behavior. Figure 4.43, matrix No. 5 presents the influence of the control variables by the priority of the objectives. The new state vector of the system can be taken and the system switches to another operation mode.
4. **Continuous system operation:** Since the system behavior can change during the operation, e.g. by energy consumption, continuous operation of the system is simulated as well. How the system state is changing over time is described in Fig. 4.43, matrix No. 8. The consumption, for example of the available energy is simulated over time. When the system state changes to another mode the self-optimization process is initiated again.

This procedure is conducted for all scenarios of the test cases. The respective priority of the objectives as well as the system state is recorded to be able to assign the operation costs and the situational benefit to each test case. For the conventional mechatronic system the simulation is conducted in a similar way, except that the changes of the objectives are limited to the defined control strategies.

Phase 4 - estimate costs: In this phase, the costs for the solution concepts are estimated. The costs for the company consist of the development, investment and production costs, which in turn are composed of various expenses. For the customer the life-cycle costs of the system are of interest. The presented method provides a guideline to estimate the relevant costs for the self-optimizing and mechatronic system.

To determine the **development costs**, the costs of all initiated processes for the development are required. For this purpose, the following questions must be answered: What do we do? How do we do it? and Who does it? [12]. To this end, the reference process (cf. Chap. 3) for the development of self-optimizing system is tailored due to the individual development task. This is accomplished using the

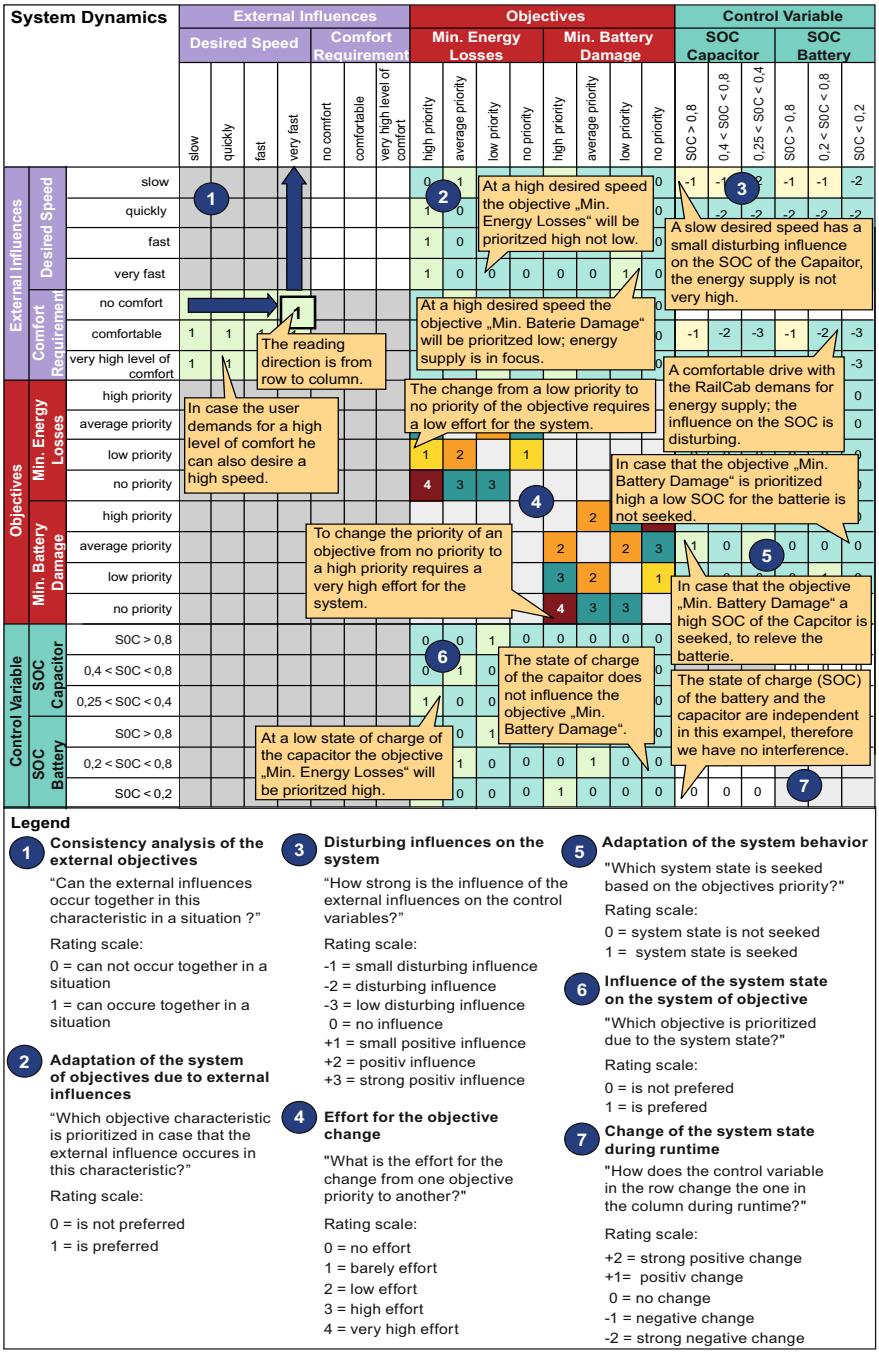


Fig. 4.43 Causal description of the system dynamics with a Multiple Domain Matrix

framework for a self-optimizing process development of advanced mechatronic systems by Kahl (2013) [33] (cf. Sect. 3.4). For each process step it can be determined, which developer performs it, how long it will take and what potential additional expenses to the personnel costs will arise. The required personnel costs plus other expenses form the development costs.

The **investment costs** for the company result from the procurement of material resources and from training costs for the developer in self-optimization specific expertise. For self-optimizing systems, this may result in part from the provision of new production plants, and also from the procurement of test bed and platforms. The investment costs for the production system can be derived from the principle solution for the production system. The necessary test beds and test platforms as well as training for the developers are associated with the solution pattern for self-optimization. Depending on the selected solution patterns, the necessary investments for the system test can be estimated.

The core of the product-related costs are the **production costs**. These consist of the basic mechanical structure by the material, manufacturing, assembly and testing costs. The manufacturing and assembly costs can, for example, be determined by the process costs based on the principle solution for the production system (cf. [42]). The production costs for the electronic components result in addition to the material and manufacturing costs from the so-called yield-loss costs as well as test costs [49]. To calculate the production costs for the software components, the staff required for the implementation, integration, and testing in personnel-months is included. In addition, there are charges which are incurred in the administration and sales which can not be directly associated with the system. These are shown in the surcharge calculation of overhead rates and added to the production costs. This results in the cost for the company for the product [12]. Based on these the market price can be calculated.

The **life-cycle costs** for the customer include the purchase price, the operating costs of the system as well as maintenance and recycling costs. The purchase price corresponds to the market price set by the company earlier in this phase. The operating costs of the system can be derived based on the simulated behavior of the system in phase 3. Over all operating situations, the changes of the system behavior were simulated. For each modification switching costs result. For the simulation these costs are taken as a factor. Furthermore, the consumption of the system has been simulated as well based on these factors. By the expertise of the developer these factors can be transform to monetary costs. The maintenance cycles can be estimated through the product life-cycle based on the simulation. Disposal costs are estimated based on the active structure and the expertise of the developer.

Phase 5 - estimate benefit: The benefit of a self-optimizing system during operation must be evaluated in terms of its situational dependency. The behavior of the system in different operating situations has already been simulated in phase 3. In this phase the benefit that results from certain behaviors of the system for the customer is identified with the traditional conjoint analysis [3]. In the first step the system properties and their characteristics to be queried are determined. For this purpose,

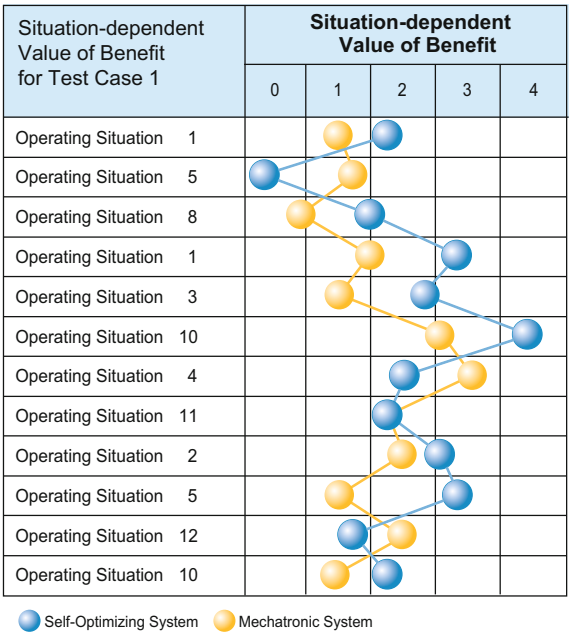


Fig. 4.44 Roadmap for the value of benefit in different operating situations

the operating situations with the highest probability, the value of the system of objectives in the situation, as well as the operating costs for the situation are chosen. These are prioritized during a survey. Furthermore the price that the customer is willing to pay for the system should be queried in order to determine possible prices for the system based on the desired profit margins.

In the second step, the survey design is created. With the aid of the profile method, the stimuli which means the combinations of property characteristics are created. Then the number of stimuli is determined and a reduced design, which makes an evaluation manageable, is designed. The selected stimuli and operating situations are clearly described.

Then the stimuli are evaluated by the focus group. This group is selected by stakeholders in regard to the relevant market segments. The respondents are asked to prioritize the stimuli so that the resulting ranking order matches their personal preferences. Upon completion of the survey, partial values of benefit are determined for all property characteristics based on the empirically determined ranking-data. From this, the total values of benefit for all stimuli and the relative importance of each property can be derived. The aggregation of values of benefit is achieved using cluster analysis [21].

Finally, the values of benefit are assigned to the operating conditions for each test case and the situation-dependent benefit of each alternative solution is presented in a benefit-roadmap (cf. Fig. 4.44). The cumulative benefit of a the solution variant is derived from the weighted sum of the partial values of benefit. The monetary benefit is derived from the survey, based on the computed preferred price for the system and expenditure for a situation.

Phase 6 - select economical solution concept: The expected benefit and costs for the company and the customer are compared and the economic efficiency of each solution is calculated. The comparison provides the basis for decisions on the selection of the most economical solution variant that will be developed further in the domain-spanning design and development (cf. Sect. 3.3).

References

1. Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdhlking, I., Angel, S.: A Pattern Language. Oxford University Press, Oxford (1977)
2. Backhaus, K.: Industriegütermarketing, 13th edn. Vahlen, München (2003)
3. Backhaus, K., Erichson, B., Plinke, W., Weiber, R.: Multivariate Analysemethoden - Eine anwendungsorientierte Einführung, 13th edn. Springer, Heidelberg (2011)
4. Bertsche, B.: Reliability in Automotive and Mechanical Engineering - Determination of Component and System Reliability. Springer, Heidelberg (2008)
5. Birkhofer, H.: Analyse und Synthese der Funktionen technischer Produkte. Ph.D. thesis, Technische Universität Braunschweig, VDI-Verlag, Düsseldorf (1980)
6. Birolini, A.: Reliability Engineering - Theory and Practice, 5th edn. Springer, Heidelberg (2007)
7. Clark, N.: The Airbus Saga - Crossed Wires and a Multibillion-euro Delay - Business - International Herald Tribune, `http://www.nytimes.com/2006/12/11/business/worldbusiness/11iht-airbus.3860198.html?pagewanted=all` (accessed July 1, 2013)
8. Deyter, S., Gausemeier, J., Kaiser, L., Poeschl, M.: Modeling and Analyzing Fault-Tolerant Mechatronic Systems. In: Proceedings of the 17th International Conference on Engineering Design, Stanford (2009)
9. Dorociak, R.: Early Probabilistic Reliability Analysis of Mechatronic Systems. In: Proceedings of the Reliability and Maintainability Symposium (2012)
10. Dumitrescu, R.: Entwicklungssystematik zur Integration kognitiver Funktionen in fortgeschrittene mechatronische Systeme. Ph.D. thesis, Fakultät für Maschinenbau, Universität Paderborn, HNI-Verlagsschriftenreihe, Band 286, Paderborn (2011)
11. Ehrlenspiel, K.: Integrierte Produktentwicklung, 2nd edn. Carl Hanser Verlag, München (2003)
12. Ehrlenspiel, K., Kiewert, A., Lindemann, U.: Cost-Efficient Design. Springer, Heidelberg (2007)
13. for Electrotechnical Standardization (CENELEC), E.C.: CENELEC EN 50129: 2003. Railway Applications - Communication, Signalling and Processing Systems - Safety Related Electronic Systems for Signalling. Norm (2003)
14. Eppinger, S., Whitney, D., Smith, R., Gebala, D.: A Model-Based Method for Organizing Tasks in Product Development - Reserarch in Engineering Design. Springer, Heidelberg (1994)
15. Ericson, C.: Hazard Analysis Techniques for System Safety. John Wiley & Sons, Hoboken (2005)
16. Erixon, G.: Modular Function Deployment - A Method for Product Modularization. Ph.D. thesis, Royal Institute of Technology, KTH, Stockholm (1998)
17. Fenelon, P., McDermid, J.A., Nicolson, M., Pumfrey, D.J.: Towards Integrated Safety Analysis and Design. ACM SIGAPP Applied Computing Review 2(1), 21–32 (1994)

18. Frank, U.: Spezifikationstechnik zur Beschreibung der Prinziplösung selbstoptimierender Systeme. Ph.D. thesis, Fakultät für Maschinenbau, Universität Paderborn, HNI-Verlagsschriftenreihe, Band 26, Paderborn (2006)
19. Freeman, R.E.: Strategic Management – A Stakeholder Approach. Pitman, Marschfield (1984)
20. Friedenthal, S., Steiner, R., Moore, A.C.: Practical Guide to SysML: The Systems Modeling Language. Elsevier, Amsterdam (2008)
21. Gausemeier, G., Plass, C., Wenzelmann, C.: Zukunftsorientierte Unternehmensgestaltung - Strategien, Geschäftsprozesse und IT-Systeme für die Produktion von morgen. Carl Hanser Verlag, München (2009)
22. Gausemeier, J., Brandis, R., Kaiser, L.: Integrative Conceptual Design of Products and Production Systems of Mechatronic Systems. In: Procedings of the Workshop on Research and Education in Mechatronics, Paris (2012)
23. Gausemeier, J., Dorociak, R., Pook, S., Nyssen, A., Terfloth, A.: Computer-Aided Cross-Domain Modeling of Mechatronic Systems. In: Proceedings of the International Design Conference, Dubrovnik (2010)
24. Gausemeier, J., Frank, U., Donoth, J., Kahl, S.: Specification Technique for the Description of Self-optimizing Mechatronic Systems. Research in Engineering Design 20(4), 201–223 (2009)
25. Gausemeier, J., Kaiser, L., Pook, S.: FMEA von komplexen mechatronischen Systemen auf Basis der Spezifikation der Prinziplösung. ZWF 11 (2009)
26. Gausemeier, J., Rammig, F.J., Schäfer, W., Sextro, W. (eds.): Dependability of Self-optimizing Mechatronic Systems. Springer, Heidelberg (2014)
27. Gausemeier, J., Steffen, D., Donoth, J., Kahl, S.: Conceptual Design of Modularized Advanced Mechatronic Systems. In: Proceedings of the 17th International Conference on Engineering Design, Stanford (2009)
28. Gimpel, B., Herb, R., Herb, T.: Ideen finden, Produkte entwickeln mit TRIZ. Hanser Verlag, München (2000)
29. Greenyer, J.: Scenario-based design of mechatronic systems. Ph.D. thesis
30. Harel, D., Maoz, S.: Assert and Negate Revisited: Modal Semantics for UML Sequence Diagrams. Software and System Modeling 7(2), 237–252 (2007)
31. International Electrotechnical Commission (IEC): Analysis Techniques for System Reliability - Procedure for Failure Mode and Effects Analysis (FMEA), IEC 60812 (2006)
32. International Electrotechnical Commission (IEC): Fault Tree Analysis (FTA), IEC 61025 (2006)
33. Kahl, S.M.: Rahmenwerk für einen selbstoptimierenden entwicklungsprozess fortschrittlicher mechatronischer systeme. Ph.D. thesis, Fakultät für Maschinenbau, Universität Paderborn, HNI-Verlagschriftenreihe, Band 308, Paderborn (2013)
34. Koller, R., Kastrup, N.: Prinziplösungen zur Konstruktion technischer Produkte. Springer, Heidelberg (1998)
35. Langlotz, G.: Ein Beitrag zur Funktionsstrukturentwicklung innovativer Produkte. Ph.D. thesis, Institut für Rechneranwendung in Planung und Konstruktion, Universität Karlsruhe, Shaker Verlag, Band 2, Aachen (2000)
36. Langseth, H., Portinale, L.: Bayesian Networks in Reliability. In: Reliability Engineering & System Safety (2007)
37. Lee, J., Ni, D., Djurdjanovic, H., Qiu, H., Liao, H.: Intelligent Prognostic Tools and E-maintenance. Computers in Industry 57(6), 476–489 (2006)
38. Lindemann, U., Maurer, M.: Individualisierte Produkte - Komplexität beherrschen in Entwicklung und Produktion. Springer, Heidelberg (2006)

39. Blackenfelt, M.: On the Development of Modular Mechatronic Products. Royal Institute of Technology, KTH Stockholm (1999)
40. Mehrabian, A., Russell, J.A.: An Approach to Environmental Psychology. MIT Press, Cambridge (1974)
41. Mitchell, R.K., Agle, B.R.: Towards a Theory of Stakeholder Identification and Salience - Defending the Principle of Who and What Really Counts. Journal = Academy of Management Review 22(4), 11–14 (1997)
42. Nordsiek, D., Gausemeier, J., Lanza, G., Peters, S.: Early Evaluation of Manufacturing Costs within an Integrative Design of Product and Production System. In: Proceedings of the APMS 2010, International Conference on Advances in Production Management Systems, Como (2010)
43. Pahl, G., Beitz, W., Feldhusen, J., Grote, K.H.: Engineering Design - A Systematic Approach, 3rd edn. Springer, Heidelberg (2007)
44. Pilone, D., Pitman, N.: UML 2.0 in a Nutshell: A Desktop Quick Reference. O'Reilly (2005)
45. Pook, S.: Eine Methode zum Entwurf von Zielsystemen selbstoptimierender mechatronischer Systeme. Ph.D. thesis, Fakultät für Maschinenbau, Universität Paderborn, HNI-Verlagsschriftenreihe, Band 296, Paderborn (2011)
46. RailCab - Neue Bahntechnik Paderborn: The Project Web Site, `http://railcab.de` (accessed March 5, 2012)
47. Roth, K.: Konstruieren mit Konstruktionskatalogen: 1. Band: Konstruktionslehre, 3rd edn. Springer, Heidelberg (2000)
48. Sauer, T.: Ein Konzept zur Nutzung von Lösungsobjekten für die Produktentwicklung in Lern- und Anwendungssystemen. Ph.D. thesis, TU Darmstadt, VDI-Verlag, Düsseldorf (2006)
49. Scheffler, M.: Cost vs. Quality Trade-off for Gigh-density Packaging of Electronic Systems. Ph.D. thesis, Swiss Ferderal Institute for Technology, Eidgenössische Technische Hochschule, Zürich (2001)
50. Schmidt, A.: Wirkmuster zur Selbstoptimierung - Konstrukte für den Entwurf selbstoptimierender Systeme. Ph.D. thesis, Fakultät für Maschinenbau, Universität Paderborn, HNI-Verlagsschriftenreihe, Band 204, Paderborn (2006)
51. Sondermann-Wölke, C., Geisler, J., Sextro, W.: Increasing the Reliability of a Self-optimizing Railway Guidance System (2010)
52. Stahl, T., Voelter, M.: Model-driven Software Development: Technology, Engineering, Management. John Wiley & Sons, Hoboken (2006)
53. Steffen, D.: Ein Verfahren zur Produktstrukturierung für fortgeschrittene mechatronische Systeme. Ph.D. thesis, Fakultät für Maschinenbau, Universität Paderborn, HNI-Verlagsschriftenreihe, Band 207, Paderborn (2006)
54. Strube, G.: Wörterbuch der Kognitionswissenschaft. Klett-Cotta, Stuttgart (1996)
55. Thommen, J.P.: Managementorientierte Betriebswirtschaftslehre, 8th edn. Versus Verlag, Zürich (2008)
56. Tumer, I., Stone, R., Bell, D.: Requirements for a Failure Mode Taxonomy for Use in Conceptual Design. In: Proceedings of the International Conference on Engineering Design, Stockholm (2003)
57. Vaßholz, M., Gausemeier, J.: Cost-Benefit Analysis - Requirements for the Evaluation of Self-Optimizing Systems. In: Proceedings of the 1st Joint International Symposium on System-Integrated Intelligence, Hannover (2012)
58. Wilkinson, P., Kelly, T.: Functional Hazard Analysis for Highly Integrated Aerospace Systems. In: Proceedings of the Ground/Air Systems Seminar (1998)