Chapter 1
The Paradigm of Self-optimization

Michael Dellnitz, Roman Dumitrescu, Kathrin Flaßkamp, Jürgen Gausemeier, Philip Hartmann, Peter Iwanek, Sebastian Korf, Martin Krüger, Sina Ober-Blöbaum, Mario Porrmann, Claudia Priesterjahn, Katharina Stahl, Ansgar Trächtler, and Mareen Vaßholz

Abstract. Machines are ubiquitous. They produce, they transport. Machines facilitate and assist with work. The increasing fusion of mechanical engineering with information technology has brought about considerable benefits. This situation is expressed by the term mechatronics, which means the close interaction of mechanics, electrics/electronics, control engineering and software engineering to improve the behavior of a technical system. The integration of cognitive functions into mechatronic systems enables systems to have inherent partial intelligence. The behavior of these future systems is formed by the communication and cooperation of the intelligent system elements. From an information processing point of view, we consider these distributed systems to be multi-agent-systems. These capabilities open up fascinating prospects regarding the design of future technical systems. The term self-optimization characterizes this perspective: the endogenous adaptation of the system's objectives due to changing operational conditions. This resuls in an autonomous adjustment of system parameters or system structure and consequently of the system's behavior. In this chapter self-optimizing systems are described in detail. The long term aim of the Collaborative Research Centre 614 "Self-Optimizing Concepts and Structures in Mechanical Engineering" is to open up the active paradigm of self-optimization for mechanical engineering and to enable others to develop these systems. For this, developers have to face a number of challenges, e.g. the multidisciplinarity and the complexity of the system. This book povides a design methodology that helps to master these challenges and to enable third parties to develop self-optimizing systems by themselves.

J. Gausemeier et al. (eds.), *Design Methodology for Intelligent Technical Systems*, Lecture Notes in Mechanical Engineering,
DOI: 10.1007/978-3-642-45435-6_1,

1.1 From Mechatronics to Intelligent Technical Systems

Roman Dumitrescu, Jürgen Gausemeier, Peter Iwanek, and Mareen Vaßholz

Machines are ubiquitous, from white goods to medical devices and transportation systems. Their purpose is to make life easier. The increasing integration of information and communication technology in the field of conventional mechanical engineering implies considerable potential for innovation. Most modern products created in the field of mechanical engineering and related areas such as automobile technology, already rely on the close symbiotic interaction between mechanics, electrics/electronics, control engineering and software technology, which is aptly expressed by the term mechatronics [29] . Mechatronics – a portmanteau of the words mechanics and electronics [10] – represents the potential for the development of fundamentally new solutions in the area of mechanical engineering and related disciplines, which in turn can significantly improve the cost-benefit ratio of familiar products and also stimulate the innovation of new products. In the following, the way from Mechatronic Systems across Adaptive Systems towards Intelligent Technical Systems will be outlined.

Mechatronic Systems

The aim of mechatronics is to improve the behavior of a technical system by using sensors and information from humans or other systems to obtain information on the environment and also on the system. Thus, they can respond to changes in their environment, detect critical operating states and control processes which are difficult to control in real-time by humans [2, 22, 34]. Mechatronic systems are distinguished by the functional and/or spatial integration of sensors, actuators, information processing and a basic system. The basic structure of a mechatronic system is shown in Fig. 1.1. In general, mechatronic systems can also be composed of subsystems which themselves are mechatronic systems (cf. Sect. 1.3) [2].

The **basic system** is commonly a mechanical structure. Generally any desired physical system is conceivable as a basic system, so that it is even possible to hierarchically represent mechatronic systems structurally (cf. Sect. 1.3) [2]. The relevant physical (continuous) values of the basic system or its environment are measured using **sensors**. Sensors in this case can be physically present measured-value pickups or straightforward software sensors [2] (so-called "observers", see for example [34]). The sensors supply the input variables for **information processing**, which in most cases takes place digitally, i.e. discretely in terms of value and time. The information processing unit determines the necessary changes to the basic system using the measurement data as well as the user specifications (human-machine interface) and also available information from other processing units (communication system). The information processing unit is often realized as an electronic microprocessor, which realize open-loop and closed-loop control functions. The behavior adaptation of the basic system will be caused by **actuators**. In general, an

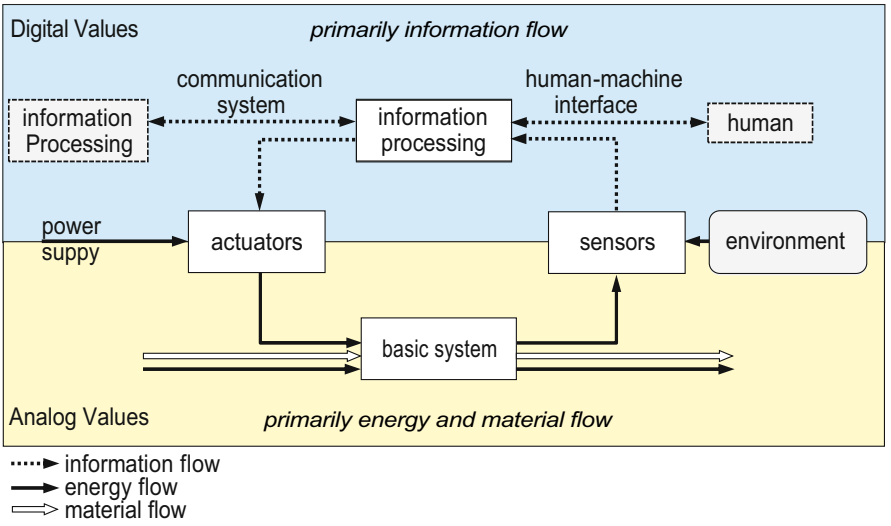

Fig. 1.1 Basic structure of a mechatronic system [2]

actuator converts energy into motion, for example by using a motor or a hydraulic cylinder.

The relationships between the basic system, sensors, information processing and actuators are represented as flows (cf. Fig. 1.1). For example there is a flow from the sensors (e.g. informations of the environment) to the information processing unit of the system. In principle, three types of flows can be distinguished: information flows, energy flows and material flows [44], whereby the information flows are frequently also referred to as signal flows.

- **Material flows:** Examples for materials which flow between units of mechatronic systems are for example gases, fluids, solids, dust and also raw products, objects being tested or objects being treated.
- **Energy flows:** Energy is understood to be any form of energy, such as for example mechanical, thermal, electrical, optical energy, or force and electric current.
- **Information flows:** Information exchanged between units of mechatronic systems or components are, for example measured variables, control pulses, or data.

To serve customer needs and to create cost and energy efficient systems, mechatronic systems have to fulfill increasingly advanced functions and requirements. These requirements can vary or conflict depending on the situation. An example is the consideration of execution speed and resource use: if the application situation requires the fastest possible execution of a task (e.g. an important rush order), the requirement resource efficiency is considered to be less important. In the development of conventional mechatronic systems, the developer has to decide on a compromise between two competing requirements. Thus, the controller of the system is

designed for acceptable behavior under specific circumstances. A first step to solve this challenge is to use adaptive control strategies [3, 8].

Adaptive Systems

Adaptive systems use adaptive control strategies, that make it possible to adjust the controller of the system in real time by using **adaptation algorithms**. These algorithms help to achieve a desired level of control system performance for situations in which the parameters of the basic system are a priori unknown and/or change in time [39]. Therefore adaptive systems form a necessary step towards intelligent technical systems. Adaptive controllers enable higher levels of adaptability, but still remain a compromise for anticipated situations [3]. The improvement of communication and information technology opens up more and more fascinating perspectives, which go far beyond current standards of mechatronics: mechatronic systems having inherent partial intelligence.

Intelligent Technical Systems

Future mechanical engineering systems will comprise of configurations of system elements with inherent partial intelligence. These system elements are able to realize functions such as "to share knowledge", "to coordinate behavior" or "to learn from experience". Those functions are also typical for cognitive systems and are known as cognitive functions [16]. Systems having these cognitive functions, are also called intelligent technical systems. There are four central properties, which describe intelligent technical systems [14, 15]:

- **Adaptive:** Adaptive systems react autonomously and flexibly on changing operation conditions. They are able to learn and optimize their behavior at runtime.
- **Robust:** These systems are able to behave "acceptable", even in situations which have not been considered during the development phase of the system. Uncertainties can be compensated to certain extends.
- **Foresighted:** Based on gained experience, these systems have the ability to recognize emerging states and situations. Thus they are able, to spot possible dangers and accordingly change their behavior.
- **User-friendly:** The systems are able to adapt their behavior to the specific user. They interact closely with the user and their behavior is always comprehensible.

Keywords as "Things that Think", "Cyber-Physical Systems", "Industry 4.0" or "Self-optimization" express this perspective on intelligent technical systems. These systems exceed the functionality of mechatronic systems and set new requirements on design methodologies. Therefore the gap between system complexity and performance of the design methodology is increasing (see Fig. 1.2). To realize intelligent technical systems, such as self-optimizing systems, not only the domains mechanical, software, control and electrical/electronic engineering have to be considered, but also experts from higher mathematics and artificial intelligence have to

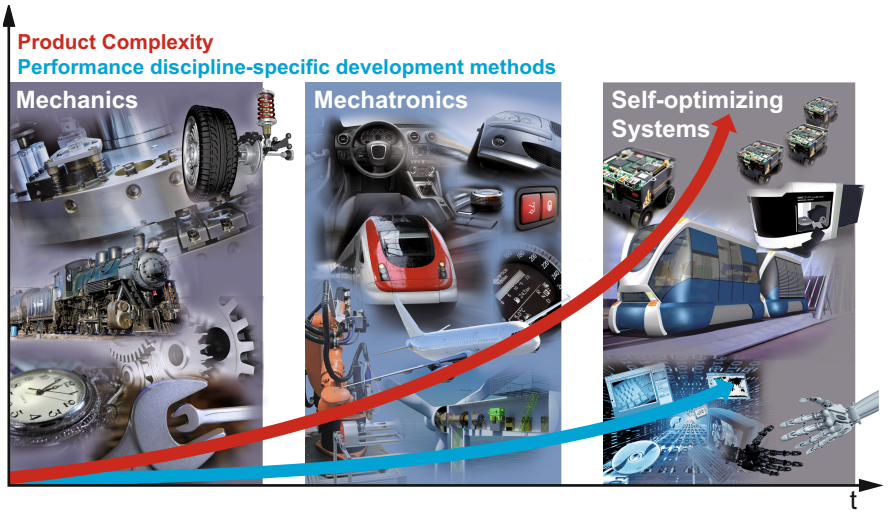

Fig. 1.2 Complexity of self-optimizing systems versus performance of design methodologies

be involved in the development process. This book provides a design methodology for self-optimizing systems consisting of a reference process, methods and tools that master the shortcomings of existing design methodologies. This methodology closes the gap between system complexity and performance of the design methodology, to enable developers who are not specifically trained in higher mathematics and artificial intelligence to develop self-optimizing systems.

1.2 Introduction to Self-optimization

Roman Dumitrescu, Jürgen Gausemeier, and Peter Iwanek

The Collaborative Research Center (CRC) 'Self-Optimizing Concepts and Structures in Mechanical Engineering' defines **self-optimization** as follows:

"Self-optimization describes the ability of a technical system to endogenously adapt its objective regarding changing influences and thus adapt the system's behavior in accordance to the objectives. The behavior adaptation may be performed by changing the parameters or the structure of the system. Thus self-optimization goes considerably beyond the familiar rule-based and adaptive control strategies; Self-optimization facilitates systems with inherent "intelligence" that are able to take action and react autonomously and flexibly to changing operating conditions." [3]

Figure 1.3 shows the key aspects and the mode of operation of a self-optimizing system. Factors that **influence the technical system** originate in its surroundings (environment, users, etc.) or from the system itself. They can support the system's

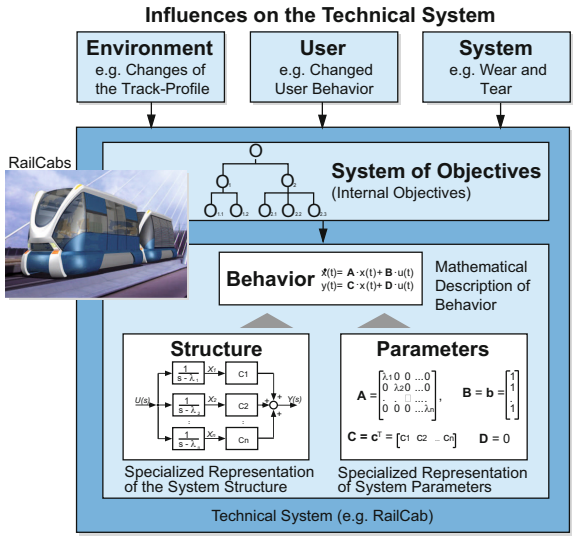

Fig. 1.3 Aspects of self-optimizing systems [21, 25]

objectives or hinder them. Influences from the **environment**, for example such as strong winds or icy conditions, are unstructured and often unpredictable. If they hinder the system fulfilling its pursued objectives they are called disturbance variables. The **user** can influence the system, for instance by choosing preferred objectives. It is also possible that the **system** itself or other technical systems will influence the system's objectives, for example if mechanical components are damaged, the objective "max. safety" has to be prioritized [20].

The self-optimizing system determines its current pursued objectives (**System of Objectives**) on the basis of the encountered influences on the system for example the environment, the user or the system itself. New objectives can be added, existing objectives can either be rejected or the priority of objectives can be modified during system operations. Therefore, the system of objectives and its autonomous change is the core of self-optimization [3, 15]. Objectives can be distinguished between external and inherent objectives. External objectives are set from the outside of the self-optimizing system, by other systems or by the user (e.g. for a driving module this could be "max. comfort"). Inherent objectives reflect the design purpose of the self-optimizing system. An inherent objective of a driving module can be for example "max. energy efficiency". Objectives build a hierarchy and each objective can thus be refined by sub-objectives (e.g. "min. energy consumption" is a possible sub-objective of "max. energy efficiency"). Inherent and external objectives that are pursued by the system at a given moment during its operation are called internal objectives [13].

Adapting the objectives, leads to a continuous adjustment of the **system behavior** to the occuring situation. This is achieved by adapting **parameters** or reconfiguring the **structure** (e.g. adapting control strategies) [15]. The self-optimization process consists of the following three actions [23]:

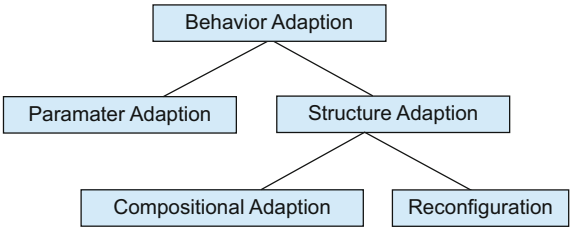

Fig. 1.4 Behavioral adaptation in a self-optimizing system by structure and/or parameter adaptation [3]

1. **Analyzing the current situation:** The current situation includes the current state of the system as well as all observations of the environment that have been carried out. Observations can also be made indirectly by communication with other systems. Furthermore, a system's state contains previous observations that were saved. One basic aspect of this first step is the analysis of the fulfillment of the objectives [3].

2. **Determining the system's objectives:** The system's objectives can be extracted by choice, adjustment, and generation. By choice means the selection of one alternative output of a predetermined quantity of possible objectives; the adjustment of objectives means the gradual modification of existing objectives respective to their relative weighting. Generation means, if new objectives are being created that are independent from the existing ones [3].

3. **Adapting the system behavior:** The changed system of objectives demands an adaptation of the behavior of the system and its components. As mentioned before, this can be realized by adapting the parameters and, if required, by adapting the structure of the system. The different types of behavior adaptation strategies are shown in Fig. 1.4. Parameter adaptation means for example changing a control parameter. Structure adaptations affect the arrangement of the system elements and their relationships. Here we distinguish between reconfiguration, which changes the relationships between a fixed set of available elements, and compositional adaptation, in which new elements are integrated into the existing structure or existing elements are removed from it [20]. The self-optimization process leads, according to changing influences, to a new system state. Thus a state transition takes place [3]. The behavior adaptation finally concludes the self-optimization process.

The self-optimization process takes place if the three actions are performed repeatedly by the system. The three actions do not need to be performed in a specified sequence. For example, within the scope of planning, different situations are considered and according to the situations, the objectives are adapted. This results in repeated situation analysis, based on the determination of objectives. Thus, the self-optimization process is executed, if the situation of the system is changed or a planning for possible system scenarios is performed [3].

Thus, self-optimization can be considered as an extension of classical and advanced control engineering [8]. In order to provide an optimal conformity to the

environment of the system at any time, self-optimizing systems utilize implemented adaptation strategies, instead.

To control self-optimizing systems, a consistent structuring of the information processing is needed. We distinguish between the macro structure of the mechatronic system and the structure of the information processing, represented by the Operator-Controller-Module. These structures will be explained in detail in the following section.

1.3 Architecture of Self-optimizing Systems

Martin Krüger and Ansgar Trächtler

Two types of structuring are presented. First, a description is given on how the entire system can be divided into subsystems or modules according to their function within the system. Such a decoupling naturally leads to a hierarchical ordering of the modules, with simpler modules on the lowest level and the entire system on the topmost level [40]. Second, the Operator-Controller-Module (OCM), a multi-level architecture, is introduced [32]. It includes all of the types of information processing which are necessary to realize an intelligent system: classical quasi-continuous controllers, discrete or event-based methods like error analysis and monitoring concepts as well as methods for cognitive capabilities, e.g. learning or optimization algorithms, to name but a few. Both structuring types complement one another and can be used in combination.

1.3.1 Structure of Self-optimizing Mechatronic Systems

One main step in the design of self-optimizing systems is to develop a hierarchy of functions based on the system requirements, see 4.1 for more details. A self-optimizing system can be divided into subsystems using this hierarchy of functions. The first step is to create a hierarchy of *motion functions* which describe the controlled motion of bodies, c.f. [3, 31]. Each motion function of the hierarchy can be realized by one of the three structuring elements:

Mechatronic Function Module (MFM): The MFMs are the basic elements of the entire mechatronic system. Each MFM includes sensors, actuators, information processors and the basic mechanical system. The motion of the mechanical system is measured using sensors and can be controlled by means of actuators. The control input is computed by the information processing. The actuators of a MFM can again be given by another MFM.

Autonomous Mechatronic System (AMS): The AMS is on the top level of an actual mechatronic system. It is associated with the complete mechanical structure of the physical element and thus forms the top level of the mechanical structure. Besides the associated mechanical structure, the AMS includes sensors and information processing elements. There is no need for actuators in an AMS. The actuating elements are given by the underlying MFMs and are coordinated by the AMS.

Networked Mechatronic System (NMS): NMS elements make up the top level of the hierarchy. A NMS is comprised of information processing and sensors. However, the CNS is made up of several AMS, which are linked by signals alone. The NMSs do not necessarily have their own physical representation in terms of a data-processing hardware. The function of the NMS might be implemented in the AMS, so that the NMS is generated whenever several AMS are interconnected.

Figure 1.5 exemplarily shows the hierarchical structure of the RailCab System which is described in detail in Sect. 2.1. On the topmost level a convoy consisting of a couple of RailCabs is represented by a NMS. Each RailCab itself is described by an AMS and includes several MFMs, e.g. the active suspension system. The information processing units of the structuring elements can be seen as agents. An agent in this context is an information processing unit which is used to fulfill a particular functionality pursuing the corresponding objectives. It analyzes its environment conditions and has the ability to adjust its own behavior autonomously according to the current situation and to the requirements of the remaining agents. In this way hierarchically structured self-optimizing systems can be seen as multi-agent systems.

Information processing of each structuring element can itself be a complex unit consisting of several software components. Hence, it also has to be structured in

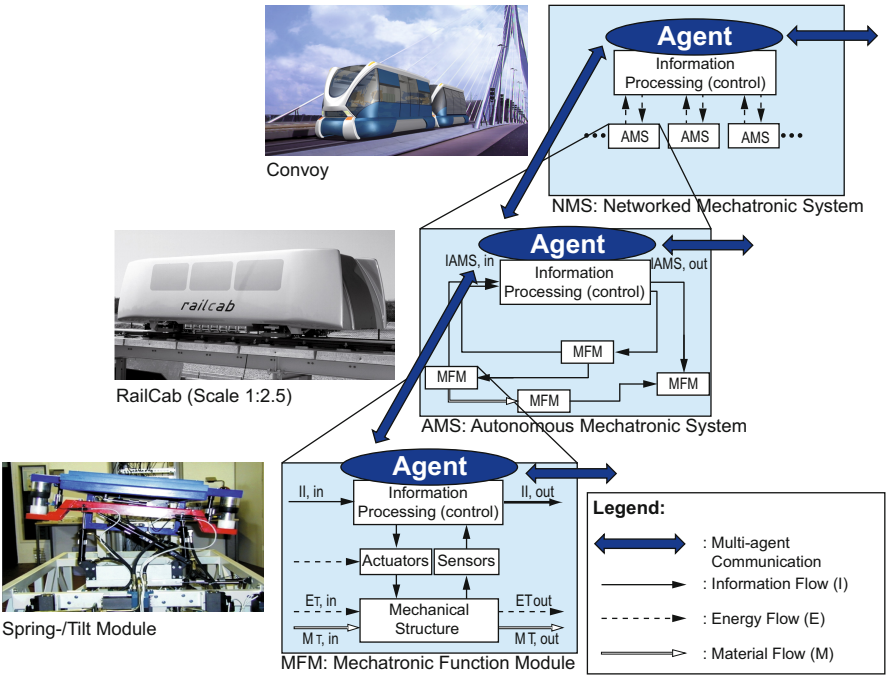

Fig. 1.5 Structure of intelligent mechatronic systems [3]

order to ensure a systematic design and a dependable functionality. The Operator-Controller-Module described in the following section can be used to structure the information processing of self-optimizing systems.

1.3.2 Operator-Controller-Module

The information processing unit of a self-optimizing system has to perform a multitude of functions: quasi-continuous control of the plant motion, monitoring in view of occurring malfunctions, adaptation of the control strategy to react to changing environmental conditions, communication with other systems to name a few of these functions.

In order to ensure a clear and manageable information processing, an architecture is needed which contains all these functions. Additionally, the hierarchical structuring concept described in the last section has to be taken into account. The **Operator-Controller-Module** (OCM) is an architecture with three levels that has been proven to be an advantageous and effective structure for self-optimizing systems (see Fig. 1.6). It is based on results of cognition science, see [48], and was first published in [32]. It is used for the information processing on each level of a complex mechatronic system. The result is a hierarchy of OCMs that is also beneficial for modeling and optimization as described in detail in Sec. 5.3.3. The three different levels of an OCM are geared to the kind of effect on the technical system.

Controller: The controller, which is on the lowest level, realizes the desired dynamical behavior of the plant. It is similar to a classical control loop. Measurements are used to compute control signals which directly affect the plant. Hence, it can be called a "Motor Loop". The controller operates in a quasi-continuous way under hard real-time constraints. Several types of controllers can be implemented at the same time with the possibility to switch between them. Different switching strategies can be used, e.g. a flatness-based approach presented in [42].

Reflective Operator: The reflective operator monitors and regulates the controller. It consists of sequential control, emergency routines as well as adaptation algorithms for the control strategies. The reflective operator does not access the actuators of the system directly, but modifies the controller by initiating changes of controller parameters or switching between different controllers. The reflective operator works mostly in an event-oriented manner. It also has to operate under hard real-time constraints, because it is tightly linked to the controller. However, it is also the connecting element to the cognitive level of the OCM and provides an interface between those elements that are not capable to operate in real-time and the controller. It filters incoming signals and results from the cognitive level and inputs them to the subordinated level.

Cognitive Operator: The topmost level of the OCM is represented by the cognitive operator. On this level the system can gather information on itself and its environment by applying various methods such as learning, use of knowledge-based systems, or model-based optimization. The results can be used to improve the

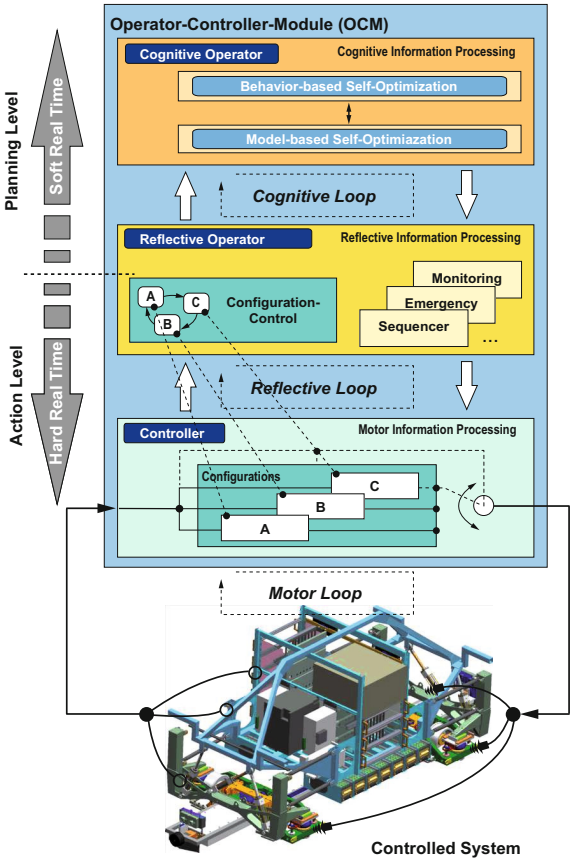

Fig. 1.6 Structure of Operator-Controller-Module [3]

system behavior. This optimizing information processing can roughly be divided into model-based and behavior-oriented optimization, introduced in Sect. 1.4.1 and Sect. 1.4.2, respectively. The former class of optimization techniques is based on a model for the dynamical behavior of technical systems while the latter uses methods from artificial intelligence and soft-computing. While both the controller and the reflective operator are subject to hard real-time constraints, the cognitive operator can also operate asynchronously to real-time. Nevertheless, it has to respond within a certain time limit. Otherwise, self-optimization would not find utilizable results in view of changing environmental conditions. Hence, the cognitive operator is subject to soft real-time. Consequently we are able to integrate cognitive functions into the technical system that previously only biological systems were capable of.

1.4 Self-optimization in Intelligent Technical Systems

Michael Dellnitz, Kathrin Flaßkamp, Philip Hartmann, and Sina Ober-Blöbaum

Self-optimizing systems adapt their behavior according to current situations and objectives. Therefore, appropriate strategies and methods have to be implemented into the Cognitive Operator of the Operator-Controller-Module (cf. Fig. 1.6). As introduced in Sect. 1.2 (cf. in particular Fig. 1.4), the system's adaptation can be realized by parameter adaption and/or by reconfiguration. Finding parameters that optimize a current set of objectives is an optimization or an optimal control problem. Methods to solve these kind of problems typically rely on models of the system's dynamic behavior. In Sect. 1.4.1, an introduction to formal problem statements and solution approaches is given for these *model-based methods for self-optimization*. In case an explicit physical model of the system or process is not available, *behavior-oriented self-optimization* is used. These approaches work on a mapping of input values to output values (cf. Sect. 1.4.2). Strutural reconfiguration affects all levels of a self-optimizing system, from software to hardware. This kind of adaptation is realized by exchanging system parts, e.g. software components or areas of FPGAs (Field Programmable Gate Arrays). We will provide a closer look on reconfiguration in Sect. 1.4.3.

1.4.1 Model-Based Self-optimization

Michael Dellnitz, Kathrin Flaßkamp, and Sina Ober-Blöbaum

The development of self-optimizing mechatronic systems requires the solution of optimization problems from the early design phase to system operation. Model-based design techniques, which are state of the art in particular in control engineering, allow an automatic, model-based computation of solutions that are guaranteed to be optimal for the given problems by numerical optimization methods.

Optimization problems are classified by the type of variables, which can be discrete or continuous. **Discrete optimization problems** typically arise in logistic or planning problems where long term forecasts have to be computed and can be either addressed with discrete model-based or discrete behavior-based methods. The optimization of the design and the dynamical behavior of intelligent mechatronic systems gives rise to various **continuous optimization problems**. If time-dependent steering maneuvers for technical systems or processes have to be optimized, we are faced with **optimal control problems**. In many applications, in particular for self-optimizing systems, there are several objectives which have to be simultaneously optimized leading to **multiobjective optimization**. Regarding the process of self-optimization, it is multiobjective optimization which enables the identification of objectives (step 2 of the self-optimization process, cf. Sect. 1.2) during operation.

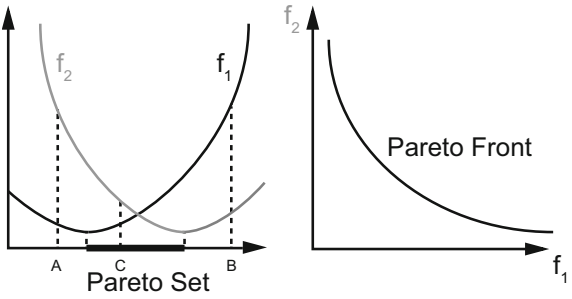

Fig. 1.7 Sketch of an MOP with two objectives f_1 and f_2. While the points (A) and (B) are not optimal, (C) is a point of the Pareto set

1.4.1.1 Multiobjective Optimization

Multiobjective optimization (sometimes also called multicriteria optimization) takes several conflicting objectives into account and searches for optimal compromises, so called **Pareto points**. Simple examples of trade-offs between conflicting objectives are "minimal energy consumption versus minimal time" or "maximal quality, but minimal time and minimal (e.g. production) costs". A number of detailed examples of concurring objectives in technical applications are given in Sect. 2.

While ordinary optimization problems typically have a single global optimum, the solution of multiobjective optimization problems results in an entire set of Pareto points, the **Pareto set**. Formally, the multiobjective optimization problem (MOP) is stated as

$$\min\{\mathbf{F}(\mathbf{p}) : \mathbf{p} \in \mathbb{R}^n\}, \tag{1.1}$$

with $\mathbf{F}$ being a vector of objectives $f_1, \ldots, f_k : \mathbb{R}^n \to \mathbb{R}$, i.e. $\mathbf{F} : \mathbb{R}^n \to \mathbb{R}^k$, $\mathbf{F}(\mathbf{p}) = (f_1(\mathbf{p}), \ldots, f_k(\mathbf{p}))$. Here, $\mathbf{p}$ denotes the optimization parameters. These could be design parameters for the mechanical or electrical subsystem, for instance, or control parameters of the regulators. Minimization is meant with respect to the following partial ordering $\leq_p$ on $\mathbb{R}^n$: given $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$, the vector $\mathbf{u}$ is smaller than the vector $\mathbf{v}$, $\mathbf{u} \leq_p \mathbf{v}$, if $u_i \leq v_i$ for all $i \in \{1, \ldots, k\}$. The solutions of MOP can then be defined as follows: a point $\mathbf{p}^\star \in \mathbb{R}^n$ is called globally Pareto optimal for MOP (or a global Pareto point for MOP)if there does not exist any $\mathbf{p} \in \mathbb{R}^n$ with $\mathbf{F}(\mathbf{p}) \leq_p \mathbf{F}(\mathbf{p}^\star)$ and $f_j(\mathbf{p}) < f_j(\mathbf{p}^\star)$ for at least one $j \in \{1, \ldots, k\}$. That means, no other point $\mathbf{p}$ gives a better or equal (but not entirely identical) value in all objectives. However, there typically exists other Pareto optimal points which are, compared to $\mathbf{p}^\star$, better in one objective but worse in another. Figure 1.7 gives an illustration of an MOP and Pareto optimal points. If the Pareto optimality property only holds for some neighborhood $U(\mathbf{p}^\star) \subset \mathbb{R}^n$, $\mathbf{p}^\star$ is called locally Pareto optimal. The image of the Pareto set, i.e. the corresponding function values, is called the Pareto front (cf. Fig. 1.7).

Necessary optimality conditions for Pareto optimality are given by the Karush-Kuhn-Tucker (KKT) equations, i.e. for an optimal point $\mathbf{x}^\star$, there exist multipliers $\beta_i \in \mathbb{R}$ with $\beta_i \geq 0$ and $\sum_{i=1}^k \beta_i = 1$ such that

$$\sum_{i=1}^{k} \beta_i \nabla f_i(\mathbf{p}^\star) = 0. \tag{1.2}$$

The MOP (cf. Eq. (1.1)) can be extended to include equality or inequality constraints, which have to be considered in the KKT equations involving additional terms with additional multipliers. Iteratively solving Eq. (1.2) to determine Pareto points $\mathbf{p}^\star$ is the basic idea of many multiobjective optimization algorithms.

For the solution of real world multiobjective optimization problems, numerical techniques have to be applied. There exist a number of methods for the computation of single Pareto points, for an overview we refer to [17]. However, for self-optimizing systems, it is important to gather knowledge about the entire Pareto set for later selections of specific design configurations, the **decision making** (cf. Sect. 1.4.1.3) during operation of the system. In the last decades, a number of techniques for the computation of entire Pareto sets have been developed (cf. e.g. [9, 11, 33, 37, 46]). In the course of the research of the CRC 614, set-oriented methods for multiobjective optimization (cf. e.g. [12] for an early reference or [47] for an overview) have been developed. Due to the approximation of the entire Pareto set (or the front, respectively) by box coverings, the methods are outstanding in their robustness and applicability to real world MOP problems, in particular for self-optimizing systems. These techniques are described in detail in Sect. 5.3.1 (cf. also Sect. 5.3.2, Sect. 5.3.4, and Sect. 5.3.5 for extensions to hierarchical and parametric multiobjective optimization problems).

In the following, we give a short introduction to optimal control problems, which often arise in control applications for technical systems. Solution methods for problems with single and multiple objectives are presented in Sect. 5.3.6.

1.4.1.2 Optimal Control

Optimal control problems arise, when the system's dynamical behavior has to be optimized by determining a time-dependent steering maneuver. In other words, such an optimal maneuver has to satisfy certain constraints and has to minimize a given cost functional like the control effort or the maneuver time. Typical examples are the optimal control of open chain industrial robots, the finding of optimal paths for vehicles, or the optimal control of engines. Formally, an optimal control problem (OCP) is defined by a cost functional (1.3a), e.g. the control effort, the time duration, or the deviation to a reference path, that has to be minimized with respect to several constraints:

$$\min_{\mathbf{x}(t),\mathbf{u}(t)} J(\mathbf{x},\mathbf{u}) = \int_0^T C(\mathbf{x}(t),\mathbf{u}(t))\,dt \tag{1.3a}$$

$$\text{with respect to } \dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t),\mathbf{u}(t)) \tag{1.3b}$$

$$\mathbf{r}(\mathbf{x}(0),\mathbf{x}(T)) = 0, \text{ and} \tag{1.3c}$$

$$\mathbf{h}(\mathbf{x}(t),\mathbf{u}(t)) \leq 0. \tag{1.3d}$$

The dynamical system (1.3b) describes the system's equations of motion in its state **x** under the influence of some control **u**. Equations (1.3c) and (1.3d) are called boundary and path constraints, respectively, and take into account technical restrictions on the states or controls.

There exists a number of different approaches for numerically solving single objective OCP, for a good overview we recommend [7] and the references therein. The solution methods can be divided into indirect and direct methods. While indirect methods generate and then solve a boundary value problem according to the necessary optimality conditions of the Pontryagin maximum principle[1], direct methods start with a discretization of the problem (1.3a)-(1.3d). Thus, one obtains a nonlinear optimization problem that can be addressed by appropriate state of the art techniques such as sequential quadratic programming (SQP, cf. e.g. [28]).

In the case of differentially flat systems, the entire dynamics of the technical system can be described via (artificial) outputs and therefore, only the output functions have to be approximated by a finite number of parameters (equally spread nodes or splines as in [26]). Otherwise, the system's continuous states have to be discretized (by a finite number of nodes or even by some appropriately chosen short pieces of trajectories, so called primitives, cf. Sect. 5.3.7). A method that is especially tailored to the optimal control of mechanical systems (*DMOC, Discrete Mechanics and Optimal Control*, cf. [41]) is presented in Sect. 5.3.6.

If several cost functionals of the form (1.3a) have to be optimized simultaneously (e.g. the energetic effort and the duration of a steering maneuver), we are faced with a **multiobjective optimal control problem**. In Sect. 5.3.6, a method is presented that combines a multiobjective optimization algorithm with a direct optimal control technique to address problems of this kind.

1.4.1.3 Decision Making and Self-optimization

The computation of entire Pareto sets of multiobjective optimization or optimal control problems is computationally costly but important for the design of a knowledge base on which the self-optimization during operation time relies. The Pareto optimal alternatives are computed offline in advance and stored in this knowledge base. During operation, one specific optimal configuration of the Pareto set has to be chosen at every time: this process is called decision making.

If only a (small) finite number of Pareto points is stored in the knowledge base, one possibility to implement the decision making process is given by the **hybrid planning** method, cf. Sect. 1.4.2 below and Sect. 5.3.8. Self-optimization based on precomputed knowledge bases of Pareto sets can be also realized by path following methods for parameter-dependent MOPs (cf. Sect. 5.3.4 and [49] for details) or in a model predictive control fashion for scalarized online optimization problems (cf. [26]).

[1] The Pontryagin maximum principle and a discussion of indirect optimal control methods are e.g. given in [7].

1.4.2 Behavior-Oriented Self-optimization

Philip Hartmann

The term behavior-oriented optimization describes methods without an explicit physical model of the system or process. Instead, these approaches work on mapping input values to output values. The actual system and the considered process are observed as a black box and usually a discretization of the processes goes hand in hand. The relationship between system state, behavior, and objectives are provided by the developers of the system or learned by the system by using learning methods and exploration strategies. Because a classification of the environmental conditions and the system behavior are assumed when using defaults defined by experts or learning methods, the model of the behavior-oriented self-optimization is in general coarser than the model of the model-based self-optimization (cf. Sect. 1.4.1). Thus it is possible to plan the system's behavior for longer planning horizons.

Planning refers to a process that determines and examines the future behavior of the system instead of considering only the current situation. To analyze alternative options for execution, planning methods work on simplified models of the system behavior. An integral part of the simplification is the mapping to a discrete state space, so that only a snapshot of the system state at defined points in time will be considered. In general this approach corresponds to the definition of the planning problem in artificial intelligence, which can be used with a variety of methods. Due to the exploration of the state space for future situations a proactive reaction to future influences or avoidance of undesirable situations is made possible by the planning [3].

Any task of a mechatronic system (e.g. the transportation of persons or goods between two locations) can be expressed by a function [43]. This function describes the relationship between input and output variables [1] of the system by converting incoming energy, material and information flows into outgoing flows of the same types. Subtasks (such as driving with an active suspension system) are represented by analogous partial functions which are logically connected and make up the hierarchy of the overall function. The effect of a partial function also depends on the physical effect leading to a particular partial function solution [43]. Thus, partial functions can be implemented using various solutions (e. g. high or low compensation of disturbances). The choice of solutions to these partial functions determines the solution to the overall function and its effect on the mechatronic system.

Let PF_{leaf} be the set of all partial functions at the lowest level in the overall function hierarchy. Then, the selected solutions in the overall function at time t_j (cf. Fig. 1.8, black circles) can be listed as a sequence of solutions to partial functions[35]:

$$s_{of}(t) = (s_{pf_1}, ..., s_{pf_k}), \tag{1.4}$$

with $s_{pf_i} \in S_{pf_i}$ and $1 \leq i \leq k = |PF_{leaf}|$.

The behavior of the overall system at time t can be described by $V(t) = (\mathbf{x}_t, s_{of}(t), \mathbf{y}_t)$ with $\mathbf{x}_t$ as an input vector and $\mathbf{y}_t$ as an output vector. Consequently,

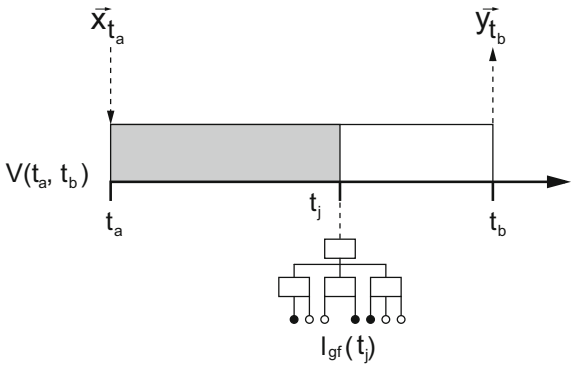

Fig. 1.8 Planning sequence for the behavior of a self-optimizing system

the behavior of one time period is: $V(t_a,t_b) = (V(t_a),...,V(t_b))$ with $t_a < t_b$ (cf. Fig. 1.8).

The mechatronic system reacts to each input variable by converting it to the output variables; it does so by executing the currently implemented overall function (linked partial functions of the functional hierarchy, reactive behavior). The information processing takes time and there is a latency involved according to the response to input variables. Because of that, the effect of the overall function on the output variables is delayed. Since the system execution takes place under real-time conditions, this has to be taken into account during the development of self-optimizing systems.

For **autonomous control** [1] of the behavior, the system has to independently select solutions to the partial functions at certain times in order to achieve the desired effect (active behavior). The selection is made by considering specific objectives. This is the decision problem of a self-optimizing system: selecting solutions to partial functions which achieve these objectives with a high reliability result (goal-directed behavior).

1.4.2.1 Deterministic Planning

Mechatronic systems are able to execute a function in different ways. From the planning perspective, these different ways are distinguished from each other by how well they achieve the possible objectives and how they change the system state. We refer to these different implementations of functions as operation modes. In a mechatronic planning domain most relevant variables are numerical, hence we restrict the state vector to real valued numerical variables. A deterministic planning model for behavior-oriented self-optimization can be formulated as follows [36]:

- OM a finite set of available operation modes
- S a finite set of possible system states
- $\mathbf{s} \in S$ a state vector with $s(i) \in \mathbb{R}$ for the i-th component

Furthermore for each operation mode $om \in OM$ exists:

- $prec^{om} := \{(x_{lower} < s(i) < x_{upper}) | x_{lower}, x_{upper} \in \mathbb{R}\}$ a set of preconditions which have to be true for executing *om*
- $post^{om}$ a set of conditional numeric functions to define the effects on the state variables of the subsequent state s'

A specific planning problem is finding a sequence of operation modes which describes a transition from an initial system state $\mathbf{s}_i \in S$ to a predetermined goal state $\mathbf{s}_g \in S$. So a single task of a mechatronic system is given as a 2-tupel $O = (\mathbf{s}_i, \mathbf{s}_g)$. A solution of the planning problem can be determined by applying a state space search algorithm (cf. [27]), for example [36].

1.4.2.2 Probabilistic Planning

Because of the uncertainty of environmental influences, probabilistic planning models are formulated based on the deterministic planning models. A probabilistic model for behavior planing for self-optimzing mechatronic system consists of probabilistic states s^p with

$$range(s^p(i)) \rightarrow W(\mathbb{R})$$

for the value range (e.g. $0 \leq SOC_k \leq 100$ with SOC_k for the state of charge of the mechatronic system in state k) and

$$distribution(s^p(i))$$

for the probabilistic distribution for state variable $s^p(i)$ (e. g. $\mathbb{P}(SOC_k \leq 50) = 0.25 \wedge \mathbb{P}(SOC_k > 50) = 0.75$). Furthermore, there are probabilistic variants of the operation modes *om* of the mechatronischen System with

- $in_s^{om} \subseteq pre^{om}$ for a subset of input variables and
- $out_s^{om} \subseteq post^{om}$ for a subset of output variables

Then for each output variable $o \in out_s^{om}$ a bayesian network (cf. [6]) bn_o^{om} is created to formulate the probabilistic effect Δ on the state variable of the subsequent state $k+1$ by conditional probabilities (e. g. $\mathbb{P}(SOC_{k+1}^{\Delta} = +10 | windKmh \leq 20 \wedge SOC_k > 50) = 0.015$). With the definition of a lower respectively upper bound for critical state values (e. g. $SOC_{k+1} \leq 10$) branching points with likely violation of these bounds can be found and alternative plans can be genrated by using just-in-case-planning [35, 36, 38].

1.4.2.3 Hybrid Planning

The planning methods described above consider discrete system states and transitions relying, for instance, on average values or approximations. However, the continuous behavior can not be neglected for mechatronic systems. Therefore the necessity arises to integrate the continuous domain also in the planning process. Furthermore, planning for mechatronic systems has to cope with changing environmental conditions and imprecisions of a priori defined models during system operation which grow further with a widening planning horizon. For these reason

continuous planning was combined with the discrete planning techniques presented above; the so called hybrid planning.

The hybrid planner uses the **discrete planning** techniques to generate an offline plan before the system starts its operation. For the RailCab this plan would, for instance, determine the course (sequence of track sections) to reach its destination and the parameter settings, i.e. the selected Pareto points based on the results of the multiobjective optimization. During system operation while executing the plan, however, deviations between the actual system state and those assumed by the offline plan can not necessarily be avoided due to the time that elapsed before the system reaches a certain state of the plan during execution. Also unforeseen conditions or changes of the environment may cause such deviations.

To counteract these shortcomings the hybrid planner simulates the system's behavior including the **continuous aspects** in an online manner. This simulation anticipates the future behavior of the system for a restricted time and allows to directly adapt the current action according to the simulation results. When the remaining part of the (discrete) offline plan is affected, the results of the just-in-case-planning may be used. If this is not possible an online replanning is initiated [4, 5, 18].

1.4.3 Self-optimization by Reconfiguration

Stefan Groesbrink, Sebastian Korf, Mario Porrmann, Claudia Priesterjahn, and Katharina Stahl

A self-optimizing system applies reconfiguration methods to adjust to changing requirements. In contrast to simple parameter changes, reconfiguration modifies the internal structure of a hardware or software system. When principles of self-optimization refer to the topology and structure of mechatronic systems, a reconfigurability of the system architecture or of dedicated system components is required. Reconfiguration decisions must be made autonomously. Therefore, parts of the classical design process have to be performed by the system at runtime: various implementation alternatives are available, from which the system selects the most appropriate realization (hardware or software) and the corresponding parameters.

Reconfiguration is executed on every system level: Self-optimizing Application, System Software, and Hardware. The levels of a self-optimizing mechatronic system are shown in Figure 1.9 and described in the following.

The **self-optimizing application** may be the execution of self-optimizing algorithms, e.g. finding optimal strategies to perform a task, or the communication between system parts. On the application level, reconfiguration means the exchange of software parts, e.g., switching between different software implementations to change the system behavior. Thereby, the self-optimizing application may change the requirements on the system software and its services. The **system software** interconnects the self-optimizing application and hardware and is composed of a virtualization layer and operating system. The virtualization layer is optional and enables the hosting of multiple operating systems on a single hardware platform. The system software supports the applications by reacting in a self-optimizing manner to

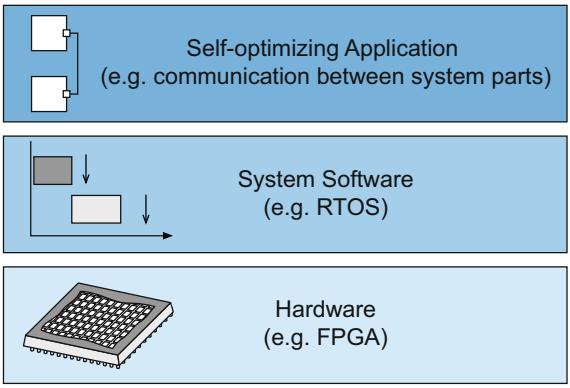

Fig. 1.9 Levels of a self-optimizing system

the changing operating conditions of both the applications and the self-optimizing hardware. Self-optimization is introduced on the **hardware** level by means of dynamically reconfigurable hardware. Here, hardware reconfiguration means changing the functionality or the interconnect of hardware modules in microelectronic systems before and even during operation. Self-optimization in hardware must be encapsulated by the system software so that applications will execute without any perceivable interference. The system software must guarantee service supply in accordance to the given real-time constraints to enable hardware reconfiguration at runtime.

Self-optimizing Application

The self-optimizing application is implemented in the Cognitive and Reflective Operator of the OCM (cf. Sect. 1.3.2). The Cognitive Operator gathers information about the system and the environment. It applies methods like learning and model-based self-optimization to optimize the system behavior. The Reflective Operator is the interface between the Cognitive Operator and the Controller. The interaction with the Controller requires operation in hard-real time and includes safety-critical tasks. This, in turn, demands a **safe software** that is free from design faults.

We therefore apply model-based software development to guarantee that the software satisfies all safety and real-time requirements. This means, the software is designed using models, the models are verified, and program code is generated, which preserves the verified properties.

On the level of the self-optimizing application, reconfiguration means the creation or removal of software components at runtime. This reconfiguration is specified by graph transformation rules (cf. Sect. 5.2.3.1) at design time. This allows to prove that no unplanned, e.g. unsafe, configurations are created at runtime.

The behavior of the components which execute the reconfiguration rules is modeled using state-based real-time behavior models. The reconfiguration rules are executed as side effects of this behavior. Therefore, the reconfiguration rules must not only guarantee to create no unplanned configurations but they must also satisfy the

time constraints of the real-time behavior. To ensure this, we extended graph transformation rules by timing information and developed a verification approach (cf. Sect. 5.2.3.2) that takes into account the state-based real-time behavior, the reconfigurations, and the execution times of the reconfigurations.

System Software

The software models also allow computing application parameters such as the worst-case execution times [30]. In the context of mechatronic systems, the operating system has to manage the execution of the applications considering timeliness and predictability of the system behavior. It needs this parameter for the required real-time scheduling. Since efficiency is an important factor for operating systems for restricted environments such as mechatronic systems, operating system designers aim to deliver an application- or a domain-specific operating system with the objective to integrate required functionality only. ORCOS (**Organic Reconfigurable Operating System**) [19] is an example for an fully customizable real-time operating system at design time. Beyond design time configurability, the entire information processing process of monitoring, analyzing and reacting must be integrated into the self-optimizing operating system. In Sect. 5.5, we will present an extension of the ORCOS architecture that builds up the basis for online reconfiguration. However, self-optimization in the operating system is not restricted to react on requirement changes only. The operating system may also implement methods that can be applied to self-optimize the performance of the operating system (e.g. resource allocation strategy) as well as the overall system performance (e.g. resource utilization).

The operating system manages the use of hardware resources. This includes the abstraction of the underlying hardware which is usually done by implementing drivers. Dedicated interfaces specify the access to the hardware. In our approach, dynamic reconfiguration is provided by a combination of dynamically reconfigurable hardware and a reconfigurable real-time operating system (RTOS). The proposed hardware platform offers the fundamental mechanisms that are required to execute arbitrary software and to adapt the system to new requirements (e.g. by dynamic reconfiguration). The operating system triggers hardware reconfiguration as a reaction to varying requirements and decides whether a task is executed in software, in hardware, or in a combination of both.

Hardware

To adapt to changing environments, dynamically reconfigurable hardware is a key technology. Dynamically reconfigurable hardware can be classified as fine-grained or coarse-grained. Fine-grained reconfigurable architectures are typically based on **Field Programmable Gate Arrays** (FPGAs), which facilitates the System on Programmable Chip (SoPC) designs with a complexity of several million logic gates, several hundred kBytes of internal SRAM memory, and embedded processor cores. For the group of coarse-grained architectures we introduce reconfigurable

embedded processors, which can change their internal structure to adapt to the currently needed environment.

The idea of dynamic and partial hardware reconfiguration is to reconfigure the hardware in a way that it maximizes the use of all available resources for the quired controller implementation. The information processing system is shared among all tasks, and offers limited resources with respect to memory, computational power, and energy. Any task may be composed of various sub-tasks and different realizations of these sub-tasks (e.g. different software implementations running on an embedded CPU and various hardware implementations for an integrated FPGA). Since all of these realizations have different computational requirements and different application characteristics, a control algorithm in a self-optimizing system can be understood as an optimal solution for the current internal and external objectives of the system. Therefore, in each new environmental condition of the mechatronic system, there is a controller architecture and a corresponding implementation variant that represent an optimal solution in this situation [45].

1.5 Structure of This Book

Mareen Vaßholz

In this Chapter self-optimizing systems were described briefly. It serves to show the potential of self-optimization for technical systems. In Chap. 2 examples of self-optimizing systems are presented that were developed in the Collaborative Research Center 614. They show the benefits that are provided by using self-optimization, but demonstrate its complexity as well. The resulting challenges for the development of these systems show the need for a design methodology presented in the following chapters. The different development tasks that have to be performed, are presented as a reference process for the development of self-optimizing systems in Chap. 3. This reference process is divided into the domain-spanning conceptual design and the domain-specific design and development phase. Chapter 4 depicts the domain-spanning development methods and tools. The ones relevant to the domain-specific design and development are presented in Chap. 5. The applications serve as examples for the description of the methods and tools for the development of self-optimizing systems. Chapter 6 gives a summary and an outlook over future work in the field of self-optimization and intelligent technical systems.

This book is one result of the research of the Collaborative Research Center 614 "Self-Optimizing Concepts and Structures in Mechanical Engineering" and is complemented by the book "Dependability of Self-Optimizing Mechatronic Systems". It focuses on tools and methods to ensure the dependability of self-optimizing systems during development and run-time. Throughout this book you will find cross-references, like [24, D.o.S.O.M.S. Chap. 2] , for detailed information on dependability specific methods and tools.

References

1. DIN 19 226 Teil 1: Leittechnik - Regelungstechnik und Steuerungstechnik - Allgemeine Grundbegriffe. Deutsche Norm (1994)
2. VDI 2206 - Entwicklungsmethodik für mechatronische Systeme. Beuth Verlag, Berlin (2004)
3. Adelt, P., Donoth, J., Gausemeier, J., Geisler, J., Henkler, S., Kahl, S., Klöpper, B., Krupp, A., Münch, E., Oberthür, S., Paiz, C., Porrmann, M., Radkowski, R., Romaus, C., Schmidt, A., Schulz, B., Vöcking, H., Witkowski, U., Witting, K., Znamenshchykov, O.: Selbstoptimierende Systeme des Maschinenbaus. In: Heinz Nixdorf Institut, Universität Paderborn, vol. 234. HNI-Verlagsschriftenreihe, Paderborn (2009)
4. Adelt, P., Esau, N., Hölscher, C., Kleinjohann, B., Kleinjohann, L., Krüger, M., Zimmer, D.: Hybrid Planning for Self-Optimization in Railbound Mechatronic Systems. In: Naik, G. (ed.) Intelligent Mechatronics, pp. 169–194. InTech Open Access Publisher, New York (2011)
5. Adelt, P., Esau, N., Schmidt, A.: Hybrid Planning for an Air Gap Adjustment System Using Fuzzy Models. Journal of Robotics and Mechatronics 21(5), 647–655 (2009)
6. Ben-Gal, I.: Bayesian Networks. In: Encyclopedia of Statistics in Quality and Reliability (2007)
7. Binder, T., Blank, L., Bock, H., Bulirsch, R., Dahmen, W., Diehl, M., Kronseder, T., Marquardt, W., Schlöder, J., von Stryk, O.: Introduction to Model-based Optimization of Chemical Processes on Moving Horizons. In: Grötschel, M., Krumke, S., Rambau, J. (eds.) Online Optimization of Large Scale Systems - State of the Art, pp. 295–340. Springer, Heidelberg (2001)
8. Böcker, J., Schulz, B., Knoke, T., Fröhleke, N.: Self-Optimization as a Framework for Advanced Control Systems. In: Proceedings of the 32nd Annual Conference on IEEE Industrial Electronics, Paris (2006)
9. Coello Coello, C.A., Lamont, G., Veldhuizen, D.V.: Evolutionary Algorithms for Solving Multi-Objective Optimization Problems, 2nd edn. Springer, Heidelberg (2007)
10. Comford, R.: Mecha.. what? IEEE Spectrum 31(8), 46–49 (1994)
11. Das, I., Dennis, J.: A Closer Look at Drawbacks of Minimizing Weighted Sums of Objectives for Pareto Set Generation in Multicriteria Optimization Problems. Structural Optimization 14(1), 63–69 (1997)
12. Dellnitz, M., Schütze, O., Hestermeyer, T.: Covering Pareto Sets by Multilevel Subdivision Techniques. Journal of Optimization Theory and Application 124(1), 113–136 (2005)
13. 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 Developement 7(1), 53–67 (2013)
14. 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)
15. Dumitrescu, R., Anacker, H., Gausemeier, J.: Design Framework for the Integration of Cognitive Functions into Intelligent Technical Systems. Production Engineering - Research and Developement 7(1), 111–121 (2013)
16. Dumitrescu, R., Gausemeier, J., Romaus, C.: Towards the Design of Cognitive Functions in Self-Optimizing Systems Exemplified by a Hybrid Energy Storage System. In: Proceedings of the 10th International Workshop on Research and Education in Mechatronics, Ostrava (2010)

17. Ehrgott, M.: Multicriteria Optimization, 2nd edn. Springer, Heidelberg (2005)
18. Esau, N., Krüger, M., Rasche, C., Beringer, S., Kleinjohann, L., Kleinjohann, B.: Hierarchical Hybrid Planning for a Self-Optimizing Active Suspension System. In: Proceedings of the 7th IEEE Conference in Industrial Electronics and Applications, Singapore (2012)
19. FG Rammig, University of Paderborn: ORCOS - Organic Reconfigurable Operating System, `https://orcos.cs.uni-paderborn.de/doxygen/html` (accessed August 12, 2013)
20. Frank, U., Gausemeier, J.: Self-Optimizing Concepts and Structures in Mechanical Engineering - Specifying the Principle-Solution (2005)
21. Frank, U., Giese, H., Klein, F., Oberschelp, O., Schmidt, A., Schulz, B.H.V., Witting, K.: Selbstoptimierende Systeme des Maschinenbaus. In: Heinz Nixdorf Institut, Universität Paderborn, vol. 155. HNI-Verlagschriftenreihe, Paderborn (2004)
22. Gausemeier, J.: From Mechatronics to Self-optimizing Concepts and Structures in Mechanical Engineering: New Approaches to Design Methodology. International Journal of Computer Integrated Manufacturing 18(7), 550–560 (2005)
23. 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)
24. Gausemeier, J., Rammig, F.J., Schäfer, W., Sextro, W. (eds.): Dependability of Self-optimizing Mechatronic Systems. Springer, Heidelberg (2014)
25. 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)
26. Geisler, J., Witting, K., Trächtler, A., Dellnitz, M.: Multiobjective Optimization of Control Trajectories for the Guidance of a Rail-bound Vehicle. In: Proceedings of the 17th IFAC World Congress, Seoul (2008)
27. Ghallab, M., Nau, D., Traverso, P.: Automated Planning - Theory and Practice. Elsevier, Amsterdam (2004)
28. Gill, P.E., Jay, L.O., Leonard, M.W., Petzold, L.R., Sharma, V.: An SQP Method for the Optimal Control of Large-scale Dynamical Systems. Journal of Computational and Applied Mathematics 120, 197–213 (2000)
29. Harashima, F., Tomizuka, M., Fukuda, T.: Mechatronics - What is it?, Why and how? An Editorial. IEEE/ASME Transactions on Mechatronics 1(1) (1996)
30. Henkler, S., Oberthür, S., Giese, H., Seibel, A.: Model-driven Runtime Resource Predictions for Advanced Mechatronic Systems with Dynamic Data Structures. Computer Systems Science & Engineering 26(6) (2011)
31. Hestermeyer, T.: Strukturierte Entwicklung der Informationsverarbeitung für die aktive Federung eines Schienenfahrzeugs. Ph.D. thesis, Fakultät für Maschinenbau, Universität Paderborn, Verlag Dr. Hut, München (2006)
32. Hestermeyer, T., Oberschelp, O., Giese, H.: Structured Information Processing for Self-Optimizing Mechatronic Systems. In: Proceedings of the 1st International Conference on Informatics in Control, Automation and Robotics, Setubal (2004)
33. Hillermeier, C.: Nonlinear Multiobjective Optimization - A Generalized Homotopy Approach. Birkhäuser (2001)
34. Isermann, R.: Mechatronische Systeme - Grundlagen. Springer, Heidelberg (2008)
35. Klöpper, B.: Ein Beitrag zur Verhaltensplanung für interagierende intelligente mechatronische Systeme in nicht-deterministischen Umgebungen. Ph.D. thesis, Fakultät für Wirtschaftswissenschaften, Universität Paderborn, HNI-Verlagsschriftenreihe, Band 253, Paderborn (2009)

36. Klöpper, B., Aufenanger, M., Adelt, P.: Planning for Mechatronics Systems - Architechture, Methods and Case Study. Engineering Applications of Artificial Intelligence 25(1), 174–188 (2012)
37. Knowles, J., Corne, D., Deb, K.: Multiobjective Problem Solving from Nature: From Concepts to Applications. Springer, Heidelberg
38. Köpper, B., Sondermann-Wölke, C., Romaus, C.: Probabilistic Planning for Predictive Condition Monitoring and Adaptation within the Self-Optimizing Energy Management of an Autonomous Railway Vehicle. Journal for Robotics and Mechatronics 24, 5–15 (2012)
39. Landau, I., Lozano, R., M'Saad, M., Karimi, A.: Adaptive Control - Algorithms, Analysis and Applications. Springer, Heidelberg (2011)
40. Lückel, J., Hestermeyer, T., Liu-Henke, X.: Generalization of the Cascade Principle in View of a Structured Form of Mechatronic Systems. In: Proceedings of the IEEE/ASME International Conference on Advanced Intelligent Mechatronics, Como (2001)
41. Ober-Blöbaum, S., Junge, O., Marsden, J.E.: Discrete Mechanics and Optimal Control: An Analysis. Control, Optimisation and Calculus of Variations 17(2), 322–352 (2011)
42. Osmic, S., Trächtler, A.: Flatness-based Online Controller Reconfiguration. In: Proceedings of the 34nd Annual Conference of the IEEE Industrial Electronics Society, Orlando (2008)
43. Pahl, G., Beitz, W., Feldhusen, J., Grote, K.H.: Kontruktionslehre. Grundlagen erfolgreicher Produktentwicklung - Methoden und Anwendung, 6th edn. Springer, Heidelberg (2005)
44. Pahl, G., Beitz, W., Feldhusen, J., Grote, K.H.: Engineering Design - A Systematic Approach, 3rd edn. Springer, Heidelberg (2007)
45. Paiz, C., Hagemeyer, J., Pohl, C., Porrmann, M., Rückert, U., Schulz, B., Peters, W., Böcker, J.: FPGA-Based Realization of Self-Optimizing Drive-Controllers. In: Proceedings of the 35th Annual Conference of the IEEE Industrial Electronics Society, Porto (2009)
46. Schäffler, S., Schultz, R., Weinzierl, K.: A Stochastic Method for the Solution of Unconstrained Vector Optimization Problems. Journal of Optimization Theory and Applications 114(1), 209–222 (2002)
47. Schütze, O., Witting, K., Ober-Blöbaum, S., Dellnitz, M.: Set Oriented Methods for the Numerical Treatment of Multi-objective Optimization Problems. In: Tantar, E., Tantar, A.-A., Bouvry, P., Del Moral, P., Legrand, P., Coello Coello, C.A., Schütze, O. (eds.) EVOLVE- A bridge between Probability. SCI, vol. 447, pp. 185–218. Springer, Heidelberg (2013)
48. Strube, G.: Modelling Motivation and Action Control in Cognitive Systems. In: Mind Modelling, pp. 89–108. Pabst, Berlin (1998)
49. Witting, K., Schulz, B., Dellnitz, M., Böcker, J., Fröhleke, N.: A new Approach for Online Multiobjective Optimization of Mechatronic Systems. International Journal on Software Tools for Technology Transfer STTT 10(3), 223–231 (2008)