Industry

Matthias Bitzer*, Martin Herrmann and Eckart Mayer-John

System Co-Design (SCODE): methodology for the analysis of hybrid systems

System Co-Design (SCODE): Methodik zur Analyse hybrider Systeme

A systematics for complexity reduction of control software in embedded systems
Eine Systematik zur Komplexitätsreduktion von regelungstechnischer Software in eingebetteten Systemen

https://doi.org/10.1515/auto-2020-0037
Received March 24, 2020; accepted March 25, 2020

Abstract: The increasing complexity of control engineering functionality and its implementation in software requires a methodology for the systematic integration of control theoretical as well as software engineering approaches. The presented SCODE methodology aims towards a systematic derivation of operational modes and their verification considering all relevant logical relations for the respective functional design. This is done in the sense of an *Essential Analysis* for mechatronic systems. Apart from guaranteed (logical) completeness and consistency, the benefit of systematic modular structuring results in reduction of complexity as well as improved maintainability and testability. The article discusses the method on the basis of a mechatronic example, demonstrates its advantages, and describes available tool support.

Keywords: System Co-Design, Essential Analysis, Morphological Box, Operational Mode, Separation of Control and Data Flow

Article note: This article has previously been published in German in at-Automatisierungstechnik [5]. Four references in German could be replaced [11, 13, 31, 37] equivalently. The following supplementary references [1, 6, 10, 17, 18, 29, 33, 34, 38] are additionally provided in the English version.

***Corresponding author: Matthias Bitzer,** Corporate Sector Research and Advance Engineering, Model-Based Systems Engineering (CR/AEE), Robert Bosch GmbH, Robert-Bosch-Campus 1, 71272 Renningen, Germany, bosch.com/research, e-mail: matthias.bitzer2@de.bosch.com
Martin Herrmann, Eckart Mayer-John, Corporate Sector Research and Advance Engineering, Model-Based Systems Engineering (CR/AEE), Robert Bosch GmbH, Robert-Bosch-Campus 1, 71272 Renningen, Germany, bosch.com/research, e-mails: martin.herrmann@de.bosch.com, eckart.mayer@de.bosch.com

Zusammenfassung: Die zunehmende Komplexität von regelungstechnischer Funktionalität und deren Umsetzung in Software erfordert eine Systematik zur Integration von systemtheoretischen als auch softwaretechnischen Vorgehensweisen. Die vorgestellte SCODE-Methode zielt auf eine systematische Herleitung von Betriebsmodi und Verifikation aller für den Funktionsentwurf relevanten logischen Zusammenhänge im Sinne einer *Essentiellen Analyse* für mechatronische Systeme. Der Nutzen einer systematischen modularen Strukturierung ist neben garantierter (logischer) Vollständigkeit und Konsistenz durch eine damit einhergehende Komplexitätsreduktion, sowie bessere Wartbarkeit und Testbarkeit gegeben. Der Artikel beschreibt die Methode anhand eines mechatronischen Beispiels, zeigt ihren Nutzen auf und stellt die vorhandene Toolunterstützung vor.

Schlagwörter: System Co-Design, Essentielle Analyse, Morphologischer Kasten, Betriebsmode, Trennung von Kontroll- und Datenfluss

1 Introduction

Modern mechatronic systems are characterized by a steady increase in control and automation functionality. The implementation of these functions in software typically leads to a corresponding increase in the complexity of the program code of embedded systems. From a mathematical point of view, a substantial factor for the complexity of control software lies in the need to integrate continuous signals (data flow) and event-oriented control flow such as activation and deactivation commands. It has to be considered that the classical ways of thinking in control engineering on the one hand and information technology on the other hand are paradigmatically opposed. While the basic understanding of control engineering relies on continuous, parallel signal models [18, 24, 10, 41], classical information technology is characterized by event-driven, sequential calculation models [15, 25].

Corresponding mixed continuous/discrete-event phenomena are generally represented formally and mathematically by hybrid automata [14, 36] or switched model structures [4, 16]. The logical part of the hybrid automata characterizes operating modes according to the current operating situations, whereas the continuous model part describes the continuous physical process of the controlled plant which is valid during these operating situations. Holistic approaches of hybrid systems analysis, which are integrating both discrete events and continuous components, have been further developed in the last two decades. Unfortunately they do not scale for realistic large complex systems without prior abstraction [19].

In order to achieve the desired complexity reduction as well as a systematic modularization of control functions, the SCODE method[1] (System CO-DEsign, short SCODE) [42, 35, 27] has been developed by Bosch. The presented method combines the two above mentioned opposite views within an integrated systematic approach. A co-design of data and control flow is performed, in which elements of hybrid systems theory are used as description tools. The approach is based on the idea of an Essential Analysis for mechatronic systems[2] and aims at the systematic identification of operating situations as well as the stringent derivation of corresponding operating modes. The original concept of an Essential Analysis goes back to [30] and was proposed in the context of classical IT problems. The core of the Essential Analysis for mechatronic systems is a morphological analysis of the combinatorial logic of the respective application in a problem-immanent way. In this respect, the SCODE method represents an extension of the purely formal mathematical approach of the above-mentioned hybrid automata [14, 36] as well as switched model structures [16], since the derivation of the logical model part is performed stringently by means of a context analysis of the considered application.

1 Within Bosch, the method was successfully applied to numerous ECU functions, such as the diesel air system [42] and flex-fuel systems [35]. Outside Bosch, the SCODE method and the associated tools are offered by ETAS [27, 2, 3, 12] and are used in the automotive industry.

2 While in control engineering *system* is understood as a delimited section of the world whose dynamics and input/output behaviour are to be considered (often the plant to be controlled), in computer science *system* is understood as the system to be built, i. e., the control algorithms and software, possibly including hardware.

Both the mechatronic system approach (controlled plant), its environmental conditions, the specified user requirements, and the control function design are included. From a modeling point of view, the Essential Analysis thus developed for mechatronic systems can be understood as a logical metamodeling to derive the discrete event part of hybrid or switched models and respective software structures.

Another major benefit of the SCODE approach is that all steps of the method can be carried out with the aid of tools, such that all complex logical correlations for each development step are immediately displayed, documented in a transparent manner, and are furthermore automatically checked for completeness and consistency.

2 Essential analysis of mechatronic systems

The core of the SCODE method is a logical analysis of the discrete event components of hybrid and switched systems as explained in the following. The principle of separation of logical control and continuous data flow represents an important basic idea which is illustrated in Section 2.1. Subsequently, Section 2.2 presents the formal procedure of the Essential Analysis in order to deal with combinatorial complexity and the structuring of discrete problem spaces. Finally, Section 2.3 discusses morphological aspects of control engineering applications with regard to the choice of a problem-specific semantics.

2.1 Separation of control and data flow

The goal of the separation of control and data flow is a systematic modularization of functional structures and a description of the continuous functional components through exclusively valid modes.

2.1.1 Motivation and problem definition

The engineering design of a control system is often addressed starting from the basic functionality in normal operation and designed as a continuous parallel data flow. This data flow is usually structured in signal-oriented block diagrams corresponding to an analog computing circuit. Once the function has been designed for normal operation, *special cases* are added (initialization and start-up, control variable limitations, deviating operating modes, shutdown, etc.). Their situation-dependent activation by means of corresponding switching events forms the control flow of the system (switching, activation and deactivation commands, IF statements, piecewise defined mathematical functions, concurrent state machines). The entire control functionality is (almost) always of hybrid nature, i. e., it comprises continuous data flows and discrete-event components in the form of control flows. Due to this incremental approach, the control flow elements are typically *buried* deep in the code or, in the case of graphical programming approaches, distributed over the calculation plan. Figure 1 shows a switch (IF statement) as an example of a *buried* control flow Σ_{CF}.

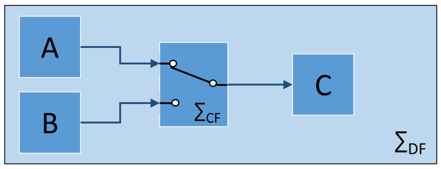


Figure 1: Hybrid system with *buried* control flow Σ_{CF} (switch or IF-Statement).

The motivation for the development of the SCODE method was the existence of evolutionarily developed *monolithic* functional structures in ECU software, in which up to 30 such switching elements occurred and were typically distributed over the calculation plan, cf. [35, 42]. Because of this distribution and the related combinatorial explosion ($2^{30} \approx 10^9$ theoretically possible switching positions), such structures elude the systematic analysis and test methodology which is increasingly required for software quality and safety (e. g., requirements according to ISO 26262). Systems developed in this way and repeatedly extended over their life-time ultimately exhibit significant deficiencies in comprehensibility, expandability and variant stability and therefore generate high maintenance costs.

By applying the method presented in this article, the above mentioned monolithic implementations could be restructured into modularly structured functions with 6 [35] or 10 [42] exclusively valid operational modes. The basic idea of the separation of control and data flow is illustrated in simplified form in Figures 1 and 2: The control flow Σ_{CF} *buried* in the data flow Σ_{DF} in Figure 1 is shifted from the inside to the outside. This leads to two modes **M1** and **M2**, of which exactly one is valid at any time and which contains only the relevant data flow Σ^1_{DF} or Σ^2_{DF}, respectively.

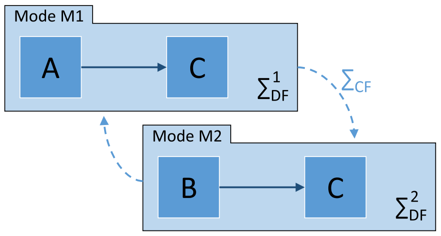


Figure 2: Hybrid mode structured system (For the implementation, block C is instantiated in both modes, i. e., no code duplication).

2.1.2 System theoretical interpretation

From the point of view of hybrid systems theory, the separation of data and control flow represents the splitting of the system behavior into mode-dependent continuous dynamic systems Σ^i_{DF}, $i = 1, \ldots, N_{\mathrm{M}}$ according to

$$\Sigma^{\mathrm{i}}_{\mathrm{DF}}: \begin{cases} \dot{\boldsymbol{x}}_i = \boldsymbol{f}_i(\boldsymbol{x}_i, \boldsymbol{u}), & t \in (t_{\mathrm{Start}}, t_{\mathrm{End}}) \\ \boldsymbol{y} = \boldsymbol{h}_i(\boldsymbol{x}_i, \boldsymbol{u}), & t \in [t_{\mathrm{Start}}, t_{\mathrm{End}}) \\ \boldsymbol{x}_i(t_{\mathrm{Start}}) = \boldsymbol{g}_i(\boldsymbol{x}_{i^-}(t_{\mathrm{Start}}), i^-), \end{cases} \tag{1}$$

and a discrete-event control flow component

$$\Sigma_{\mathrm{CF}}: \begin{cases} \text{Successor:} \quad i^+ = \mu(i, e_1, \ldots, e_{N_{\mathrm{E}}}), \\ \text{Event:} \quad e_j = e_j(\mathfrak{i}_{\mathrm{d}}, \mathfrak{z}(\boldsymbol{u}, \boldsymbol{y}, \tilde{\boldsymbol{y}}, t)) \end{cases} \tag{2}$$

for the realization of the switching conditions, see Figure 3.

The notation in (1) corresponds to a nonlinear state space representation with the input $\boldsymbol{u}$, the output $\boldsymbol{y}$ and the (possibly mode-dependent) state vector $\boldsymbol{x}_i$. Here, t_{Start} and t_{End} denote the point of changeover triggered by an event e_j, $j = 1 \ldots N_{\mathrm{E}}$, cf. (2), characterizing the start and end time

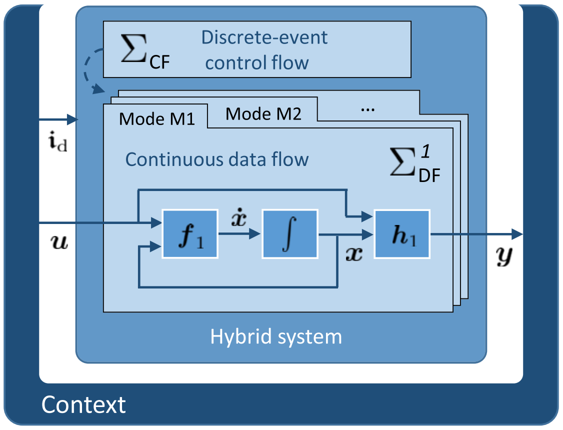


Figure 3: Hybrid system with continuous and discrete event component, discrete-value control flow input $\mathfrak{i}_{\mathrm{d}}$, as well as continuous input and output signals $\boldsymbol{u}$ and $\boldsymbol{y}$.

of a mode i. The initial value $\boldsymbol{x}_i(t_{\text{Start}})$ in (1) is initialized (if required) as a function of the continuous state $\boldsymbol{x}_{i^-}$ of the preceding mode i^-.

The control flow component (2) defines the mode switches with the events $e_j \in \{\text{TRUE}, \text{FALSE}\}$, $j = 1, \ldots, N_E$ and thus represents the switching logic of the considered system. The notation in (2) is based on the notation used in [36] for input/output machines, cf. also [25]. Here i^+ denotes the successor mode of the current mode i. In addition to possible discrete-value control flow inputs in the vector $\mathfrak{i}_d$, the events are defined by a mode-independent mapping $\mathfrak{z} = \mathfrak{z}(\boldsymbol{u}, \boldsymbol{y}, \tilde{\boldsymbol{y}}, t)$. The latter describes a discrete-value partitioning of the continuous signals and their assignment to logical alternatives on the basis of the procedure explained in Section 2.2 below. The mode-independent signal $\tilde{\boldsymbol{y}} = \tilde{\boldsymbol{h}}_i(\boldsymbol{x}_i)$ represents an extended output of the system (1), which supplements the data flow of each mode (1) by the signals that contribute to the event conditions.

For the special cases that events in (2) depend only on exogenous signals or on time t, i. e., $e_j = e_j(\mathfrak{i}_d, \mathfrak{z}(\boldsymbol{u}, t))$, so-called explicit switching occurs. If the events e_j depend on $\tilde{\boldsymbol{y}} = \tilde{\boldsymbol{h}}_i(\boldsymbol{x}_i)$, implicit switching arises, so-called guards, cf. [36].

From the point of view of system dynamics, the approach (1)–(2) is comparable to the use of hybrid automata [14, 36], but further combines and expands this framework with the systematic derivation of modes and mode transition conditions on the basis of the Essential Analysis discussed in the following section.

2.2 Management of combinatorial complexity

Essential Analysis according to SCODE is a design method for the systematic structuring of a problem over a discrete space. It is assumed that a complex problem can be broken down into smaller subproblems, which become easier to understand, solve and implement on the one hand, and have increased maintainability and simplified extensibility on the other hand. Due to the generic character of the SCODE method it can be used for both initial development and the restructuring of existing systems.

The objective of the Essential Analysis is the unique and consistent derivation of operating modes and corresponding mode transitions in order to describe the discrete event behaviour Σ_{CF} (control flow) of the hybrid system under consideration, cf. Figure 3. It is assumed that the respective system context is known and that the system boundaries are defined. Essential Analysis is carried out in three steps:

1. Definition of input and output space (problem space)
2. Derivation and definition of operating modes
3. Definition of mode transitions.

The individual steps are explained below and illustrated in Section 4 with the control logic for a hybrid vehicle.

2.2.1 Definition of input and output space

The description and coverage of the problem space is carried out via a morphological box [26], also known as a Zwicky-Box,[3] cf. Figures 4 and 6. The filling of the Zwicky-Box spans the combinatorial possibilities of the underlying problem. Each relevant aspect (feature) is entered as a dimension in the form of a line. The different manifestations of a dimension are captured in this line by so-called alternatives. The partitioning $\mathfrak{z} = \mathfrak{z}(\boldsymbol{u}, \boldsymbol{y}, \tilde{\boldsymbol{y}}, t)$ of the continuous signals into distinct value ranges, as introduced in (2), is thus systematically described in the Zwicky-Box formalism by using exclusive alternatives of a Zwicky-Box dimension. The size of the combinatorial problem space therefore results from the product over the numbers of alternatives per dimension.

Dimension	Alternatives			
Dim A	a1	a2	a3	a4
Dim B	b1	b2		
Dim C	c1	c2	c3	

Figure 4: Zwicky-Box with two input dimensions Dim A and Dim B and an output dimension Dim C (marked yellow).

The identification and naming of the relevant dimensions and alternatives for the technical system under consideration represents primarily a creative process and is to be understood in the sense of an expert discussion. In addition to SCODE analysts, domain experts are in particular required as participants – experience has shown that the optimal size is 4 to 6 persons. This form of structured analysis has a decisive advantage over other methods since it leads in general to both a compact representation and a deep understanding of the system, thus enabling the target-oriented discovery of solution approaches and the disclosure of open questions.

3 According to the Swiss astronomer Fritz Zwicky * 1898 †1974.

With regard to the definition of modes and events in the second and third analysis step, the dimensions considered are divided into *input dimensions* and *output dimensions*. Input dimensions form the decision space $\mathcal{E}$, where valid subspaces of modes and events are defined. Output dimensions represent the space $\mathcal{A}$ of the possible actions and are used to describe the behavior of a mode. Each mode thus corresponds to a subspace of the problem space $\mathcal{M} = \mathcal{E} \cup \mathcal{A}$. The mode definition is composed of its area of validity in the input space $\mathcal{E}$ and the description of its behavior in the output space $\mathcal{A}$. The alternatives of each dimension of the input space should be exclusive, i. e., they must not overlap. At runtime, exactly one alternative of each input dimension is valid at any time.

2.2.2 Definition of modes

In the second step of the analysis, the system modes are defined by selecting the relevant alternatives for each dimension. A combination of selected alternatives is called rule,[4] e. g.,

$$r_1 = (\,\textsc{Dim}\ \mathrm{A} == \{\mathrm{A1} \mid \mathrm{A3}\} \wedge \\ \textsc{Dim}\ \mathrm{B} == \mathrm{B2}\,). \tag{3}$$

The selected alternatives within a dimension are linked with OR, those between dimensions with AND. A mode definition m_i consist of at least one rule r_k of the form (3). If several rules are used together in a mode definition, they are linked by OR:

$$m_i = \bigvee_{k \in \mathcal{R}_{\mathcal{M}}(i)} r_k \quad \text{with } i = 1, \ldots, N_\mathrm{M}, \tag{4}$$

in which $\mathcal{R}_{\mathcal{M}}(i)$ specifies the index set of rules relevant to a mode i.

In a SCODE mode system, exactly one mode is always active at runtime and each mode usually has a unique output combination. The tool for SCODE analysis presented in Section 4.3 ensures that the modes do not overlap (consistency) and that the input space is completely covered (completeness). In other words the system reacts deterministically in every situation according to the defined behavior (*static check*). The systematic approach of the SCODE method thus considers all possible states of the system in the input space and all possible combinations in the output space (the latter being optional). Subspaces that have not yet been considered are displayed. Such combinations may also be excluded as undesirable or physically impossible.

4 As far as a rule covers only input dimensions, it is strictly speaking a condition. For generalization, however, the term rule is used for both cases in the following.

2.2.3 Definition of mode transitions

In the third step of the analysis, transitions between system modes are defined in a mode transition table, which could take the following form:

Mode transition table for an example system with three modes **M1**, **M2**, **M3** and four events $e_1, \ldots, e_4$.

	successor mode i^+		
current mode i	**M1**	**M2**	**M3**
M1	▨	e_1	e_2
M2	e_3	▨	–
M3	–	e_4	▨

In the mode transition table, it is determined for the set of all directed mode pairs whether a transition should be possible and under which event

$$e_j = \bigvee_{k \in \mathcal{R}_{\mathcal{E}}(j)} r_k \quad \text{with } j = 1, \ldots, N_\mathrm{E}, \tag{5}$$

cf. (2), it will occur. The N_E required events (5) are again defined using rules r_k on the basis of the Zwicky-Box. For this purpose, only the decision space $\mathcal{E}$ of the input dimensions may be used. The check of the mode transitions as to their completeness (all events considered) and consistency (events fit to the target mode) is also possible using tool support (*dynamic check*).

Essential Analysis is not limited to *switch-case* systems where the currently valid mode results statelessly from the actual input values. Rather, the current mode of a switching system is generally also a function of the previously valid mode (state-depending system). This ranges from simple hysteresis to finite automata (*state machines*). Essential Analysis also allows for a formal check on completeness, on absence of overlaps between modes not involved in the hysteresis, as well as on uniqueness of the mode transitions. In practice, several analysis approaches concerning state-dependent systems are available, e. g., overlapping modes (as shown in Section 4.1), systematic consideration of actions in the transition (*transition actions*) or the insertion of the previously valid mode i^- into the vector $\mathbf{i}_\mathrm{d}$, i. e., as a further input dimension of the Zwicky-Box.

2.3 Semantics and abstraction

In order to systematically span a problem space in all its combinatorial possibilities, the essential dimensions for the solution of the respective task are sought. In this respect, the choice of the dimensions corresponds to a problem-specific semantics taking the respective context into account. If a certain behavioural aspect of a system is to be investigated, it is often sufficient to choose a suitable abstraction for the analysis. Since this generally does not require refinement to the level of concrete signals and state machines, the corresponding detail aspects can be neglected. This reduces the complexity of the system under consideration to a manageable order of magnitude.[5] However, the discovery of an abstract view (*abstraction level*) suitable for the respective problem poses the greatest challenge. Essential Analysis helps here, since an unsuitable abstraction level usually becomes apparent at an early stage. Exemplary questions for the derivation of corresponding Zwicky-Box dimensions are as follows (cf. the context diagram in Figure 5):

- Which physical cause-effect topologies of the controlled plant can be distinguished?
- Which discrete-value user requests have to be considered?
- Which discrete states of the environment can occur that influence the controlled plant and corresponding cause-effect topologies?
- Which discrete event aspects of continuous quantities, e.g input, state, and output constraints [4, 17, 20], have to be considered?

The choice of a suitable abstraction represents an essential aspect of the application of the SCODE method, both for the analysis of existing systems (refactoring) as well as for the structuring of a new system to be built. It must be emphasized that mixed continuous/discrete-event aspects can occur on the most different abstraction levels of the development process: during modeling at the component and system level, in course of control function design, in its resulting algorithmic solution structure, as well as during the actual code implementation.

5 Practical experience has shown that the structuring of individual functions should ideally lead to a number of 7 ± 2 modes. The size 7 ± 2 is based on works from psychology and is generally regarded as a rule of thumb for orders of magnitude controllable by humans [37].

3 Structuring principles

In the preceding sections, Essential Analysis was introduced considering the functional design for a single system. However, real implementations of control systems or their software models generally consist of a large number of subsystems and corresponding individual functions.[6] In order to master the complexity of such systems, the well-known principles of modular and hierarchical structuring are available in addition to abstraction technologies, see Sections 3.1 and 3.2. However, only by integrating and combining these principles into a system view, cf. Section 3.3, does the SCODE method become an instrument which allows the mastering of realistically large networked systems.

3.1 Modular structures

The structural principle of *modularization* is used both for continuous data flow and for event-oriented control flow and divides the system under consideration into concurrent subsystems.

Modularization in the context of *data flow* is dividing the system into permanently and simultaneously calculating units in the sense of an analog computer and is usually represented using the familiar pattern of block diagrams. In this context, the individual blocks may represent locally separated parts of a physical system or the subfunctions of a control function approach. Using the example of a two degrees of freedom control design [41], the respective blocks would be *feedforward control*, *trajectory planning*, and *feedback control*. Modularization in the context of *control flow* refers to the derivation of concurrently switched mode systems, e. g., in the form of parallel automata.

The principle of encapsulation (also referred to as *information hiding* [32]) is decisive for modular structuring. Each subsystem communicates only via its interfaces in control and data flow and no implicit assumptions must be made about the internal realization of a subsystem. In the case of concurrent *data flow*, temporal consistency must primarily be ensured. Concurrent *control flow*, on the other hand, addresses switches whose combinatorial interactions are often not systematically addressed from the beginning of the controller design, and therefore later lead to ad hoc extensions (so-called *balconies*) and to software that is difficult to maintain. A comprehensive view of

6 For instance, engine control units contain up to several thousands of functions.

concurrent switching is indispensable if the switching of a module shows interactions with other modules. In this case, an analysis of the combinatorics on a suitable abstraction level is necessary [40]. The modes of the superordinate system as well as those of the subsystems are both determined by Essential Analysis.

3.2 Hierarchical structures

A further mechanism for the aggregation of aspects, besides modularization, is *hierarchical structuring*. The concept of hierarchy can as well be found in both control flow and data flow. Hierarchical block diagram structures (*A block contains a block diagram*) are a standard tool for structured design of data flows.

Hierarchy also occurs as a control concept in the context of cascaded control loops. These can as well be extended to cascaded 1-to-*n* structures in the sense of the automation pyramid for plant-wide control concepts [21, 22, 33, 34, 39]. There, a superimposed coordinator specifies the reference variables for *n* subordinate control loops. The cascading of control engineering structures is based on different time constants and thus on a separation of the time scales of subsystems.

Hierarchical structuring of *control flow* occurs when modes are combined into a common mode at a higher level. The concept of hierarchical state machines is available for this purpose. However, this concept often also leads to monolithic systems that are difficult to maintain. Instead, the principle of encapsulation again proves to be important: direct transitions to inner modes of another mode system (*inter-level transitions*) should be avoided. Encapsulation of the control flow also allows the application of Essential Analysis in this context (separately for each hierarchy level). An Essential Analysis across hierarchy levels remains possible by defining corresponding couplings with physical meaning.

3.3 System view

A system view [11, 23, 33] combines modular and hierarchical structuring principles and requires a stringent analysis and synthesis of continuous data and discrete-event control flow in the sense of the proposed system co-design concept.

From a control engineering perspective, such system-level approaches are often subsumed under the term of *plant-wide control*. One structuring approach for plant-wide control, known from production as well as process engineering, is provided by the paradigm of the automation pyramid [6, 21]. The control requirements are subdivided hierarchically into a company operating level, a pro-

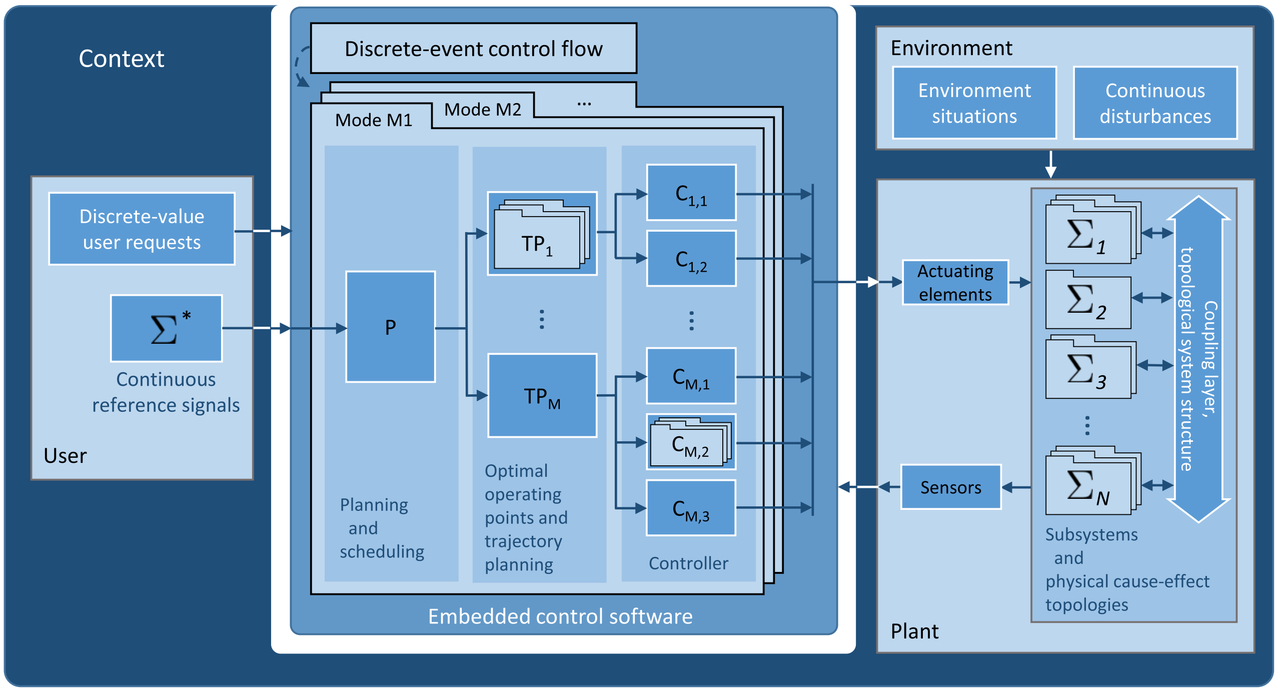


Figure 5: Representation of a cascaded block diagram structure with hierarchical 1-to-*n* planning coordinators as hierarchically encapsulated, discrete-event mode systems for the management of networked (and possibly decentrally controlled) coordinate sub-systems.

Dimension	Alternatives			
ICE ACTUAL STATE	OFF	ON	TRANSITION	
ICE DESIRED STATE	OFF	ON		
ICE CONNECTION TO WHEELS	DISCONNECTED	CONNECTED	TRANSITION	
SYNC TARGET	IDLE SPEED	GEARBOX SPEED		
OPERATION	DRAG START	SOFT STOP	ELECTRIC DRIVE	HYBRID DRIVE

Figure 6: Zwicky-Box for the combustion engine control. The four input dimensions span the input space with 36 possible combinations (3 · 2 · 3 · 2), while the output space consists of four possibilities.

cess control level, and a control and regulation level (or *field level*). From an algorithmic perspective, planning and coordination problems [7, 28], computation of optimal operating points and reference signals [31, 34, 39], as well as classical control engineering tasks [9, 10, 18, 24, 41] are assigned to the three levels mentioned above. In the block *Embedded control software* in Figure 5, such an approach can be seen.[7]

All three levels typically exhibit phenomena of mixed continuous/discrete-event type. The SCODE method offers a structuring approach that systematically captures the control flow portion of mixed continuous/discrete-event (hybrid) models and can therefore be applied to all of the above-named levels. Thereby, it is essential to encapsulate the modules and hierarchy levels as well as to ensure logical consistency of the data flows on the one hand and correct mode combinatorics of the subsystems and sublevels on the other hand. By linking Essential Analysis with the shown principles for the design and development of control software, significant efficiency gains in maintainability, extensibility, testability and variant stability of ECU software can be achieved.

4 Application and tool support

This section presents the application of the SCODE method to two functions of the control logic of an axle-split hybrid vehicle [13] characterized by a front wheel drive via an internal combustion engine and a rear wheel drive via an electric motor. According to the discussion in Section 3, the function *Engine control* described in Section 4.1 is hierarchically superimposed on the function *Combustion start* in Section 4.2. In order to serve as an introductory presentation of the method, the two applications are structurally simplified.

7 The different levels of the automation pyramid are depicted horizontally in Figure 5.

4.1 Engine control

The *Engine Control* function determines the operating state of the Internal Combustion Engine (ICE) in the hybrid vehicle. Depending on the requirements of the vehicle strategy and the current context (engine stopped, transmission on/off, speed synchronized), the combustion engine is operated and switched on or off. As in Section 2.3, the relevant discrete dimensions of the task are composed of the requirements from the context (*user*) as well as the discrete state components of the controlled system and are shown for the first step in the form of a Zwicky-Box in Figure 6. The four input dimensions span the space of the operating situations and are supplemented by the output dimension OPERATION with regard to possible output signals of the control.

Based on this Zwicky-Box, four modes were defined in the second step, which in this case are identical to the four alternatives of the output dimension OPERATION: Combustion start (**DRAG START**), Combustion stop (**SOFT STOP**), Combustion off (**ELECTRIC DRIVE**) and Combustion running (**HYBRID DRIVE**). For example, the rule r_1 of the mode **DRAG START** reads as

$$\begin{aligned} r_1 = (\ & \text{ICE DESIRED} == \text{ON} \ \wedge \\ & \text{ICE STATE} == \{\text{OFF} \mid \text{TRANSITION}\} \ \wedge \\ & \text{OPERATION} == \text{DRAG START}\ ). \end{aligned} \tag{6}$$

When defining the four modes, an overlap between the modes **DRAG START** and **HYBRID DRIVE** was found,[8] which represents a hysteresis in the system: If the system was already in **HYBRID DRIVE** before, the system remains in this mode, otherwise it switches to the mode **DRAG START**. The conditions for this case can be easily displayed in a decision tree, see Figure 7.

In the third step of the Essential Analysis, the mode transitions were defined. The result is shown in Figure 8

8 The tool-assisted SCODE analysis shows whether the mode system has overlaps, see Section 4.3. If there are any, the tool displays which modes overlap and which rules cause the respective overlap.

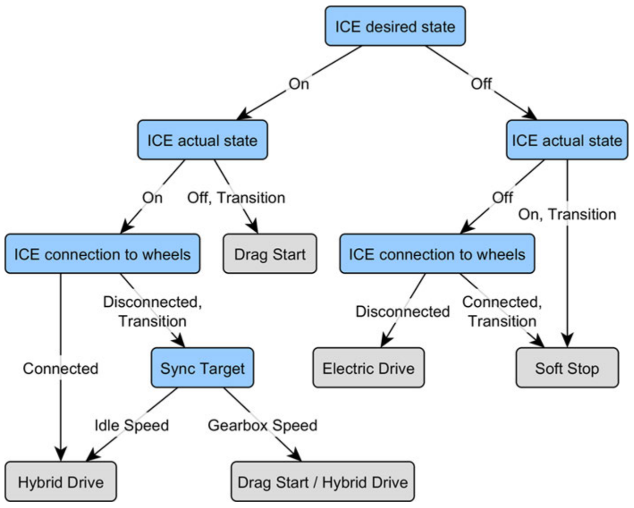


Figure 7: Decision tree of the combustion engine control (blue: input dimension, grey: mode). Due to the partial overlapping of the modes **Drag Start** and **Hybrid Drive**, these modes appear both in one sheet (bottom center).

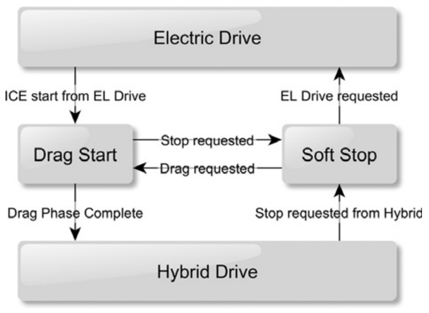


Figure 8: Mode-transition graph of the combustion engine control.

as a mode-transition graph. For example, the event Drag Phase Complete (e_1) determining the mode transition from **Drag Start** to **Hybrid Drive** consists of the two rules

$$r_2 = (\text{ICE DESIRED} == \text{ON} \wedge \text{ICE STATE} == \text{ON} \wedge \text{ICE-WHEELS} == \text{CONNECTED}), \tag{7}$$

$$r_3 = (\text{ICE DESIRED} == \text{ON} \wedge \text{ICE STATE} == \text{ON} \wedge \text{ICE-WHEELS} == \{\text{DISCONNECTED} \mid \text{TRANSITION}\} \wedge \text{SYNC TARGET} == \text{IDLE SPEED}), \tag{8}$$

which are linked with $e_1 = r_2 \vee r_3$.

The static overlap of the modes is resolved by the definition of the mode transitions within the dynamic analysis. Unintended mode transitions are also marked in the transition table. These rules define for each mode the conditions under which no mode change shall take place (*non-transition*). For the mode **Hybrid Drive**, the corresponding *non-transition* rule is given as

$$r_4 = (\text{ICE DESIRED} == \text{ON} \wedge \text{ICE STATE} == \{\text{OFF} \mid \text{TRANSITION}\}). \tag{9}$$

This prevents the transition from **Hybrid Drive** to **Drag Start**, see Figure 8, and the system remains in the mode **Hybrid Drive**.

The decisive advantage of the SCODE analysis is that a subsequent control engineering design can then be carried out separately for each mode. This procedure is much simpler because only those surrounding conditions and behaviors that are valid in the course of the respective mode have to be taken into account. Accordingly, the hierarchically subordinate behavior of a mode may again be described using an Essential Analysis. This is exemplarily discussed in the following section.

4.2 Combustion start

The function *Combustion start* models the starting of the combustion engine and thus represents a hierarchical subsystem of the mode **Drag Start** from Section 4.1. This application is a pure sequence control: The mode system runs through several modes, after which, on termination, further control flow is again taken over by the hierarchically superior mode system from Section 4.1. This function supports a **Normal Start**, and also an accelerated one (**Power Start**). Depending on the currently valid situation (context), intermediate steps can be skipped or a change from **Normal Start** to **Power Start** can occur.

By applying the SCODE method, all modes and desired processes, for example a possible *change-of-mind* between **Normal Start** and **Power Start**, could be discussed with the experts and defined completely and precisely during the course of function development. Figure 9 shows the modes and all possible transitions between them.

The SCODE analysis forces systematic consideration of the combinatorics of the entire system and thus helps to find and treat critical peripheral cases early in the development process. As a result, the control flow of the system is specified and documented completely, statically as well as dynamically.

4.3 Tool support

The application of the SCODE method for the analysis of complex technical systems requires the use of a tool to

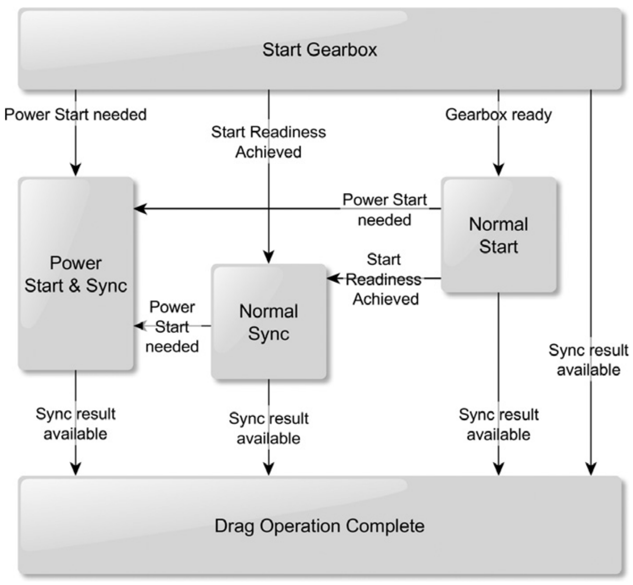


Figure 9: Mode-transition graph of the *Combustion start*.

handle combinatorics involved. Bosch/ETAS uses a development tool – SCODE-ANALYZER [12] – that supports the three-step procedure described above. A mode definition (4) is exemplarily shown in Figure 10 by selecting appropriate alternatives.

When defining the modes and the mode transitions, the tool already checks the corresponding mode definitions (4) and event rules (5) for completeness, uniqueness and consistency (static and dynamic check, see Section 2.2 and Figure 10).

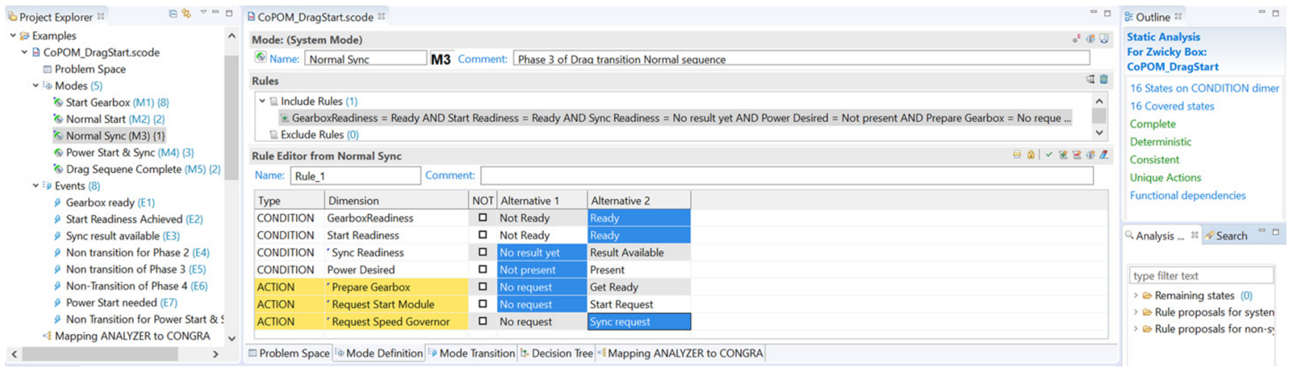
The algorithms for performing the mode analyses also scale for very large input and output spaces and their combinatorics. Experience shows that the limiting factor is the human and not the tool. Systems with a combinatorial input space of 10^8 and more elements can be analyzed without problems in practice. For very complex systems, a further substructuring into parallel or hierarchical mode systems has been proven successful, cf. Section 3.

An essential contribution to model-based development is provided by the code generation of the mode switching logic as well as the generation of test cases which allow the testing of all possible paths through the mode switching. The tool offers integration possibilities into existing development environments such as ASCET or Matlab/Simulink as well as export possibilities for different programming languages, e. g., C, see [12]. Decision trees and mode-transition graphs (see Figures 7 and 9) as well as an export to different report formats (PDF, HTML or Microsoft Office) are available as visualization and documentation options.

5 Summary and outlook

For the engineering of mechatronic systems, the SCODE method represents an important analysis and synthesis tool for the logical control flow of switched systems and controls. It should be emphasized that this approach can be used both for new development and for the restructuring of existing software. A major benefit of the method is that design decisions are specified in a compact and legible manner and that subsequent extensions can be easily inserted into the control structure, taking the relevant data flow into account.

In the development of control software at Bosch, the SCODE method was successfully applied to systems and abstractions with combinatorial spaces of 16 and more dimensions. It is only through the representation and derivation of the modes via morphologies (Zwicky-Boxes) in the SCODE method that the inherent system complexity can be abstracted to a controllable order of magnitude. Thereby,



Mode: (System Mode)

Name: Normal Sync **M3** Comment: Phase 3 of Drag transition Normal sequence

Rules

Include Rules (1)

GearboxReadiness = Ready AND Start Readiness = Ready AND Sync Readiness = No result yet AND Power Desired = Not present AND Prepare Gearbox = No reque ...

Exclude Rules (0)

Rule Editor from Normal Sync

Name: Rule_1 Comment:

Type	Dimension	NOT	Alternative 1	Alternative 2
CONDITION	GearboxReadiness	☐	Not Ready	Ready
CONDITION	Start Readiness	☐	Not Ready	Ready
CONDITION	' Sync Readiness	☐	No result yet	Result Available
CONDITION	Power Desired	☐	Not present	Present
ACTION	' Prepare Gearbox	☐	No request	Get Ready
ACTION	' Request Start Module	☐	No request	Start Request
ACTION	' Request Speed Governor	☐	No request	Sync request

Static Analysis
For Zwicky Box:
CoPOM_DragStart

16 States on CONDITION dimer
16 Covered states
Complete
Deterministic
Consistent
Unique Actions
Functional dependencies

Remaining states (0)
Rule proposals for systen
Rule proposals for non-sy

Figure 10: Screenshot ETAS SCODE-ANALYZER [12]: Definition of mode **Normal Sync** from subsystem *Combustion start*, see Fig. 9. On the right side, all static analysis results (here in green) are permanently updated.

important logical system properties of the control flow level, such as guaranteed completeness or consistency, can be verified (*correct-by-construction*). The fact that analysis and formal representation already take place early in the design process (*frontloading*), as well as the tool-supported code and test case generation, lead to a reduction of implementation and test effort.

With regard to the development of industrial synthesis tools for (e. g., modular and hierarchical) controllers [8], a formal linking of the presented approach with the continuous dynamics is necessary. One such contribution is the tool SCODE-CONGRA [12], which allows for a systematic structural analysis of the occurring continuous equation systems with constraint graphs. Furthermore, the combination with modern system theoretical approaches, e. g., model predictive control [9, 20, 22, 29, 38], offers relevant questions and topics for future research and development [1].

Acknowledgment: The authors would like to thank the participating colleagues from Bosch Research and ETAS GmbH as well as Arno Hafner and his colleagues from Bosch Powertrain Solutions for the application example.

References

1. F. Allgöwer, J. Borges de Sousa, J. Kapinski, P. Mosterman, J. Oehlerking, P. Panciatici, M. Prandini, A. Rajhans, P. Tabuada and P. Wenzelburger. Position paper on the challenges posed by modern applications to cyber-physical systems theory. *Nonlinear Analysis: Hybrid Systems*, 34:147–165, 2019.
2. M. Behle, A. Leonhardi and A. Mayer. Increasingly complex systems demand new software engineering methods. Technical report, ETAS GmbH, 2018. https://www.etas.com/en/downloadcenter/26949.php.
3. M. Behle and A. Mayer. New paths in software engineering. Technical report, ETAS GmbH, 2018. https://www.etas.com/en/downloadcenter/28739.php.
4. A. Bemporad and M. Morari. Control of systems integrating logic, dynamics and constraints. *Automatica*, 35:407–427, 1999.
5. M. Bitzer, M. Herrmann and E. Mayer-John. System Co-Design (SCODE): Methodik zur Analyse hybrider Systeme. *at – Automatisierungstechnik*, 67(9):739–750, 2019.
6. J. Busch, J. Oldenburg, M. Santos, A. Cruse and W. Marquardt. Dynamic predictive scheduling of operational strategies for continuous processes using mixed-logic dynamic optimization. *Computers and Chemical Engineering*, 31:574–587, 2007.
7. P. M. Castro, I. Harjunkoski and I. E. Grossmann. Optimal scheduling of continuous plants with energy constraints. *Computers and Chemical Engineering*, 35:372–387, 2011.
8. D. Elixmann, J. Puschke, H. Scheu, R. Schneider, I. J. Wolf and W. Marquardt. A software environment for economic NMPC and dynamic real-time optimization of chemical processes. *at – Automatisierungstechnik*, 62(2):150–161, 2014.
9. R. Findeisen, K. Graichen and M. Mönnigmann. Eingebettete Optimierung in der Regelungstechnik – Grundlagen und Herausforderungen. *at – Automatisierungstechnik*, 66(11):877–902, 2018.
10. M. Fliess, J. Lévine, P. Martin and P. Rouchon. Flatness and defect of non-linear systems: introductory theory and examples. *International Journal of Control*, 61(6):1327–1361, 1995.
11. E. D. Gilles. Network Theory for Chemical Processes. *Chem. Eng. Technol.*, 21:121–132, 1998.
12. ETAS GmbH. SCODE – Software for describing and visualizing complex closed-loop control systems, 2018. https://www.etas.com/en/products/scode.php.
13. Robert Bosch GmbH. Axle-Split-Hybrid Vehicle, 2018. https://youtu.be/bIN1fSFRkW8.
14. R. Goebel, R. G. Sanfelice and A. R. Teel. Hybrid dynamical systems: Robust stability and control for systems that combine continuous-time and discrete-time dynamics. *IEEE Control Systems Magazine*, 29:28–93, 2009.
15. D. W. Hoffmann. *Theoretische Informatik*. Hanser, 2018.
16. M. Johansson. *Piecewise Linear Control Systems*. Springer, 2003.
17. S. Joos, M. Bitzer, R. Karrelmeyer and K. Graichen. Constrained online-trajectory planning for nonlinear flat SISO systems using a switched state variable filter. *Automatica*, 110, 2019.
18. T. Kailath. *Linear Systems*. Prentice Hall, 1979.
19. F. Lamnabhi-Lagarrigue et al. Systems & control for the future of humanity, research agenda: Current and future roles, impact and grand challenges. *Annual Reviews in Control*, 43:1–64, 2017.
20. M. Löhning, M. Reble, J. Hasenauer, S. Yu and F. Allgöwer. Model predictive control using reduced order models: Guaranteed stability for constrained linear systems. *Journal of Process Control*, 24:1647–1659, 2014.
21. K. D. Listmann, P. Wenzelburger and F. Allgöwer. Industrie 4.0 – (R)evolution ohne Regelungstechnik? *at – Automatisierungstechnik*, 64(7):507–520, 2016.
22. J. Z. Lu. Closing the gap between planning and control: A multiscale MPC cascade approach. *Annual Reviews in Control*, 40:3–13, 2015.
23. J. Lunze. *Feedback Control of Large-Scale Systems*. Prentice Hall, 1992.
24. J. Lunze. *Regelungstechnik 1 und 2*. Springer, 2014.
25. J. Lunze. *Ereignisdiskrete Systeme*. De Gruyter, 2017.
26. J. Ölvander, B. Lundén and H. Gavel. A computerized optimization framework for the morphological matrix applied to aircraft conceptual design. *Computer-Aided Design*, 41:187–196, 2009.
27. A. Mayer, A. Meyer and M. Behle. New methods for control algorithm design. In *AUTOREG 2017: Automatisiertes Fahren und vernetzte Mobilität. 8. VDI/VDE-Fachtagung, Berlin, 5.–6. Juli*, 2017.
28. E. Mayer, K. Wulff, C. Horst and J. Raisch. Optimale Ablaufplanung für zyklische Prozesse mit Pooling-Ressourcen. *at – Automatisierungstechnik*, 56(4):181–188, 2008.
29. D. Q. Mayne. Model predictive control: Recent developments and future promise. *Automatica*, 50:2967–2986, 2014.
30. S. McMenamin and J. Palmer. *Essential Systems Analysis*.

Yourdon Press, New York, 1984.
31. J. Neupert, E. Arnold, O. Sawodny and K. Schneider. Tracking and anti-sway control for boom cranes. *Control Engineering Practice*, 18:31–44, 2010.
32. D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 15(12):1053–1058, 1972.
33. R. Scattolini. Architectures for ditributed and hierarchical Model Predictive Control – A review. *Journal of Process Control*, 19:723–731, 2009.
34. P. Tatjewski. Advanced control and on-line process optimization in multilayer structures. *Annual Reviews in Control*, 32:71–85, 2008.
35. G. Torres Ferreira, G. Tineli and M. Herrmann. Flex Fuel Software Maintainability Improvement: A Case Study. *SAE Technical Paper Series*, 2016-36-0214 E, 2016.
36. A. van der Schaft and H. Schumacher. *An Introduction to Hybrid Dynamical Systems*. Springer, 2000.
37. Wikipedia. The Magical Number Seven, Plus or Minus Two, 2019. https://en.wikipedia.org/wiki/The_Magical_Number_Seven,_Plus_or_Minus_Two.
38. I. J. Wolf and W. Marquardt. Fast NMPC schemes for regulatory and economic NMPC – A review. *Journal of Process Control*, 44:162–183, 2016.
39. L. Würth, R. Hannemann and W. Marquardt. A two-layer architecture for economically optimal process control and operation. *Journal of Process Control*, 21:311–321, 2011.
40. J. Zaytoon and B. Riera. Synthesis and implementation of logic controllers – a review. *Annual Reviews in Control*, 43:152–168, 2017.
41. M. Zeitz. Differenzielle Flachheit: Eine nützliche Methodik auch für lineare SISO-Systeme. *at – Automatisierungstechnik*, 58:5–13, 2010.
42. M. Zimmermann, T. Bleile, F. Heiber and A. Henle. Mastering the Complexity of Engine Control Functions. *Motorentechnische Zeitschrift MTZ*, 76:38–41, 2015.

Bionotes

Matthias Bitzer
Corporate Sector Research and Advance Engineering, Model-Based Systems Engineering (CR/AEE), Robert Bosch GmbH, Robert-Bosch-Campus 1, 71272 Renningen, Germany, bosch.com/research
matthias.bitzer2@de.bosch.com

Matthias Bitzer is senior expert for control engineering in the Corporate Sector Research and Advance Engineering of Robert Bosch GmbH and has contributed to the development of the SCODE method. He studied Engineering Cybernetics at the University of Stuttgart, Germany and at the University of Wisconsin-Madison, Madison, USA and was a research associate at the Institute for System Dynamics and Control of the University of Stuttgart.

Martin Herrmann
Corporate Sector Research and Advance Engineering, Model-Based Systems Engineering (CR/AEE), Robert Bosch GmbH, Robert-Bosch-Campus 1, 71272 Renningen, Germany, bosch.com/research
martin.herrmann@de.bosch.com

Martin Herrmann is senior expert for software and modeling languages in the Corporate Sector Research and Advance Engineering of Robert Bosch GmbH. He co-developed the SCODE method and the corresponding tools and applied them in different domains. He is currently involved in the modeling and verification of autonomous and learning systems. He studied electrical engineering at the University of Stuttgart, Germany and graduated with a diploma.

Eckart Mayer-John
Corporate Sector Research and Advance Engineering, Model-Based Systems Engineering (CR/AEE), Robert Bosch GmbH, Robert-Bosch-Campus 1, 71272 Renningen, Germany, bosch.com/research
eckart.mayer@de.bosch.com

Eckart Mayer-John is a researcher in the Corporate Sector Research and Advance Engineering of Robert Bosch GmbH and has been concerned with the software implementation of control functions for many years. He studied Engineering Cybernetics at the University of Stuttgart, Germany and the University of Exeter, UK, was a research associate at the Max Planck Institute for Dynamics of Complex Technical Systems in Magdeburg, Germany, and received his doctorate at the Department of Electrical Engineering and Computer Science at the TU Berlin.