Chapter 2
Development of Self-optimizing Systems

Abstract. In the development of self-optimizing systems, developers have to face different challenges, such as the multidisciplinarity of mechatronics, cross-linking between the subsystems, the lack of current knowledge in the fields of advanced mathematics and artificial intelligence, and increased dependability requirements. To support the developer in this challenging task adequately, the Collaborative Research Center 614 has developed a design methodology consisting of a reference process, methods and tools. The reference process is divided into two phases: the "Domain-Spanning Conceptual Design" and the "Domain-Specific Design and Development". In the first phase, the domain-spanning model-based aproach CONSENS (CONceptual design Specification technique for the ENgineering of complex Systems) is used to create a common understanding basis between the different domains involved. Based on the Principle Solution developed in this phase, the "Domain-Specific Design and Development" is planned and implemented. To ensure the development of dependable self-optimizing systems, domain-spanning and domain-specific dependability engineering methods can be used. The developer encounters a significant number of these methods, that have to be integrated into the process to obtain an individualized development process for the specific development task.

Jürgen Gausemeier and Mareen Vaßholz

The Collaborative Research Center 614 has developed a design methodology that expands existing ones for mechatronic systems such as the VDI guideline 2206 [12], the approach by Isermann (2008) [8] or Ehrlenspiel (2007) [4], the iPeM-Modell [1] or the V-Model by Bender (2005) [2] and supports developers by providing domain-spanning and self-optimization-specific methods and tools [9]. The methodology consists of a reference process, methods and tools for the development of self-optimizing systems, and is presented in detail in the book "Design Methodology for Intelligent Technical Systems" *D.M.f.I.T.S*, [7], Chap. 3.

During the development of self-optimizing systems, different domains, such as mechanical, electrical/electronic, software and control engineering and experts from advanced mathematics and artificial intelligence, are involved. Therefore, a domain-spanning development process is needed which supports common understanding of the product development activities and their cross-domain relationship for all participating developers. The development of self-optimizing systems (cf. Fig. 2.1) is basically structured into two main phases: the "Domain-spanning Conceptual Design" and the "Domain-specific Design and Development". The result of the first phase is the specification of a Principle Solution for the system to be developed. It describes the basic structure and operational mode of the system, as well as its desired behavior, computer-internally as partial models. In particular, the specification of the Principle Solution contains the description of the System of Objectives (cf. Sect. 2.1). Based on the Principle Solution, a first analysis of the dependability can be carried out.

This specification of the Principle Solution constitutes the basis for the Domain-specific Design and Development (cf. Sect. 2.2). During this phase, the domains involved work in parallel and use their domain-specific methods, models and tools. The partial solutions developed by the domains involved are continuously integrated into the overall solution with model transformation and synchronization techniques [11]. During the Domain-specific Design and Development, the ability of the system to adapt its behavior during operation is implemented in the domains. For example, multiobjective optimization is used in the domain control engineering to enable the system to adapt its behavior by changing the control parameter during operation.

These steps particularly require the involvement of experts of advanced mathematics and artificial intelligence. The optimization is specific to self-optimizing systems, as the main focus is on the implementation of the self-optimization process. In addition, system tests are performed; both virtual and real prototypes are used for

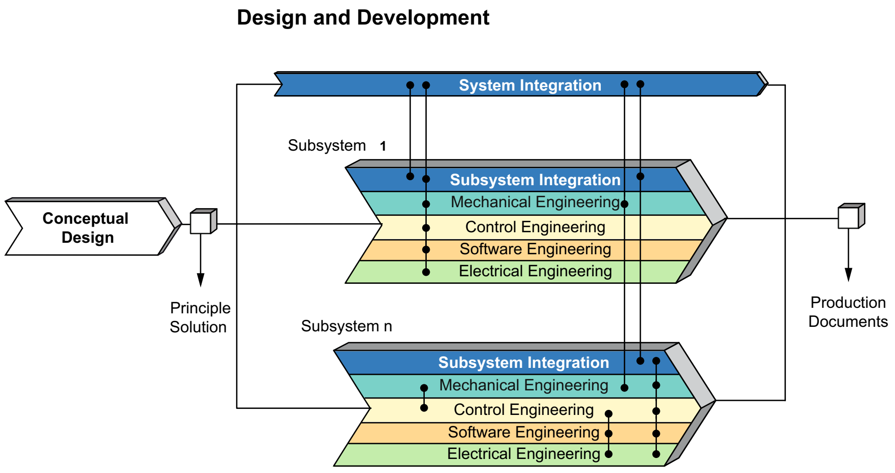

Fig. 2.1 Macro cycle for the development of self-optimizing systems

this purpose. The result of the Design and Development phase is engineering data such as manufacturing drawings and part lists.

The reference process for the development of self-optimizing systems is recommendatory and gives an overview of the tasks that have to be performed during the development. It is based on our experiences from the development of the RailCab and its function modules. It serves as basis for the application to a specific development task and company in form of an implementation model. This model provides a detailed process sequence according to the project, the system type and the environment of the development project. It consists of process steps from the reference process and builds the starting point for the project management. For example the implementation model for the RailCab consists of about 850 steps and 900 development objects. During the development project execution, deviations and changes from the planned development process can occur, for example due to changing development objectives such as time and costs. To support the management during the project execution to react to these changes we adopted the self-optimization approach to the management of the development process (cf. *D.M.f.I. T.S*, [7], Sect. 3.5). In the following sections the reference process is described in more detail.

2.1 Domain-Spanning Conceptual Design

The aim of the "Domain-spanning Conceptual Design" is to create a common understanding of the system between the domains involved. The main result of this phase is the Principle Solution, which is described by the specification technique CONSENS.

There are eight separate but highly interrelated aspects that need to be covered by the Principle Solution. Computer-internally, these aspects are represented as *partial models*. The eight aspects and their respective partial models are shown in Fig. 2.2 (cf. [6]). While each partial model represents only one individual aspect of the system, together they form a coherent model of the complete system [6]. The partial models are:

Environment:
: This model describes the environment of the system to be developed and the system's embedment into the environment. Relevant spheres of influence and influences will be identified and interactions between them will be examined [6].

Application Scenarios:
: These scenarios concretize the behavior of the system in a particular state and situation. Application Scenarios characterize a problem which needs to be solved in certain cases and also briefly outline the possible solution [6].

Requirements:
: This partial model presents an organized collection of requirements that need to be fulfilled during product development. Every requirement is described textually and, if possible, concretized by attributes and their characteristics [6].

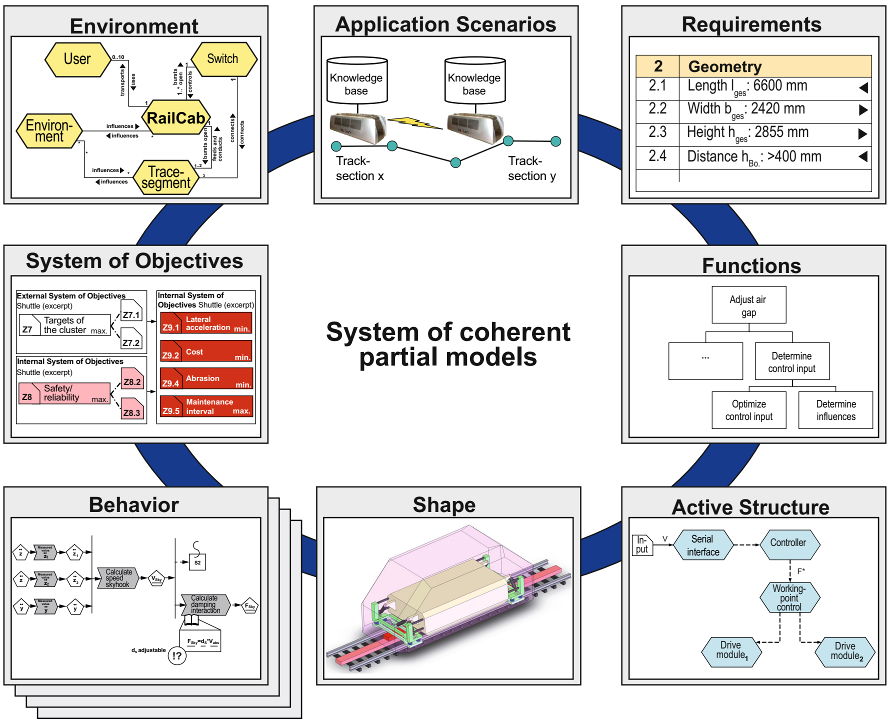

Fig. 2.2 Partial models of the domain-spanning description of the Principle Solution of a self-optimizing system [6]

Functions:
A function is the general and required relationship between input and output parameters with the aim to fulfill a task. Functions are implemented using solution patterns and their concretizations. Functions should be specified in a hierarchical manner [6].

Active Structure:
This partial model describes the system elements (e.g. drive and brake modules, energy management) and their attributes, as well as their relations (energy, material or information flow). Incoming parameters are also described (e.g. comfort) [6].

Shape:
The Shape is usually created by a 3D CAD-System and gives a first impression of the physical appearance of the system [6].

Behavior:
The partial model Behavior is subdivided into the aspects Behavior – State and Behavior – Activities. The aspect Behavior–State defines the states of the system and the state transitions. The state transitions describe the reactive behavior of the system towards incoming events, which can also be system activities. The

aspect Behavior – Activities describes the logical sequence of activities which are carried out by the different system elements. Activities describe how functions are executed within different system states [6].

System of Objectives:

This partial model includes external, inherent and internal objectives, as well as the connections between them. Internal objectives are achieved by the system itself during operation mode and as a consequence represent the system's intentionality [6] (see also Sect. 1.1).

In particular, the partial model System of Objectives is of high importance for self-optimizing systems: it describes the objectives of the system during operations, their hierarchical relationship, and potential conflicts between objectives [10]. The specification of the Principle Sution provides a holistic domain-spanning description of the self-optimizing system and forms the basis for the communication and cooperation between the developers from the domains involved. As such, it enables them to avoid design mistakes and corresponding iteration loops in later development phases, which would otherwise occur due to differences in understanding of the system [3].

To develop the Principle Solution the reference process for the Domain-spanning Conceptual Design consists of four main phases: "Planning and Clarifying the Task", "Conceptual Design on the System Level", "Conceptual Design on the Subsystem Level" and "Concept Integration" (cf. Fig. 2.3).

In "Planning and Clarifying the Task", the design task of the system and the requirements are identified and evaluated. The results are the List of Requirements, the Environment model, the recommended product structure type and the design rules for this, as well as the Application Scenarios.

Based on previously determined requirements the system must fulfill, the main functions of the system are identified and set into a function hierarchy in the "Conceptual Design on the System Level". Each function has to be fulfilled to satisfy the requirements; therefore solution patterns are sought which can execute the desired functions. Within a morphologic box, the solution patterns are combined to obtain consistent solution variants. These are evaluated and the best one is chosen and integrated into the Principle Solution on the system level. The resulting solution does

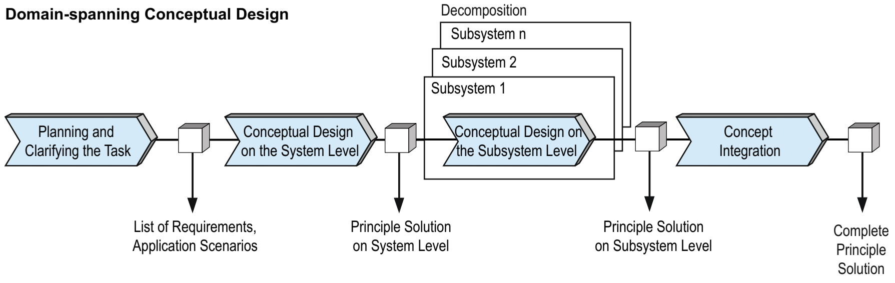
Domain-spanning Conceptual Design
Decomposition
Subsystem n
Subsystem 2
Subsystem 1
Planning and Clarifying the Task
Conceptual Design on the System Level
Conceptual Design on the Subsystem Level
Concept Integration
List of Requirements, Application Scenarios
Principle Solution on System Level
Principle Solution on Subsystem Level
Complete Principle Solution

Fig. 2.3 The four main phases of the Domain-spanning Conceptual Design

not have to be self-optimizing at this stage. The consistent grouping of solution patterns allow the modeling of the Active Structure, an initial construction shape and the System Behavior.

At this point the potential for the use of self-optimization can be identified, based on contradictions within the System of Objectives. These can be solved either by compromise or by self-optimization. In the latter case, the System of Objectives is developed, and the List of Requirements and the function hierarchy are extended. In preparing the self-optimization concept, the self-optimization processes will eventually be defined, whether there is the necessary absence of conflicts of the self-optimization process will be analyze, and the boundary conditions within which self-optimization has to be working in, will be defined as well. For the newly integrated cognitive functions, solution patterns for self-optimization are identified and integrated into the partial models of the Principle Solution. The Principle Solution on the system level is the result of this phase.

To describe the system in detail, it is modularized and a Principle Solution for each single subsystem is developed in the "Conceptual Design on the Subsystem Level". This procedure corresponds to the Conceptual Design on the system level, starting out with planning and clarification of the task. This phase results in the Principle Solutions on the subsystem level. This process can be conducted recursively, because the subsystem itself can be a system with subsystems, and so forth.

The Principle Solutions of the subsystems are integrated into one Principle Solution during the phase "Concept Integration" to represent the whole system. Afterwards the thus-specified system is analyzed regarding its dynamical behavior and dependability. In this analysis phase, contradictions between the Principle Solution on the subsystem level are identified. Again, it must be determined whether these contradictions can be solved by self-optimization. At the end of this step, a technical-economical evaluation of the solution is made. The result of this phase is the Principle Solution of the whole system, which serves as a starting point for the subsequent Domain-specific Design and Development. This is carried out simultaneously in the specific domains (mechanical engineering, electrical/electronic engineering, control engineering and software engineering) [5].

In this early development phase, the consistency of the system is ensured by this domain-spanning approach. Development failures and therefore time-consuming and costly iteration loops can be avoided. To eliminate failures in this early stage, the dependability of the system is analyzed at the end of the concept integration. A number of methods can be used here to analyze the Principle Solution, such as the early Failure Mode Effect Analysis (FMEA) in combination with the Fault Tree Analysis (FTA) (cf. Sect. 3.1.1). To ensure the dependability of the self-optimization itself, the Multi-Level Dependability Concept can be applied (cf. Sect. 3.1.2). During the Design and Development, the Multi-Level Dependability Concept is realized in the domains where the failure has been identified (cf. Sect. 3.2.1). As long as failures or failure modes cannot be eliminated, the Principle Solution is tweaked and readjusted. The resulting Principle Solution forms the basis for the following Domain-specific Design and Development.

2.2 Domain-Specific Design and Development

As mentioned above, the Principle Solution is the starting point for the Domain-specific Design and Development. The Principle Solution contains the information that forms the basis for domain-specific development tasks. The transition of the Principle Solution to the domains involved is described briefly in *D.M.f.I.T.S*, [7], Sect. 5.1. In classical mechatronic development processes, the four domains mechanical, control, software, and electrical/electronic engineering are involved. For self-optimizing systems the optimization during operation needs to be additionally taken into account in the domains. The domains involved use their specific methods, tools and modeling languages to concretize the system. This phase is characterized by a high coordination effort; to ensure the consistency of the system, the results of the domains are continuously integrated. For this purpose, model transformation and synchronization techniques are used. The integrated system is tested as a virtual prototype to identify faults. This allows a short response time concerning design failures and therefore reduces time and cost-intensive iterations.

The reference process for the Design and Development of self-optimizing systems shows the ideal approach in which the particularities that need to be considered for the development of a self-optimizing system are pointed out. It is based on the development of the RailCab and its function modules. To reduce development time and therefore costs, the domains work in parallel where possible. Within the process, important synchronization points are depicted, where the domains exchange their results and get informations needed for further development. Even though the process gives the impression of being overly stringent, iterations can emerge in particular at these synchronization points, although they are possible at every stage of the process. The application of the presented approach for the Design and Development of a specific development task and company must be developed individually.

The approach presented here is recursive and conducted on different hierarchy levels of the system. The system itself consists of subsystems, whereby the subsystems are also systems themselves that consist of subsystems, and so forth. Fig. 2.4 presents a schematic representation of the process for one subsystem. (A detailed description is given in *D.M.f.I.T.S*, [7], Sect. 3.3.) It is intended to supplement the existing Design and Development process of the domains involved. In the following, we will give an overview of the tasks belonging to the various domains.

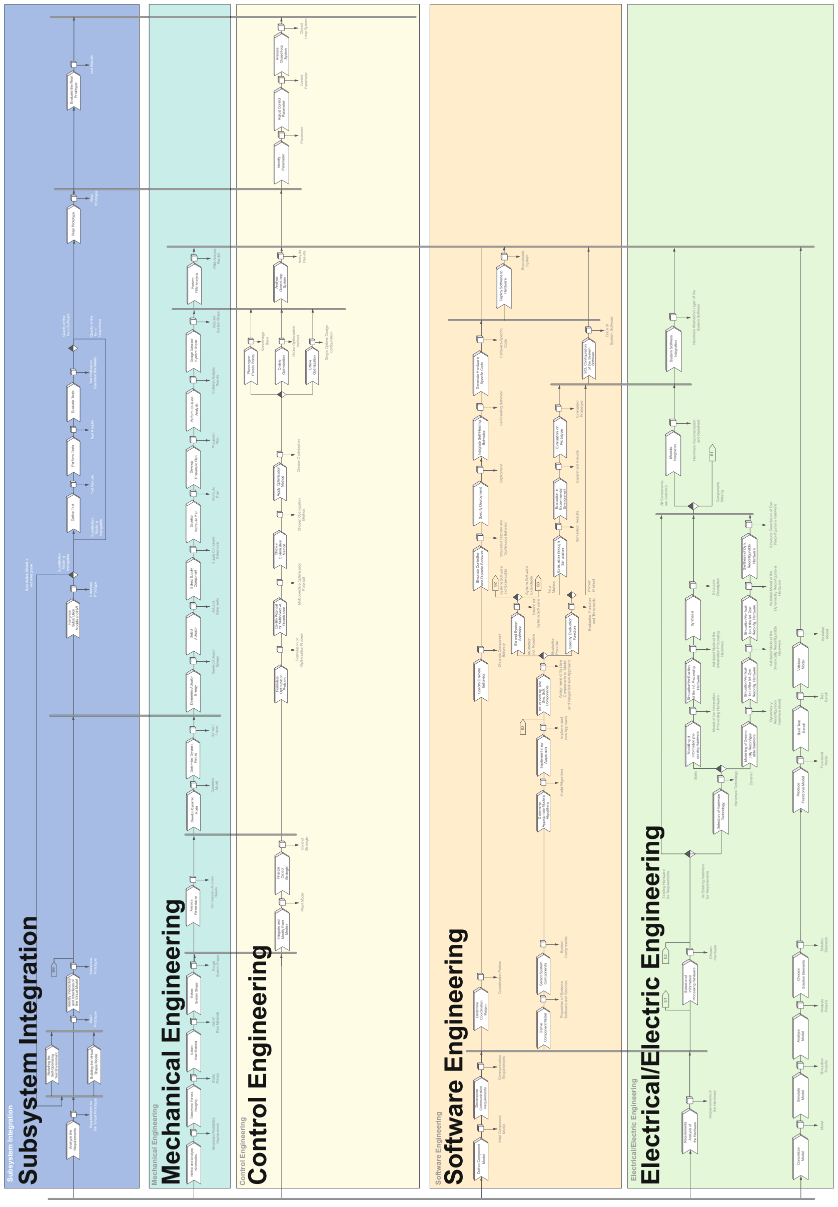

Fig. 2.4 Schematic representation of the Domain-specific Design and Development

Mechanical engineering:

The domain mechanical engineering has as its goal the Design and Development of the system shape of the self-optimizing system. Its starting point are the partial models Requirements, Active Structure and Shape of the Principle Solution. Based on these partial models, solution elements are selected and a first Shape model is built. This model serves as input for the domains control and electrical engineering, as well as for (sub)system integration. Using the Shape model, the domain control engineering is able to finalize the control strategy and the domain electrical engineering can develop a first model of the power electronics. Furthermore, a first virtual prototype can be modeled in the (sub)system integration. The result is a dynamic model that enables a simulation of the dynamic behavior. Based on these results, the Shape model can be further developed using a 3D-CAD software tool. The results of the work in this domain are the mechanical structure and derived manufacturing documents.

Control engineering:

The aim of control engineering is the design of the controller to guarantee the desired dynamical behavior of the self-optimizing system. Based on the Shape model from mechanical and electrical engineering, the control strategy can be finalized. The given dynamic model is the input for the optimization; resulting optimization strategies are integrated into the control strategy and the closed-loop system is analyzed. The analysis results are then passed along to the (sub)system integration and can be tested in the virtual prototype as hardware-in-the-loop-tests. Based on these tests, necessary parameters can be identified, control parameters adjusted, and the closed-loop can be analyzed again. These adjustments need to be done again after testing the real prototype; only then can the control strategy be finalized and implemented into the real system.

Software engineering:

The system software and discrete software are developed in the domain software engineering. The system software provides services for the discrete software; its development process is structured into three main phases. First, the required methods need to be identified and designed. Second, the developed methods need to be implemented; third they need to be evaluated.

In the first phase, the components of the system that will contain system software are selected. This information is derived from the Requirements of the Principle Solution. For each of these components, appropriate models or algorithms need to be determined if not already available. For example, self-healing may be implemented to improve the dependability of the system software (cf. Sect. 3.2.12). In the case of a high demand for processing power and memory, virtualization can reconcile the opposing requirements (cf. Sect. 3.2.15). Following this in the second phase, the new components need to be integrated into the system software and, if applicable, the system software is extended. If this extension is not possible, an iteration is necessary and the new approach needs to be adjusted or further models with their respective algorithms need to be developed. In the third phase, the system software components are assigned, including the specification of evaluation functions. The three phases are run through until all requirements are met.

During the Conceptual Design of the discrete software, a component structure is derived from the Requirements. The component structure of the discrete software is modeled by the Active Structure, while the communication between the components is modeled by Modal Sequence Diagrams.

During Design and Development, we use MechatronicUML (cf. *D.M.f.I.T. S*, [7], Chap. 5) to design the discrete software. In MechatronicUML, the component structure of the Active Structure is further refined and coordination patterns (cf. *D.M.f.I.T.S*, [7], Chap. 5) are derived from the Modal Sequence Diagrams; these coordination patterns specify communication protocols. Next, the discrete internal component behavior and the reconfiguration behavior are specified. We use model checking to guarantee that this behavior satisfies the requirements (cf. *D.M.f.I.T.S*, [7], Chap. 5); model checking also supports checking the integration of control engineering (cf. Sect. 3.2.10). The deployment of the software is based on the description of the hardware that is provided by electronic engineering. A hazard analysis is also performed at this stage to guarantee that the occurance of severe hazards remains below a specific probability limit. Hazards whose occurrence probabilities are higher may be reduced by self-healing operations, which are specified by means of reconfigurations. Then, it is analyzed whether the self-healing operations are able to achieve the intended probability reduction (cf. Sect. 3.2.8). Afterwards, the reconfiguration behavior is verified. The system computes reconfiguration plans at runtime that avoid unsafe configurations (cf. Sect. 3.2.9); this planning process is simulated at design time to validate its reliability. In addition, we apply online model checking (cf. Sect. 3.2.14). If all the requirements are fulfilled, the controller and the discrete behavior are simulated using the virtual prototype in the (sub)system integration (cf. *D.M.f.I. T.S*, [7], Chap. 5). If the simulation displays correct functionality, the hardware-specific code is generated and deployed on the hardware.

Electrical/electronic engineering:

Self-optimizing systems demand high flexibility in order to be able to adapt their behavior during operation. Dynamically reconfigurable hardware can ensure this flexibility. Furthermore, such systems demand high power transmission. Therefore, in the domain electrical/electronic engineering, both microelectronic devices for information processing and power electronics need to be developed. Input for the development of the microelectronics is given by the Requirements of the Principle Solution. After analyzing these requirements, a set of appropriate information processing hardware components is collected. In case the requirements cannot be fulfilled by existing hardware, additional hardware, for example embedded processor architecture or FPGA (Field Programmable Gate Array) technology is chosen. At this point the developer needs to decide whether the hardware to be developed consists only of static components, or of both static and dynamic ones. In the first case, the standard design flow from the Y-diagram for electronic engineering is used, whereas if dynamic reconfiguration of the hardware is considered necessary, the procedure needs to be extended with respect to the characteristics of these architectures. The concrete process is part of the design environment INDRA (Integrated Design Flow for Reconfigurable Architectures, cf. *D.M.f.I.T.S*, [7], Chap. 5. First, the dynamic reconfigurable hardware is modeled; while doing this, the requirements that are derived from system software are taken into account. The model is simulated and verified and afterwards synthesized until all of the components have been integrated. Finally, the system software can be ported to the new architecture. To increase the dependability of the system, self-healing via dynamic reconfiguration can be implemented (cf. Sect. 3.2.13). For the development of the power electronics, the partial model Requirements and Active Structure serve as input. Based on these, the type of electrical drive is selected and its capacity is analyzed. In the next step, the appropriate engine is chosen. This information is passed on to mechanical engineering and is integrated into the system model, after which the thermal resilience of the engine is tested and the data sheet of the electrical engine is derived. Afterwards, the power electronics can be defined. The results of these tasks are integrated into the overall system.

(Sub)System Integration:

To ensure the consistency of the domain-specific models, the results of the domains are integrated continuously into an interdisciplinary system model that is based on the Principle Solution; additionally the functionality of the system model is secured by a virtual prototype. To do so, a self-optimizing test bench and the environment are modeled. The basis for the virtual prototype are given by the Requirements of the Principle Solution. The virtual prototype is first modeled based on the basic Shape model developed by the domain mechanical engineering. This prototype is improved and expanded continuously over the course of development process by integrating the results from other domains. To make the simulation of the virtual prototype possible, the interactions and interfaces of the model need to be identified. For the system test, the test cases and a metric are first defined. Based on these cases, the virtual prototype is tested (cf. Sect. 3.2.3). The quality of the test results can be evaluated using the pre-defined metric. If the quality of the tests is not sufficient new tests are defined, performed and evaluated until the quality is sufficient. Then the test results are returned to the domains. When all subsystems are fully implemented into the virtual prototype and the test results are failure-free, the real prototype can be built and evaluated.

Within the development process, the dependability of the self-optimizing system needs to be continuously ensured. This can be done with dependability-specific methods such as those presented in Chap. 3. The Methodology for the Selection of Dependability Methods for the Development of Self-Optimizing Systems presented in Sect. 3.3 supports the developer in choosing the appropriate dependability method and in integrating it into the development process.

References

1. Albers, A.: Five Hypotheses about Engineering Processes and their Consequences. In: Proceedings of the 8th International Symposium on Tools and Methods of Competitive, Ancona, IT (2010)
2. Bender, K.: Embedded Systems – Qualitätsorientierte Entwicklung. Springer, Berlin (2005), doi:10.1007/b138984
3. Dorociak, R., Gaukstern, T., Gausemeier, J., Iwanek, P., Vaßholz, M.: A Methodology for the Improvement of Dependability of Self-Optimizing Systems. Production Engineering – Research and Development 7(1), 53–67 (2013), doi:10.1007/s11740-012-0425-3
4. Ehrlenspiel, K.: Integrierte Produktentwicklung, 3rd edn. Carl Hanser Verlag, Munich (2007)
5. 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), doi:10.1007/s00163-008-0058-x
6. Gausemeier, J., Rammig, F.J., Schäfer, W. (eds.): Selbstoptimierende Systeme des Maschinenbaus. In: HNI-Verlagsschriftenreihe, vol. 234. Heinz Nixdorf Institute, University of Paderborn, Paderborn (2009)
7. Gausemeier, J., Rammig, F.J., Schäfer, W. (eds.): Design Methodology for Intelligent Technical Systems. Lecture Notes in Mechanical Engineering. Springer, Heidelberg (2014), doi:10.1007/978-3-642-45435-6_2
8. Isermann, R.: Mechatronische Systeme – Grundlagen. Springer, Berlin (2008), doi:10.1007/978-3-540-32512-3
9. Kahl, S., Gausemeier, J., Dumitrescu, R.: Interactive Visualization of Development Processes. In: Proceedings of the 1st International Conference on Modelling and Management of Engineering Processes (2010)
10. Pook, S., Gausemeier, J., Dorociak, R.: Securing the Reliability of Tomorrow's Systems with Self-Optimization. In: Proceedings of the Reliability and Maintainability Symposium, Reno, NV, US (2012)
11. Rieke, J., Dorociak, R., Sudmann, O., Gausemeier, J., Schäfer, W.: Management of Cross-Domain Model Consistency for Behavioral Models of Mechatronic Systems. In: Proceedings of the 12th International Design Conference, Dubrovnik (2012)
12. Verein Deutscher Ingenieure (VDI): VDI 2206 – Entwicklungsmethodik für mechatronische Systeme. Technical Guideline (2004)