Institut für Intermodale Transport- und Logistiksysteme

Technische Universität Braunschweig

Automation Engineering (Automatisierungstechnik)

Lecture Notes

Jürgen Pannek

January 8, 2026

Jürgen Pannek
Institute for Intermodal Transport and Logistic Systems
Hermann-Blenck-Str. 42
38519 Braunschweig

FOREWORD

During the winter term of 2025/26, I am teaching the Automation Engineering module at the Technical University of Braunschweig. These lecture notes have been prepared to structure the lecture and facilitate my students' learning process. Furthermore, regular updates are made to ensure the latest notes are available for the winter term.

The objective of this module is to provide students with a comprehensive foundation in the terms and methods relevant to automation engineering. Students will be able to reproduce, describe, and apply these concepts, and explain the modeling, classification, control, and coupling of technical processes using basic examples. They will also be able to analyze information handling and transfer in technical processes. Furthermore, students will be capable of determining the organizational, distribution, and communication structures of automation systems for simple case studies. Additionally, they will be able to describe the fundamental aspects of modularization, standardization, and automation. Students will gain an understanding of digitization topics such as the industrial internet, cloud computing, and cyber-physical systems. As a result, they will be able to reproduce approaches to knowledge management, industrial big data, and decision support.

To this end, we discuss

- Aim of automation engineering
- Basics, tasks and methods of automation
- Coupling and hierarchies of systems
- Information and information management
- Control, modularization and standardization in automation
- Digitalization for industrial internet, industrial could and CPS

- Basics of industrial big data, knowledge management and GraphRAGs

within the lecture and support understanding and application within the tutorial and laboratory classes. The module itself is accredited with 5 credits with an add-on of 2 credits if the requirements of the laboratory classes are met.
An electronic version of this script can be found at

https://www.tu-braunschweig.de/en/itl/teaching/lecture-notes

Literature for further reading

- Automation engineering basics
  - LUNZE, J.: *Automatisierungstechnik*. 5. Auflage. DeGruyter, 2020. http://dx.doi.org/10.1515/9783110465624
  - PLENK, V.: *Grundlagen der Automatisierungstechnik kompakt*. Springer, 2019. http://dx.doi.org/10.1007/978-3-658-24469-9
- Networks
  - ERLEBACH, T. ; BRANDES, U.: *Network analysis: Methodological foundations*. Springer, 2005. http://dx.doi.org/10.1007/b106453
  - NEUMANN, K. ; MORLOCK, M.: *Operations Research*. Hanser, 2002. http://dx.doi.org/10.1002/zamm.19940740918
  - EMMONS, S. ; KOBOUROV, S. ; GALLANT, M. ; BÖRNER, K.: Analysis of Network Clustering Algorithms and Cluster Quality Metrics at Scale. In: *PLOS ONE* 11 (2016), pp. 1–18. http://dx.doi.org/10.1371/journal.pone.0159161
- Digitalization, CPS and high level automation
  - LAI, C.: *Intelligent Manufacturing*. Springer, 2022. http://dx.doi.org/10.1007/978-981-19-0167-6
  - LANGMANN, C. ; TURI, D.: *Robotic process automation – Digitalisierung und Automatisierung von Prozessen*. Springer, 2020. http://dx.doi.org/10.1007/978-3-658-34680-5
  - STJEPANDIC, J. ; SOMMER, M. ; DENKENA, B.: *DigiTwin: An approach for production process optimization in a built environment*. Springer, 2022. http://dx.doi.org/10.1007/978-3-030-77539-1

Contents

List of Tables

List of Figures

List of Definitions and Theorems

CHAPTER 1

INTENSION, CONCEPT AND AIMS

Generated with deepai.org

The discipline of automation engineering pertains to the domain of monitoring and control of systems or processes. Conventionally, the latter pertains to physical activities such as mechanical processes involving machines and robots or process technology including bio- and chemical systems. In the context of contemporary automation, the automation of software processes assumes significant importance. Indeed, the physical and cyber worlds have become increasingly intertwined. The development of ecosystems driven by AI methods for intelligent outcomes is merely the latest of these aspects. Such approaches have been documented in both literature and practice since the 1970s and were designated as expert systems.

1.1. Intension

As previously stated, the discipline of automation engineering is concerned with the monitoring and control of processes. Conversely, the objective of the latter is to enhance or supplant a specific task. It is a common criticism of automation that it merely substitutes labour, yet this is not an entirely accurate assessment. Whilst the improvement or substitution of tasks has been shown to reduce the required workforce for the task itself, new tasks arise in areas such as integration, control and maintenance.
In history, automation engineering has gone through different stages:

- Approximately 800 B.C., the ancient Greek word *automaton* was first mentioned dealing with devices like automatically driving tripods or automated doors of temples. While these were implemented as feed forward, the first known feedback device is the water clock of Ctesibius (285 –222 B.C.). His invention remained the most accurate time measurement device until Huygens (1629 – 1695) invented the pendulum clock. Moreover, the first steam engine was developed by the Hero of Alexandria (10 – 70). Other basic examples can be found, e.g., in the Middle East, China, and Mayan cultures.

- The period we call the First Industrial Revolution (1760 – 1840) started with the re-invention of the steam engine (respectively its governor) by Watt (1736 – 1819) with improvements by Siemens (1823 – 1883) and the programmable loom by Jacquard (1752 – 1834). The final step in this period was the development of a formal description for controlling a process by Maxwell (1831 – 1879). This period is also called the mechanical revolution.

- Being able to design machines, the Second Industrial Revolution (1870 – 1915) was driven by the development of electricity and electric devices. Here, the focus moved from single machines to efficient manufacturing methods such as production lines and the complete work split called Taylorism. The period is also called the electrical revolution.

- The Third Industrial Revolution started around 1950 and focused on complex processes and digitalization. Considering automation, the programmable logic controller (PLC) was the major step in implementing advanced control methods industrially. This technical invention was backed up by theoretical innovations from Kolmogorov (1903 – 1987) on complexity, Kalman (1930 – 2016) on observers, and Bellman (1920 – 1984) on optimization. Additionally, the concept of a robot was introduced. As computers characterize it, this phase is also called the digital revolution.

- Lately, the Fourth Industrial Revolution is postulated, which utilizes interconnected and self-functioning systems.

Remark 1.1
Note that the above interpretation is focused on automation. Other classifications are used for different topics, such as transport, capitalism or social development.

Nowadays, automation systems are components of enterprise resource planning (ERP) tools, still combining information for monitoring and control. Figure 1.1 shows a current default path of automation. It includes the steps taken in the Second Industrial Revolution (standardization and

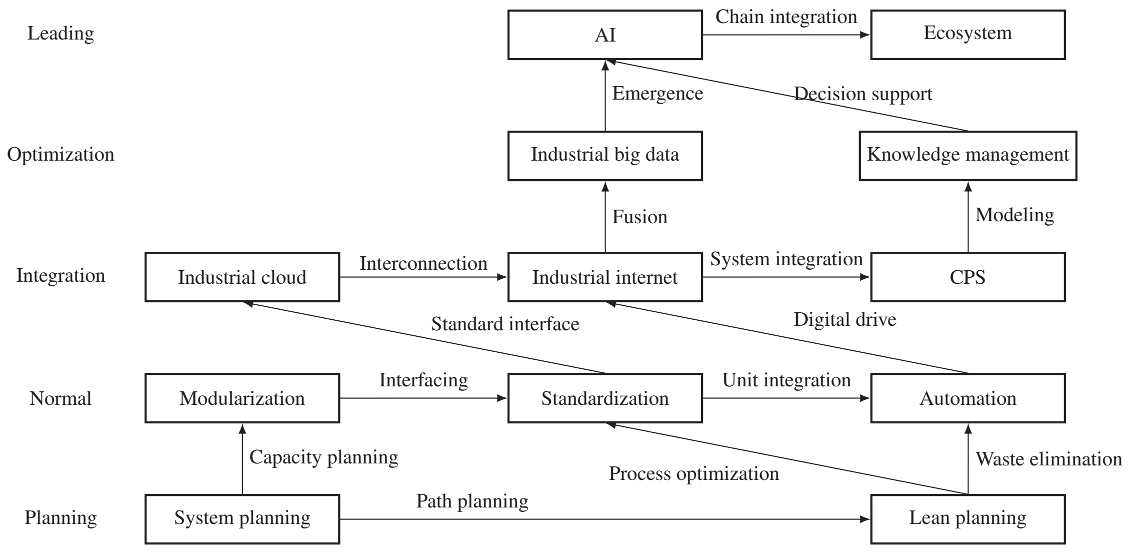


Figure 1.1.: Integrated implementation path of automation

modularization) and the Third Industrial Revolution (automation and lean). We like to point out that the layers *integration*, *optimization*, and *leading* also existed and were managed before the Forth Industrial Revolution. Yet now, the quantification of the five management tasks

- planning
- organizing
- staffing
- leading
- controlling

may be done much more accurately.

The fundamental question that must be addressed is the rationale behind the necessity of automation. The U.S. National Institute of Standards and Technology asserts that companies can adopt two distinct strategies in order to compete for market shares: the so-called cost leadership strategy and the differentiation strategy. These strategies are derived from the company vision and address the market segment. Consequently, alternative strategies have been developed to address the remaining enterprise components.
As demonstrated in Figure 1.2, the strategies can be categorised into four overarching goals: agility, quality, productivity and sustainability.

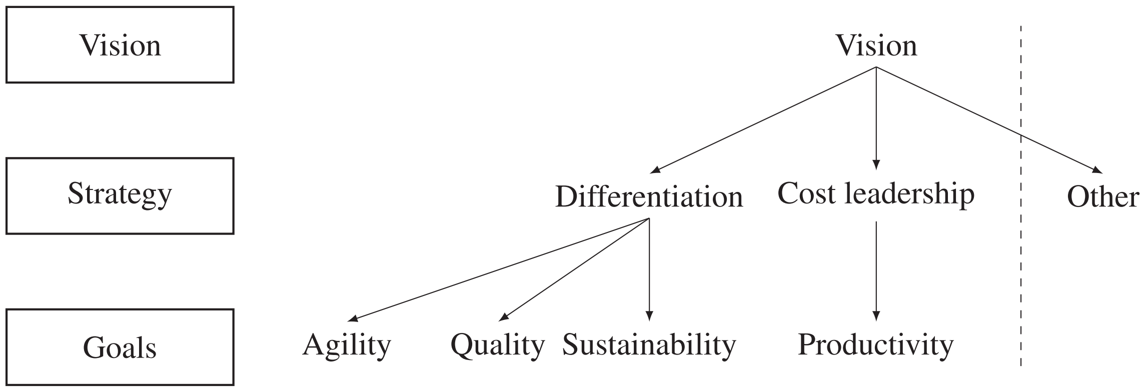


Figure 1.2.: Derivation of strategies and goals from vision

Remark 1.2
It is important to note that strategies vary significantly on a global scale. The availability of skills in different regions of the world is a fundamental driver of these trends.

- *In the United States, a significant number of prominent information technology companies are headquartered, and the National Manufacturing Innovation Network is engaged in the development of strategies that leverage artificial intelligence and substantial data sets.*
- *The Japanese Industrial Value Chain employs a CPS-based approach, leveraging its extensive history and expertise in robotics.*
- *The Chinese 2025 program leverages its substantial workforce to achieve cost leadership and promote continuous quality enhancement.*
- *The Chinese program may be found to be closely associated with the German Industrie 4.0 initiative, which emphasizes the utilization of advanced manufacturing techniques and the cultivation of a skilled workforce.*

In contrast to the Japanese and US programs, which initiate the process from AI to production, the Chinese and German programs commence at the level of production and proceed to AI.

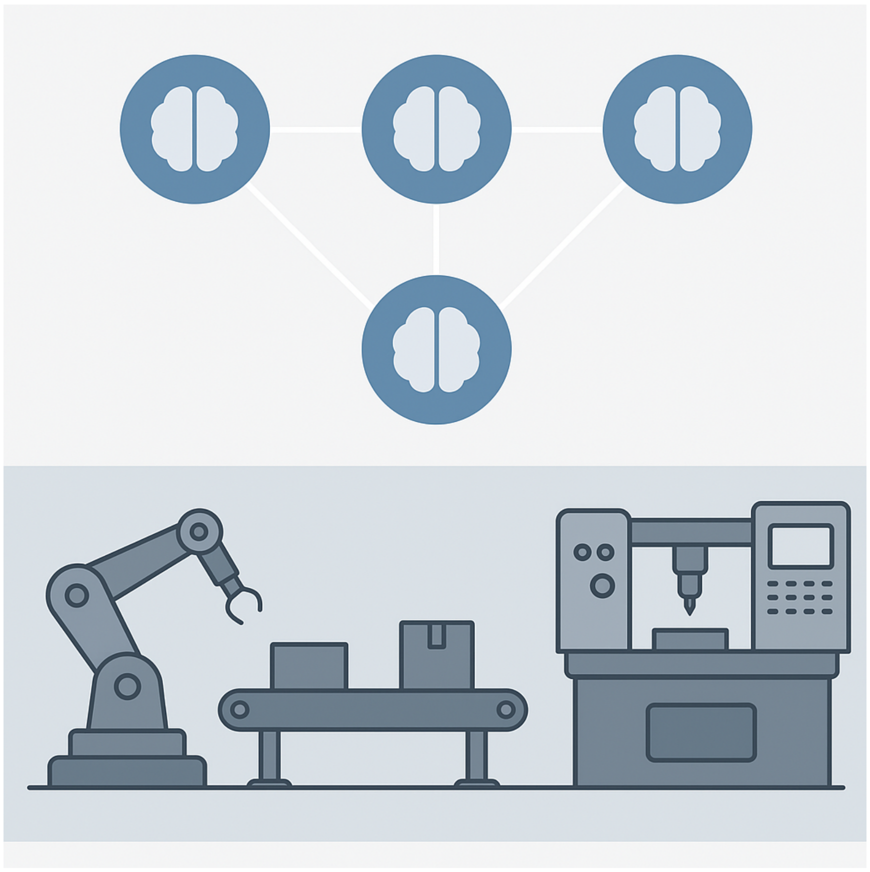


Figure 1.3.: Sketch of an agentic AI cyber-physical system — generated with chatgpt.com

In order to be successful, we need to balance these four goals. To this end, we have to define each goal's measures, which allows us to define SMART projects (specific, measurable, ambitious, realistic, terminated) and avoid waste or failure in architecture and implementation. The following Table 1.1 gives some examples of possible performance criteria.

Table 1.1.: Decomposition and measurement of key target capabilities

Strategy	**Goals**	**Capability decomposition**	**Performance measurements**
Cost leadership	Productivity	○ Production capacity ○ Overall equipment effectiveness ○ Mat./energy efficiency	○ Output per unit ○ Availability · performance · quality ○ Usage per unit output

Continued on next page

Table 1.1 – continued from previous page

Strategy	**Goals**	**Capability decomposition**	**Performance measurements**
		○ Labor efficiency	○ Labor hours per unit
Differentiation	Agility	○ Response speed	○ Response time, transaction cycle
		○ On time delivery	○ On time delivery rate
		○ Fault recovery	○ Downtime rate during operation
	Quality	○ Product quality	○ Return/rejection rate
		○ Innovation	○ Innovation cycle time
		○ Service	○ Customer evaluation
		○ Diversity	○ Product family and personalization options
	Sustainability	○ Product	○ Recyclability, energy efficiency, lifetime, reusability, manufacturability
		○ Process	○ Energy use, CO_2 footprint
		○ Logistics	○ Transport energy

Remark 1.3
For other KPIs, we refer to ISO 22400 [14], which is typically applied in the production context.

Having clarified the intention of automation, we next connect these to actual tasks.

1.2. Concept

The discipline of automation engineering is concerned with the design and implementation of systems and processes. It is important to note that a process can be defined as the manner in which a task is executed in its entirety.

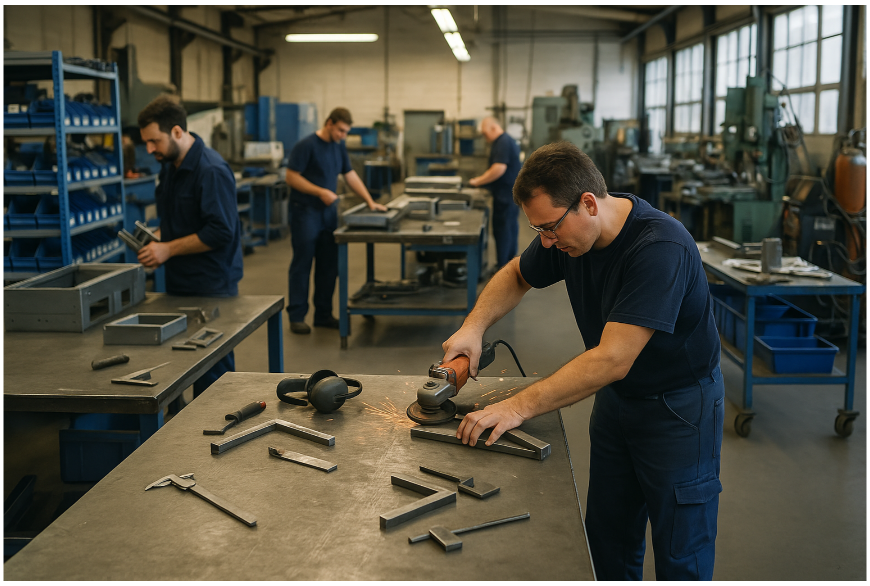


Figure 1.4.: Sketch of a jobshop — generated with chatgpt.com

In contrast, a system can be understood as the network of interconnected components that collectively facilitate the execution of a specific task. In many cases, these two terms can be used interchangeably.
In the literature [3], we see the following description for a system (translated from German):

> A system is a set of interrelated elements that are viewed as a whole in a particular context and considered as distinct from their environment.
>
> DIN IEC 60050-351 (2014)

Building on this description, a process is given as follows (translated from German):

> A process is the entirety of relations and interacting elements in a system through which matter, energy or information is transformed, transported or stored.
>
> DIN IEC 60050-351 (2014)

Within a process, we narrow down on elements within a system, which are connected via signals (translated from German):

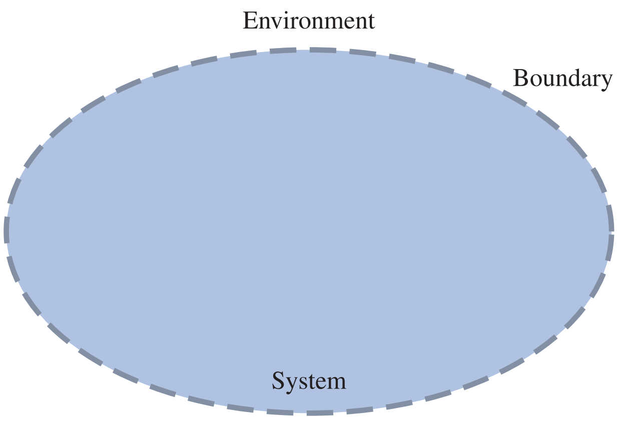


Figure 1.5.: Sketch of a system according to DIN IEC 60050-351 (2014)

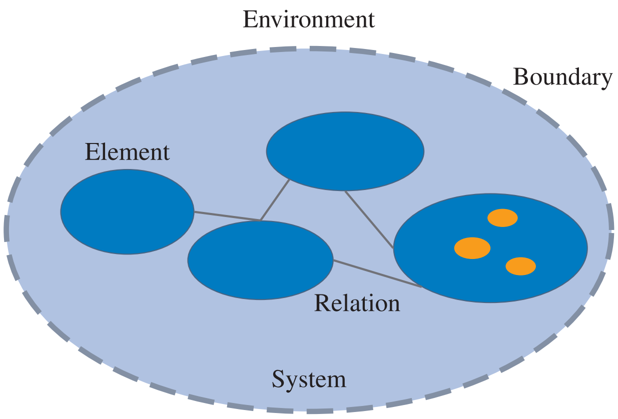


Figure 1.6.: Sketch of system and process according to DIN IEC 60050-351 (2014)

> A signal is a physical quantity that conveys information about one or more variable quantities using one or more of its parameters.
>
> DIN IEC 60050-351 (2014)

More formally, we define the following:

Definition 1.4 (System and process).
Consider two sets $\mathcal{U}$ and $\mathcal{Y}$. Then a map $\Sigma : \mathcal{U} \to \mathcal{Y}$ is called a *system*, and the application of this map to an input $\mathbf{u} \in \mathcal{U}$ to obtain an output $\mathbf{y} = \Sigma(\mathbf{u}) \in \mathcal{Y}$ is called a *process*.

Note that we extend the notion of a system by the interdependence of systems with their environments by so called *inputs and outputs*.

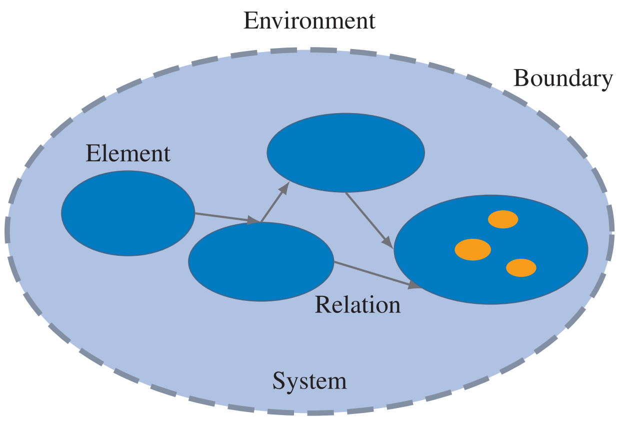


Figure 1.7.: Sketch of system, process and signal according to DIN IEC 60050-351 (2014)

This general form enables the treatment of a system as a black box, i.e. a system in which input and output are not fully understood. It should be noted that the present study does not specify whether the subject under discussion is an electric circuit, a robot, a production system or an entire circular supply chain. Concurrently, no statements are made regarding temporal parameters or sequences; consequently, the possibility of a delay between the insertion and extraction of units remains unresolved. It is evident that both an instantaneous analogue reaction to an electric current and a manufacturing delay resulting from sequential utilization of the machine are encompassed within the scope of consideration. Furthermore, the nature of the processes involving the entry and exit of materials remains ambiguous. This could potentially signify the movement of steel into the facility and automobiles out of it, akin to the process in automotive manufacturing. Alternatively, it may denote the flow of energy and information into the facility and its subsequent output, analogous to the dynamics observed in information systems.

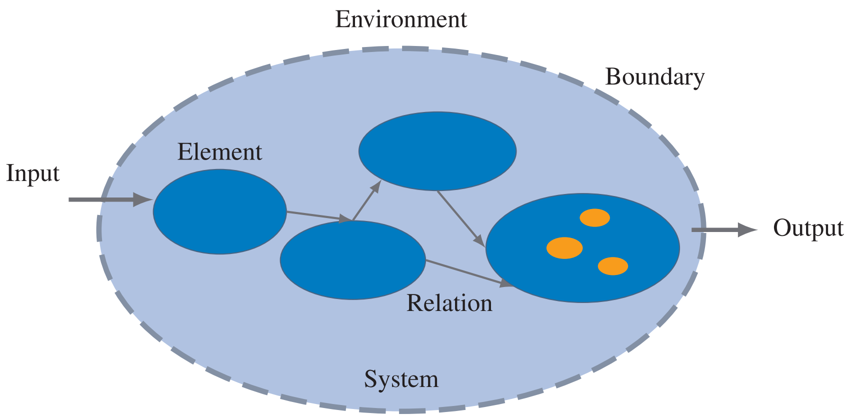


Figure 1.8.: System theoretic depiction of DIN IEC 60050-351 (2014)

Abstracting from Figure 1.8, a visual representation of both system and process is given in Figure 1.9.

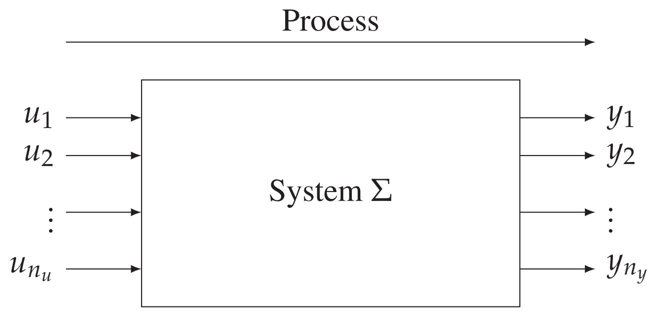


Figure 1.9.: Terms of a system and a process

More formally, the sets $\mathcal{U}$ and $\mathcal{Y}$ are called input and output sets. An element from the input set $\mathbf{u} \in \mathcal{U}$ is called an input, which acts from the environment to the system and is not dependent on the system itself or its properties. We distinguish between inputs, used to specifically manipulate (or control) the system, and inputs, not manipulated on purpose. We call the first ones *control or manipulation inputs*, and we refer to the second ones as *disturbance inputs*. An element from the output set $\mathbf{y} \in \mathcal{Y}$ is called an output. In contrast to an input, the system generates the output and influences the environment.

Link: For further details on how to design inputs/controls such that system properties regarding outputs can be generated, we refer to the lectures *Control Engineering 1 & 2*.

In order to comprehend and utilize a system or process, it is necessary to possess a model of it. It is important to note that models are only able to represent reality to a certain extent on a one-to-one basis. Should a model be employed, it is to be noted that discrepancies may emerge between the model's prediction and the actual outcome, particularly in the context of extended time horizons. The rationale underlying this phenomenon can be articulated as follows: In the field of modeling, the emphasis is placed on the aspects that are deemed to be of interest, with the objective being to provide a concise representation of the subject matter. Hence, the problem is split into two parts,

- the model, which describes what we are interested in,
- the environment, which contains everything else.

It is not possible to provide any meaningful insight into the environment, as it has not been modelled. Consequently, any interactions between the model and the environment can only be interpreted as disturbances.

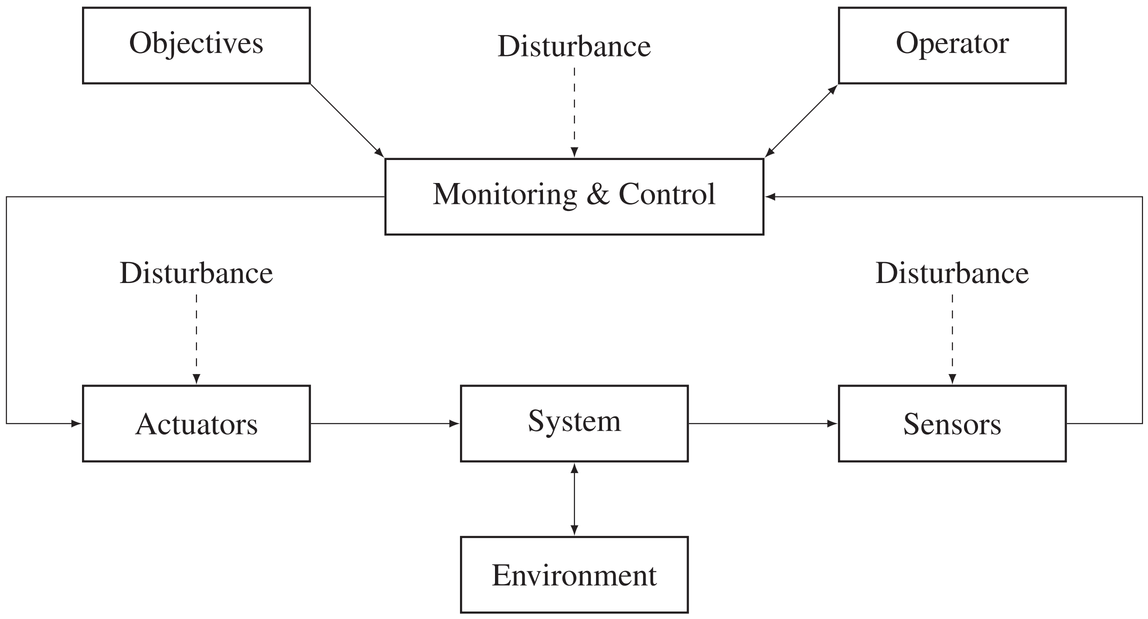


Figure 1.10.: Structure and process of automated systems

To utilize the term system and get to the standard concept of automated systems sketched in Figure 1.10, we first need to clarify basic terms.
The first of these is the notion of time.

Definition 1.5 (Time set).
A *time set* $\mathcal{T}$ is a subgroup of $(\mathbb{R}, +)$.

This definition is quite abstract, yet very generic. It allows us to use the concept of continuous time, discrete time and event time.

- Continuous time can be applied to systems operating in a continuous flow, e.g. pipelines, melting pots or biochemical reactors.
- Sensors and actors are typically not modeled in continuous, but in discrete time. As such, they follow a fixed routine as to when they send data regarding measurements or apply forces or rotate into certain positions.
- Last, event time is used to reduce the number of measurements, e.g. data is sent whenever a level rises above or falls below a certain threshold as in chemical reactors or heating systems.

Next, we introduce the so called *state of a system*.

Definition 1.6 (State).
Consider a system $\Sigma : \mathcal{U} \to \mathcal{Y}$. If the output $\mathbf{y}(t)$ uniquely depends on the history of inputs $\mathbf{u}(\tau)$ for $t_0 \leq \tau \leq t$ with $t_0, \tau, t \in \mathcal{T}$ and some $\mathbf{x}(t_0)$, then the variable $\mathbf{x}(t)$ is called *state* of the system and the corresponding set $\mathcal{X}$ is called *state set*.

Note that a state is not necessarily an output. It refers to an internal element, that can be affected by inputs and other states, and may influence outputs. Examples range from velocity of a vehicle as a state, which is influenced by the input acceleration and directly connected to the output position of the vehicle. However, it may also be a state of a finite automata, e.g. the payment status of ticket machine.
In continuous time, that is $\mathcal{T} = \mathbb{R}$, we obtain the standard description of a state space system:

Definition 1.7 (State space – continuous time system).
Consider a system $\Sigma : \mathcal{U} \to \mathcal{Y}$ in continuous time $\mathcal{T} = \mathbb{R}$ satisfying the property from Definition 1.6. If $\mathcal{X}$ is a vector space, then we call it *state space* and refer to

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t), t), \quad \mathbf{x}(t_0) = \mathbf{x}_0 \tag{1.1a}$$
$$\mathbf{y}(t) = h(\mathbf{x}(t), \mathbf{u}(t), t). \tag{1.1b}$$

as *continuous time system*. Moreover, $\mathbf{u}$, $\mathbf{y}$ and $\mathbf{x}$ are called *input*, *output* and *state* of the system.

Generally speaking, the (mathematical) description of models varies depending on the considered time, space and amplitude properties. Figure 1.11 provides a rough overview on these characteristics.

Remark 1.8
Regarding time, static models are characterized by the fact that inputs, outputs, and measurements of the system are available. In contrast to that, continuous time models exhibit data streams being received continuously. Discrete-time models differ from that by the availability of data, which is received at certain, not necessarily equidistant time instances. Last, event-triggered models require issues to trigger receiving data.
Regarding space, models may vary from a simple connection to complex systems.
Regarding amplitude, models may differ regarding continuous spaces e.g., mass, and discrete spaces such as gear shifts.

As described before, the input to a system splits into two parts $\mathcal{U}_1, \mathcal{U}_2 \subset \mathcal{U}$ with $\mathcal{U}_1 \cup \mathcal{U}_2 = \mathcal{U}$ and $\mathcal{U}_1 \cap \mathcal{U}_2 = \emptyset$, that is

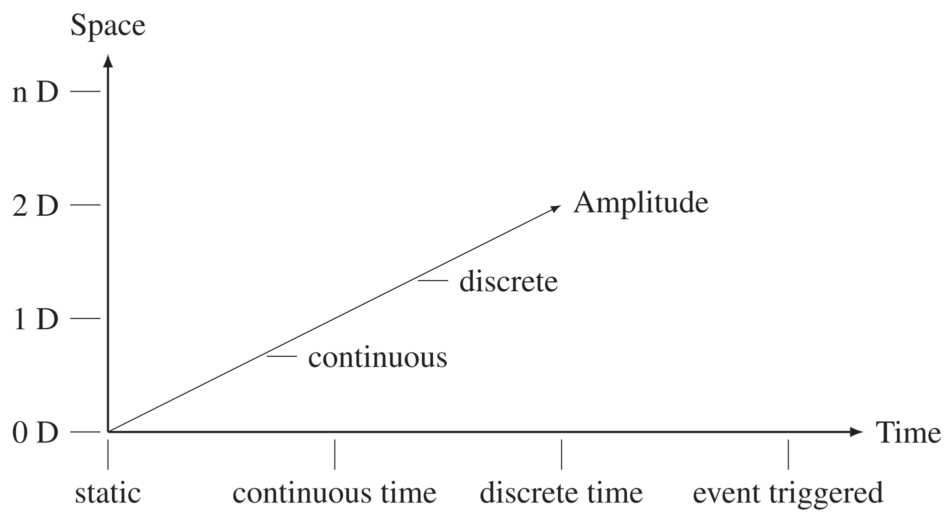


Figure 1.11.: Dimensions of model characteristics

1. the externally adjustable part $\mathbf{u} \in \mathcal{U}_1$ called control or actuator, and
2. the not influencable part $\mathbf{u} \in \mathcal{U}_2$ termed disturbance.

Since sensors and actuators are also modeled, unmodeled parts may enter the overall scheme via these components. Similarly, any monitoring and control device based on a computer is also subject to idealization, e.g., via using floating point approximations of numbers. Hence, another source of disturbances exists via this component.

Remark 1.9
Formally, each block within Figure 1.10 is a system in the sense of Definition 1.4.

The last two components, which we did not discuss so far, are *monitoring and control* and *operator*. The component monitoring and control is a system representing the inverse of the chain actuator – system – sensor, which is modified by the objectives and interacts with operators. Similar to the system itself, the monitoring and control system may exhibit an internal state. This block is typically implemented using computers or other devices such as PLCs. In the literature, the two following descriptions are found most regularly:

> A control is a process in a system in which one or more variables as input variables influence other variables as output variables due to the laws peculiar to the system.
>
> DIN IEC 60050-351 (2014)

> A control is a process in which a variable, which is to be controlled, is continuously recorded, compared with another variable, the reference variable, and influenced in the sense of an adjustment to the reference variable.
>
> DIN IEC 60050-351 (2014)

The difference between these two descriptions is given by their purpose: While both aim to influence the output, the first feeds the output from outside the system, but the second uses an external reference to feed the output back to that reference. For this reason, the first is called feed forward, and the second is feedback. More formally:

Definition 1.10 (Feed forward).
If an input is defined by a function $\mathbf{u} : \mathcal{T} \to \mathcal{U}$, then we call it to *feed forward*.

Definition 1.11 (Feedback).
If an input is defined by a function $\mathbf{u} : \mathcal{X} \to \mathcal{U}$, then we call it *state feedback*. It is called *output feedback* if it is defined as a function $\mathbf{u} : \mathcal{Y} \to \mathcal{U}$.

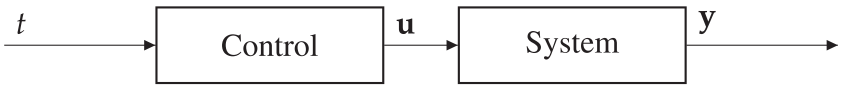


Figure 1.12.: Simple feed forward

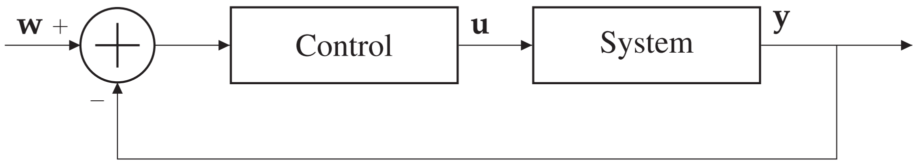


Figure 1.13.: Simple feedback

Control structures may be much more complex than the illustrated feed forward and feedback versions. Still, these are the only two main structures. A feed forward is typically used for planning, e.g. route generation using a navigational system in a car. Such a plan is idealistic, and on operational level needs to be adjusted to reality by feeding back sensor data to update the taken actions, i.e. a feedback structure implemented by a driver (or an AD system) in a car.

Last, the operator represents the human in the loop. The operator interacts with the monitoring and control system via a human-machine interface (HMI), which provides two key functions for the human: information and manipulation. In the literature [16], we find the following description:

> A user interface is all an interactive system's components (software or hardware) that provide information and controls for the user to accomplish specific tasks with the interactive system.
>
> ISO 9241 (2019)

Typical examples range from touchscreen interfaces of machines or in cars to light bulbs at production machines.
After introducing all relevant components within the automation structure of Figure 1.10, we can now look into details regarding the aims of automation.

1.3. Aims

In particular, we need to understand what is meant by automation. In the literature [2], we see the following definition (translated from German):

> Automation is equipping a device so that it works as intended, in whole or in part, without human intervention.
>
> DIN 19233 (1998)

Within this definition, we observe several components:

1. Intend: Each automation requires one or more criteria to assess whether or not it works as intended; see Table 1.1 for the derivation of such criteria.

2. Extend: Each automation requires a system boundary, that is, which parts of a system or process are automated and which are not. Moreover, it requires explicitly modeling the interfaces between automated and non-automated parts. These can be physical, like handovers from machines to storage or workers, or information to and from an operator, cf. Figure 1.10.

3. Attend: Each automation requires components to execute the monitoring and control tasks, that is, sensors, actuators, communication, and logic, cf. Figure 1.10 and Definitions 1.10, 1.11.

Remark 1.12
Note that extend is not restricted to the machine level but is more commonly found on the integration, optimization, and leading level in Figure 1.1.

Regarding intent, that is performance measurements as outlined in Table 1.1 or in ISO 22400 [14], we are going to use the term a key performance criterion throughout the lecture:

Definition 1.13 (Key performance criterion).
A *key performance criterion* is a function J, which measures defined information retrieved from the system against a standard.

The definition is abstract in nature and does not reflect the functionality of the system. In the event of an adaptive cruise control system (ACC), the vehicle's measurements are considered, including its speed. These are then compared to a standard speed of 50 km/h, which has been defined a priori. In the context of a heating system, a predefined temperature, for example for the melting of copper, can be established and subsequently compared to the temperature of the melting pot. The result is an offset.

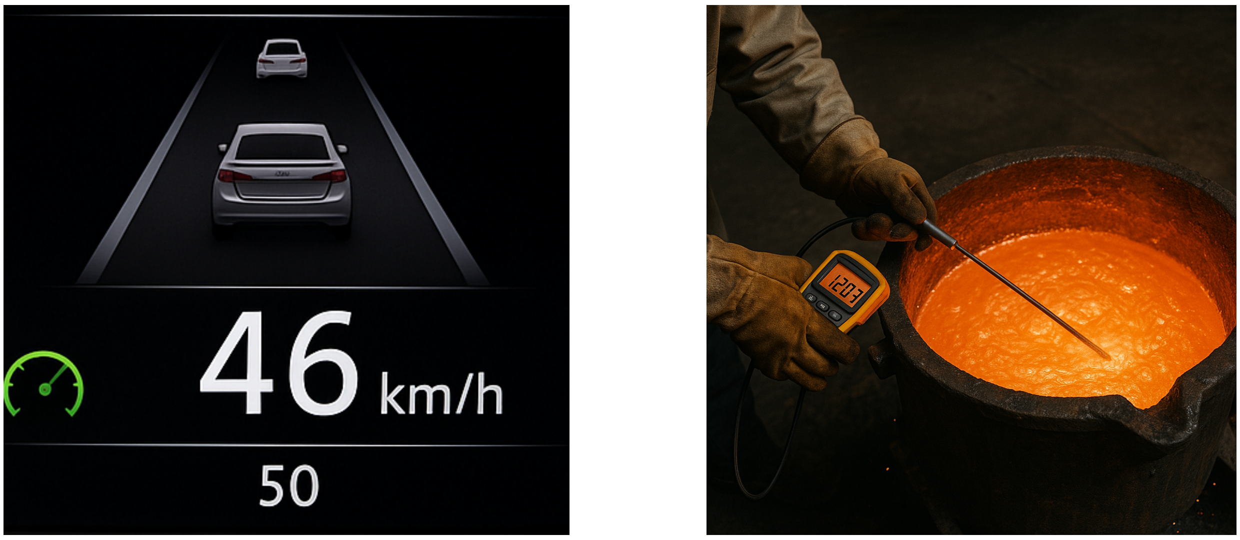


Figure 1.14.: ACC and melting pot illustrations — generated with chatgpt.com

In instances where multiple criteria are employed, they are consolidated into a single outcome. It is important to note the absence of any specification concerning the calculation of the output,

whether through the utilization of the Euclidean distance, the 1-norm, or the infinity-norm, as well as the manner in which the output should be processed.
Considering extension, we utilize the concept of constraints. The idea of constraints is to retrieve a set of conditions that must be upheld such that the system must be enabled to continue, i.e., the system must not come into a stage in which it terminates itself. These circumstances are typically modeled via sets:

Definition 1.14 (Constraints).
We call a subset $\mathbb{X} \subset \mathcal{X}$ *state constraint set* and $\mathbb{U} \subset \mathcal{U}$ *control constraint set*.

Examples of constraints in the case of the adaptive cruise control (ACC) include the vehicle's capacity to accelerate or decelerate. In the case of a car, constraints may also be road limitations. In the context of the melting pot, the extent to which the cooper can be heated and the rate at which it can be heated are both subject to limits.

Here, we like to note that each block within Figure 1.10 needs to provide features to ensure the safety and security of the overall system for both accidental and malicious mistakes. More precisely:

> Safety is defined as the freedom from unacceptable risk of physical injury.
>
> IEC 61508 (2010)

As such, safety does not mean that a system or process must be monitored and controlled so that no risks exist. Instead, the standard [10] defines that any safety-related system must either work correctly or fail in a predictable (safe) way. In other words, actions need to be taken such that the probability of a safety-related system satisfactorily performing the required safety functions under all the stated conditions within a stated period of time can be guaranteed.
Using the adaptive cruise control (ACC) system again, the latter does not preventing hazardous situations from occurring, the primary safety objective of the system is to design it such that any potential failure leads to a controlled and predictable outcome. In the event of a failure in the radar sensor or control logic, the system is required to ensure that it does not abruptly accelerate or decelerate. Instead, it is expected to undergo a graceful deactivation process, with the objective of transferring control back to the driver, ideally accompanied by a warning signal. It can thus be concluded that the system does not entirely eliminate all risk; however, it does guarantee that any residual risk is recognized, contained, and can be managed effectively.
Considering the melting pot example, the pot itself cannot prevent high temperatures or chemical reactions from occurring — risks are inherent in its operation — but it is engineered to contain

and direct those forces safely. In the event of a partial failure (e.g. a malfunctioning temperature sensor or a cooling issue), the design ensures that heat dissipation and containment prevent catastrophic outcomes. In a similar vein, a safety-related automation system is not expected to guarantee absolute absence of risk; rather, it is expected to ensure predictable, safe degradation of function when limits are exceeded.

Security, on the other hand, deals with information security.

> Information security risks relate to the loss of confidentiality, integrity and availability of data within the scope of the information safety management system.
>
> ISO/IEC 27032 (2022)

In a similar vein, it should be noted that security is not synonymous with the absence of threats. Conversely, the ISO/ICE 27032 [18] standard stipulates that an information security management system must ensure that risks to the confidentiality, integrity, and availability of information are identified, assessed, and treated in a systematic and proportionate manner. Consequently, security is not the absence of vulnerabilities; rather, it is the presence of controlled defences, detection mechanisms, and recovery measures that ensure risk remains within acceptable limits.
Using again the adaptive cruise control (ACC) system extended by vehicle-to-cloud services, the latter does not offer an absolute guarantee of protection against cyberattacks. Instead, the ISO/ICE 27032 [18] standard places a greater emphasis on the implementation of security controls, including encrypted communication, authentication protocols, and anomaly detection. These measures are designed to ensure that any attempt at a breach is detected, isolated, and mitigated. In the event of a compromise to a sensor data channel by an attacker, the system should be designed to revert to a degraded but secure state. This can be achieved by disabling automated distance control and notifying the driver, rather than acting on potentially falsified data.
With regard to the melting pot, the process cannot be made immune to heat loss, contamination, or intrusion. However, safeguarding is achieved through layered barriers, such as temperature monitoring, controlled access, and containment. A breach of one protective layer should not lead directly to a meltdown; instead, it triggers alarms and fallback measures. Conversely, the field of information security emphasizes defense in depth and controlled response, with the objective of ensuring that incidents are predictable, contained, and recoverable, as opposed to being completely preventable.

The overarching objectives of the fourth industrial revolution centre on the realisation of smart production methods. This refers to the integration of information and communication technologies in technical systems, particularly manufacturing systems, through the development of Cyber-Physical Systems (CPS). The overarching objective of Industry 4.0 is to facilitate the development of digitally advanced, customised, and eco-friendly manufacturing facilities. This transformation is designed to facilitate flexible production and interconnect all stages of the manufacturing process. In this particular context, related terms include the industrial internet of things (IIoT) and edge-cloud-control (ECC). These technological innovations have the potential to impact entire processes, thereby posing challenges in the areas of organisation, strategy and engineering. As Giangi et al. [8] demonstrate, these principles are essential for the successful integration of novel and disruptive technologies. The lecture discusses leadership aspects that are necessary for ensuring efficient operation of new or modified systems for the customer. To master these aspects, certain skills are required to an automation engineer:

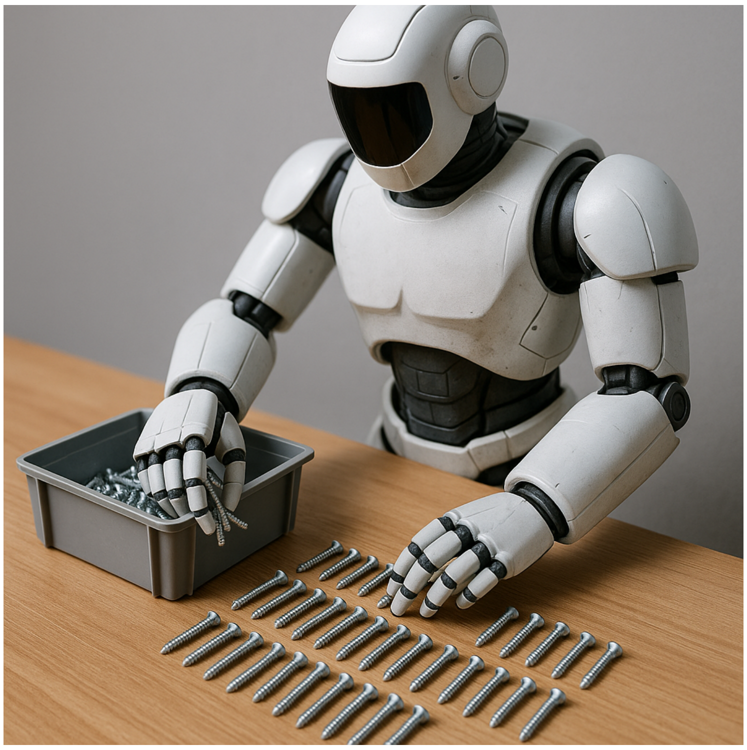
- Leadership: In projects, an automation engineer has to make technical decisions to make sure that systems work well together in the mechanical, electrical and digital areas so that the client's needs are met.
- Innovation: To improve automated processes and increase productivity, it is important to stay connected to engineering networks, to watch the development of important technologies, and to look for more information to improve the quality of plans.
- Programming: Since a human may not be there to control the systems manually, coding of software solutions or programming physical machines like robotic assembly lines is a necessary skill of automation engineers.
- Mechanical knowledge: Understanding mechanical systems enables automation engineers to align logic with physical constraints such as kinematics, tolerances, and wear.
- Communication: Automation engineers must understand and apply communication models to ensure reliable, secure, and interoperable data exchange across all network layers.
- Flexibility: Rapidly changing technologies, standards, and project conditions require adaptability to new tools, domains, and problem-solving approaches.

In the following chapters, we will introduce and discuss the fundamentals of the standard automation structure from Figure 1.10 along the automation path illustrated in Figure 1.1.

Part I.

Planning and specification

CHAPTER 2 SYSTEM PLANNING


Generated with chatgpt.com

The first rule of any technology used in a business is that automation applied to an inefficient operation will magnify the inefficiency.

Bill Gates

In this chapter, we will look at the first layer of automation, the planning layer. At this initial level, everything starts with an idea of what shall be realized regarding automation. In the past, the typical mistake at that stage was to think of a replacement instead of renewal (or re-thinking). Most automation cases involve digitization in terms of process information. This means gathering information about a running process. So we've got operational data and flow charts, which we need for automation. But this isn't enough on its own. Coming back to the overall path of automation, Figure 2.1 displays the content of the present chapter.

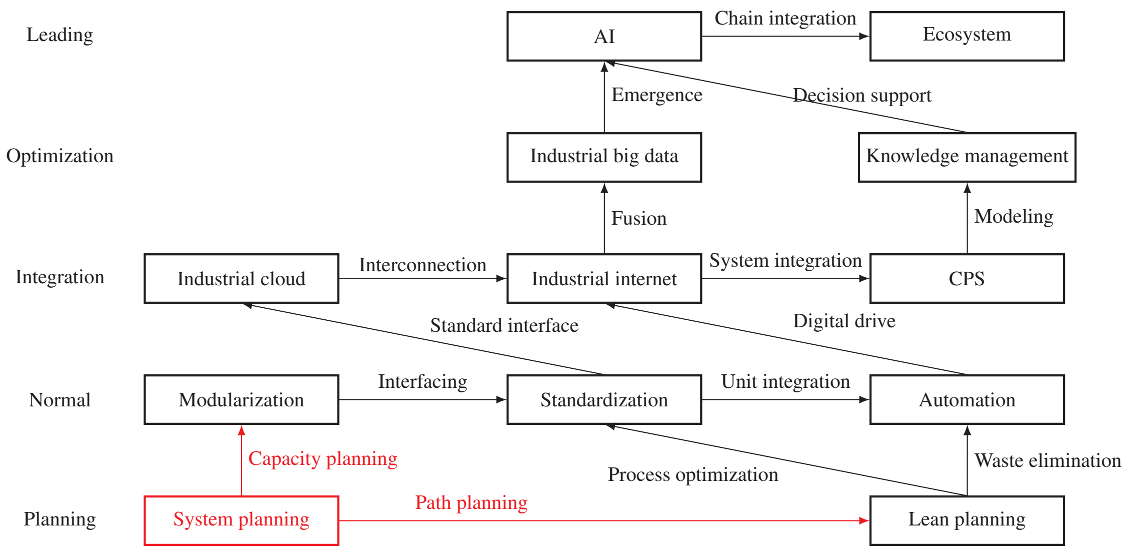


Figure 2.1.: Integrated implementation path of automation

Here, we will discuss the perspectives and interests placed upon planning. To do this, we combine systems and processes into a network, which can be analyzed. The importance of these properties depends on the peer group assessing the network. Here, we first discuss these groups. After that, we focus on modeling concepts and respective description methods. We will use the latter to derive digital models, which we will extend to a digital shadow and digital twin in Chapter 6.

2.1. Interested parties, perspectives and requirements

Automation follows the strategies of cost efficiency or differentiation, cf. Figure 1.2. And in fact, any of the following ideas are not sufficient and, in some cases, have even shown catastrophic results:

- Automation is not a purpose. Automation is a means to achieve or improve on KPIs, yet it shouldn't be implemented blindly, e.g., because others introduce it.

- Automation is not an ideology. It should be carefully checked whether the system or process at hand can be improved or if automating a system/process improves connected systems/processes. One solution here is to consider connected systems/processes.
- Automation is not a whitewashing instrument. If KPIs are chosen or designed to improve a system or process which is economically, ecologically or socially unsuitable, it will cause degradation and magnify inefficiency at the cost of marketing. In such a case, constraints are much more suitable than KPIs for automation.

Having discussed what is insufficient, we consider necessary input for planning next. Here, we must distinguish between interested parties and their perspectives. In the common literature, an interested party is most commonly referred to as a stakeholder, yet formally the norm [15] defines the following:

> An interested party can be a stakeholder, person, or organization that can affect, can be affected by, or perceive itself to be affected by a decision or activity.
>
> ISO 9001 (2015)

Hence, this gives us limits to what needs to be included in modeling a system or process, cf. Definition 1.4. Examples of interested parties can include (but are not limited to):
ISO 9001 [15] states that one must consider those interested parties, which are relevant for the system/process. For these parties, requirements need to be identified and monitored or reviewed, which are given in [17].

> A requirement is defined as an expression, in the content of a document, that conveys objectively verifiable criteria to be fulfilled and from which no deviation is permitted if conformance with the document is to be claimed.
>
> ISO/IEC Directives 2 (2021)

So, the requirements give us a design space that we must guarantee. In a model, this will be given by constraints, as explained in Definition 1.14. If the design space is empty, this means that the requirements exclude each other. Otherwise, we have a certain amount of freedom for our task. This is the part where we need to consider the different views of the interested parties. This is where we can make things better, make choices or find ways to improve in terms of a KPI

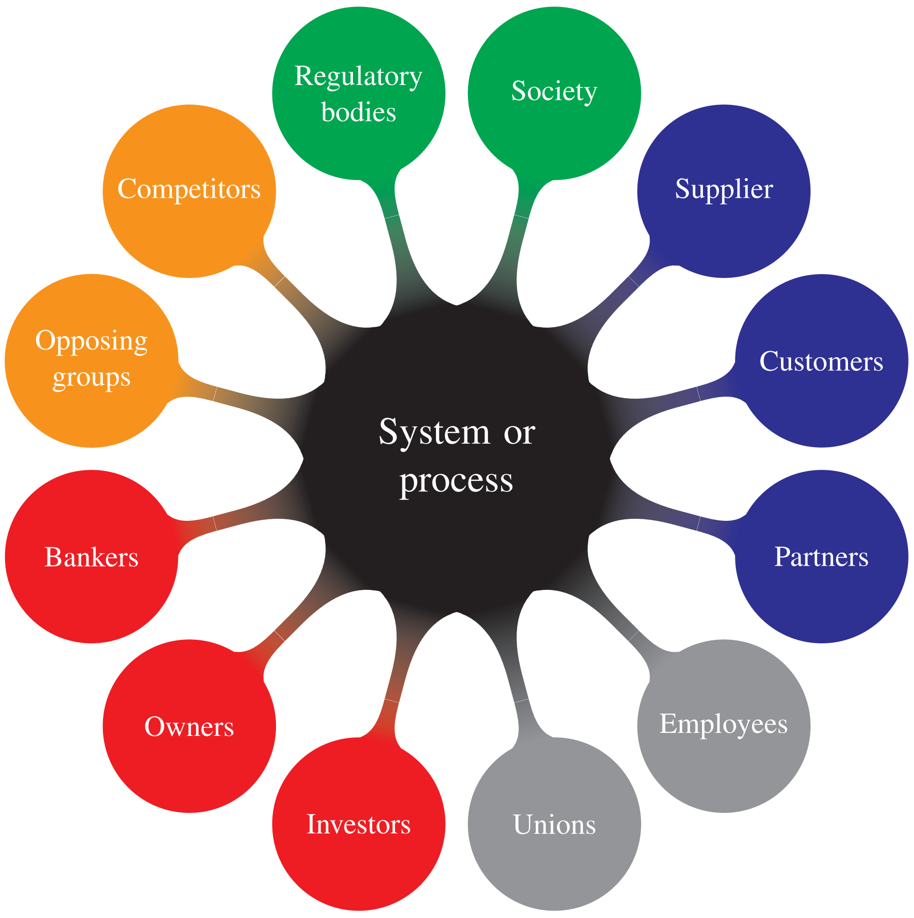


Figure 2.2.: Interested parties

(Definition 1.13). Here, the norms falls short on a proper definition of what is usually termed *needs and expectations*. Need is used to mean requirement, while expectation is used to mean perspective. Here, we apply the following:

> An expectation is a strong believe that something will happen or be the case.

The following Table 2.1 provides an overview on the most common expectations.

Table 2.1.: Relevant perspectives and features in automation

Expectation	Feature
Performance and quality	Performance, time/space sampling
Input	Operational area, industrial sector, context, qualifications
Reliability (RAMS)	Dependability, accessibility, maintainability, safety
Physics	Composition, dimensions, interfaces, implementation, materials
Ecology	EM, climate, energy, geometry, stress
Legal	Norms, regulations, conventions, laws, admission
Economic	Life cycle cost, initial/running cost, disposal cost, return on invest

Due to these diverse perspectives within the automation domain, the respective modelers tend to concentrate on the needs, terminology, and objectives of their own peer group. As a result, they often develop models that reflect discipline-specific assumptions, abstractions, and constraints, which quite likely leads to diverging system representations, inconsistent data structures, and limited model interoperability across the automation life cycle.

2.2. Concepts of modeling

As previously discussed, multiple factors can lead to divergence among models—ranging from sector-specific modeling objects and domain-dependent programming languages to differences in organizational culture and cognitive framing. To better understand these sources of variation, we take a step back and revisit the fundamental concept of modeling itself—its underlying purpose being to create a coherent and holistic representation of a system or process.

Modeling, as a cognitive method, comprises several interrelated components, as illustrated in Figure 2.3. The model can exist on various levels and degrees of detail, depending on its intended purpose and the physical phenomena it aims to represent. For instance, the robot arm illustrated in Figure 2.4 can be abstracted to its main physical components—such as links, joints, and actuators—forming a kinematic chain that describes the system's geometry and motion constraints. On a deeper physical level, the same system can be modeled dynamically, capturing forces, torques, masses, friction, and inertia to simulate its motion behavior according to Newtonian or Lagrangian mechanics. Even further, electromagnetic effects in the actuators or material

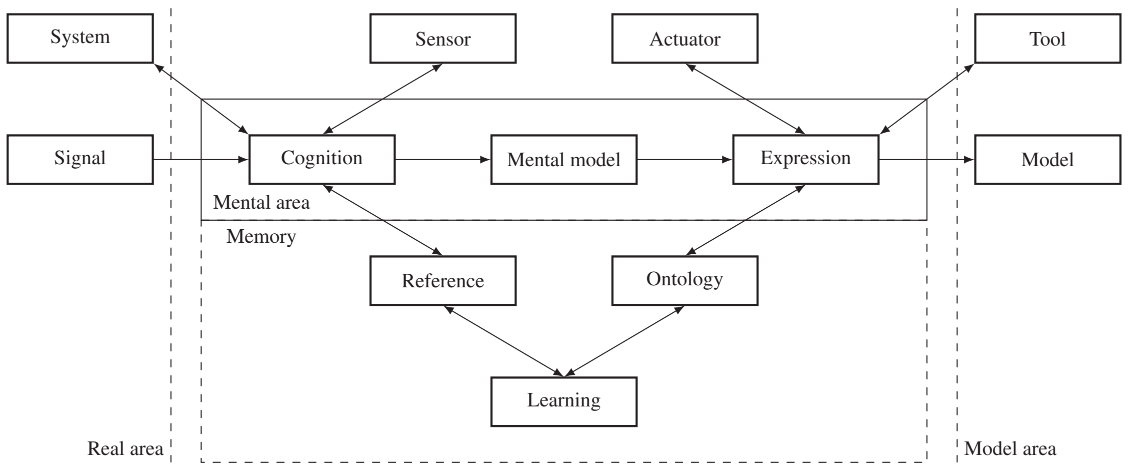
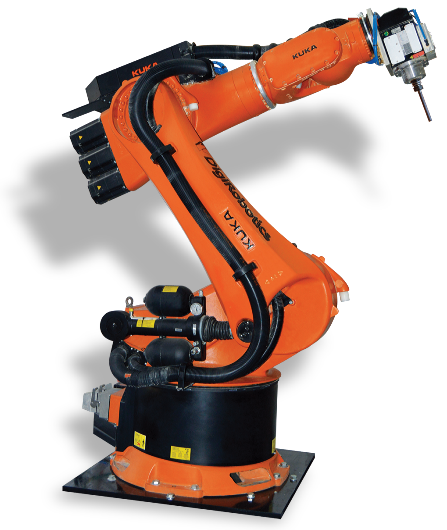


Figure 2.3.: Modeling as cognition method

elasticity under load can be incorporated to reflect energy transformations and structural deformations.



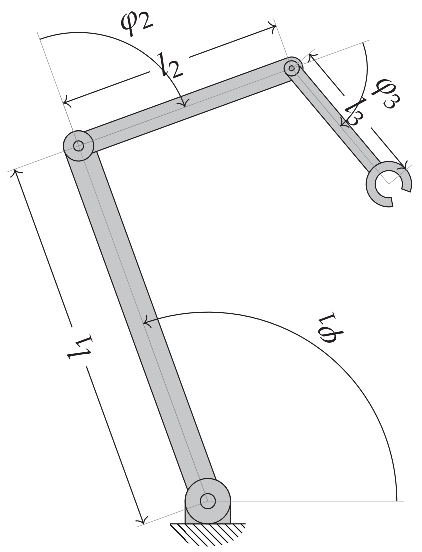


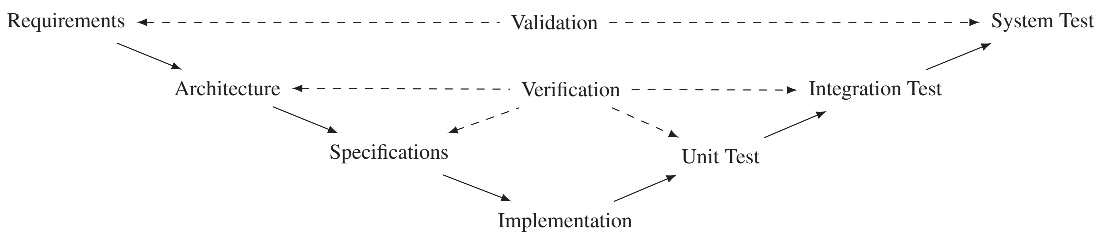
Figure 2.4.: Robot arm in real and model area

Such layered modeling—from abstract kinematics to detailed physical representation—enables engineers to balance computational complexity with model fidelity, depending on whether the focus lies on control design, structural analysis, or system-level optimization.

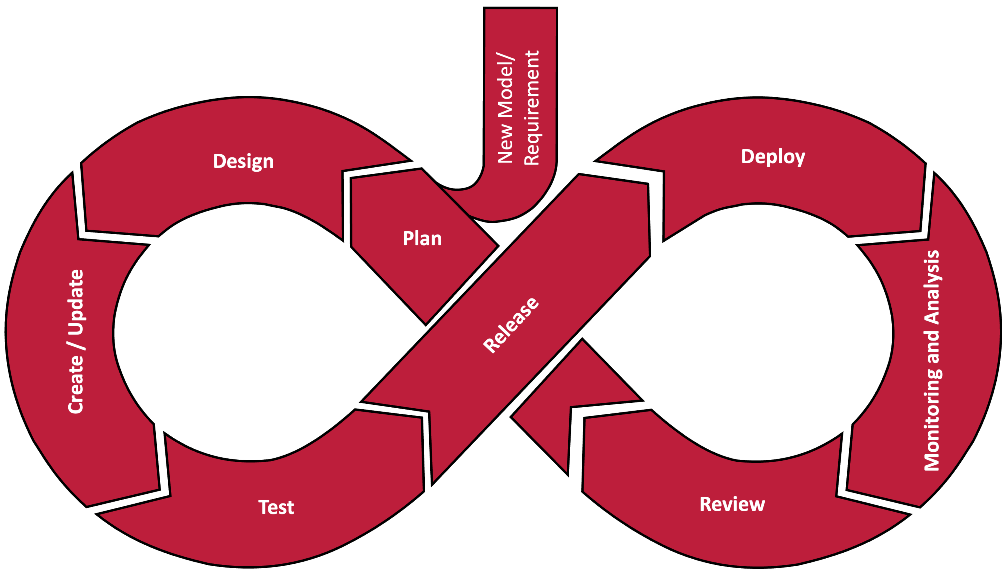
However, all of these layers can be described via, e.g., a set of differential equations

$$\begin{aligned}\dot{\mathbf{x}}(t) &= f(\mathbf{x}(t), \mathbf{u}(t), t), \quad \mathbf{x}(t_0) = \mathbf{x}_0 \\ \mathbf{y}(t) &= h(\mathbf{x}(t), \mathbf{u}(t), t),\end{aligned}$$

which need to be parameterized and identified. The layer or respectively the level of detail should depend on the purpose of the model and how it shall be used.
In contemporary engineering practice, two predominant planning paradigms can be distinguished. The first follows the classical V-model, whereas the second adopts an agile development approach. Both paradigms incorporate modeling activities within their respective process phases (see Fig. 2.5). Although these approaches differ fundamentally in their structure and progression, they share comparable requirements and expectations concerning the use of models.



(a) Engineering V-model



(b) Agile planning in DevOps loop

Figure 2.5.: Planning approaches in engineering

> **Link:** For further details on planning approaches, we refer to the lectures on *Project management*.

Here, we start of with the process requirements, that is working principles within any modeling process. These requirements are a state-of-the-art list, which is commonly used as a convention and not as a definition.

Convention 2.1 (Process requirements of modeling)
During the modeling process, six principles need to be met:

1. Principle of Correctness: A model needs to present the facts correctly regarding structure and dynamics (semantics). Specific notation rules have to be considered (syntax).
2. Principle of Relevance: All relevant items have to be modeled. Non-relevant items have to be left out, i.e. the value of the model doesn't decline if these items are removed.
3. Principle of Cost vs. Benefit: The amount of effort to gather the data and produce the model must be balanced against the expected benefit.
4. Principle of Clarity: The model must be understandable and usable. The required knowledge for understanding the model should be as low as possible.
5. Principle of Comparability: A common approach to modeling ensures future comparability of different models that have been created independently from each other.
6. Principle of Systematic Structure: Models produced in different views should be capable of integration. Interfaces need to be designed to ensure interoperability.

Similarly, any concept of modeling should satisfy the following functional requirements:

Convention 2.2 (Functional requirements of modeling)
For any modeling concept, the alignment of methods, tools, and implementation is essential to ensure consistency and applicability across development stages.

Consider, for instance, the modeling of a robotic arm in an automated production line. The chosen method must support modern software development practices, analytical soundness, and the ability to represent, compose, and simulate systems — capturing both deterministic motion control and stochastic disturbances such as sensor noise — while maintaining vertical and horizontal consistency across abstraction levels. The supporting tools, such as modeling environments

or simulation platforms, must offer portability, compatibility, and usability to facilitate efficient integration with existing design and testing workflows. Finally, the implementation must prove viable within software and hardware constraints, ensuring that models can be deployed, validated, supported and even reverse-engineered during later system updates.
Based on the latter, we now outline different description methods to approach the goal of a digital model.

2.3. Description methods

As outlined in the previous section, it is essential to minimize model divergence, particularly on the information layer, to ensure a consistent and integrated system understanding. Depending on the intended purpose and level of abstraction, a wide range of modeling methods may be employed—spanning conceptual representations, behavioral descriptions, and physical simulations. Table 2.2 organizes these methods according to their primary area of application and degree of detail. The arrangement follows a top-down logic, reflecting the typical engineering sequence when deriving and implementing automation for a given process: starting from high-level functional descriptions, progressing through control and logic modeling, and eventually reaching executable and hardware-related specifications. In an ideal implementation, a structured utilization of these methods supports traceability across levels and facilitates the alignment between conceptual design and technical realization.

Table 2.2.: List of description methods

Class	Example
Abstraction oriented	Verbal description, algebra, proposition logic, predicate logic
Structure oriented	Sequential logic system, combinatorial logic
Implementation oriented	Logic plan, function plan, contact diagram, structure diagram, timing diagram, instruction list, gantt chart
State oriented	Decision table, transition table, state diagram, state graph, Karnaugh-Veitch diagram
Technology oriented	Flow chart, switching plan, computer aided design (CAD)

Continued on next page

Table 2.2 – continued from previous page

Class	Example
Method oriented	Network diagram, Nassi-Schneidermann diagram, unified modeling language (UML), structure-analysis-real-time (SA/RT) diagram
Mathematical	Boolean algebra, differential/difference equations, Markov chains

It should be emphasized that the described methods cannot be regarded as self-contained; they require a top-down specification to ensure conceptual coherence and a bottom-up connection to maintain consistency with implementation details. One of the fundamental elements common to all categories in Table 2.2 are networks, objects, and ontologies, which serve as structural backbones for integrating diverse modeling perspectives. Within this lecture, we focus on networks and objects as the primary modeling paradigms. These approaches are well aligned with modern computer science principles such as object orientation, while simultaneously supporting mathematical representations like discrete-event or event-triggered dynamical systems.
To illustrate, consider the example of an industrial robot integrated into an automated production cell as illustrated in Figure 2.6. From a top-down perspective, the robot is defined as an object with specific properties (e.g., kinematic configuration, payload, and control interfaces) and behavioral models that describe its interaction with the production process. Bottom-up, sensor networks and actuator connections provide concrete data that link the abstract model to the physical system. In this way, object- and network-based modeling enable both a compact and hierarchical system description, as well as horizontal integration across disciplines—ranging from mechanical design to software control—thereby ensuring that the model remains consistent across all abstraction levels and descriptions (cf. Table 2.2 and Figure 1.11).
More formally, we define the following.

Definition 2.3 (Object).
A set of *objects* $\mathcal{O} = \mathcal{A} \cup \mathcal{M}$ consists of definable parts called attributes $\mathcal{A}$, and reasoning parts called methods $\mathcal{M}$.

Reconsidering the robot in the above setting of Figure 2.6, we could define the three attributes robot arm, conveyor and package, and a possible method could be putting a package on the conveyor.

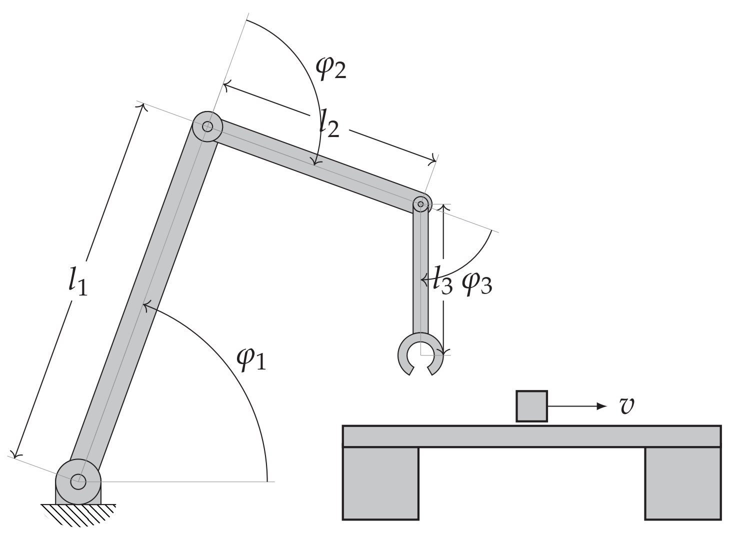


Figure 2.6.: Sketch of a robot arm

Remark 2.4
Note that Definition 2.3 does not require all parts or methods to be defined. This is in accordance with our concept of a model, for which specific parts are modeled and the remainder is considered as a disturbance.

Next, we will use the introduced objects to derive a coherent and linked system description by applying the so-called *Petri Network* method, which allows us to formally represent dynamic interactions, concurrency, and event-driven behavior within the modeled automation process.

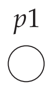
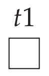
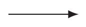
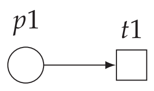
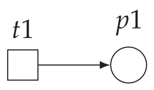
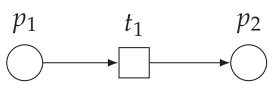
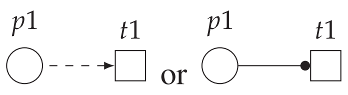
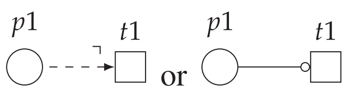
2.4. Petri Networks

In the general context of automation, an object may range from a parameter in a program to the program itself, the machine controlled by that program, a screw within the machine, or up to the entire supply chain of a company and its surrounding economy. These objects may be connected, which leads us to the definition of a network.

Definition 2.5 (Network).
Let $\mathcal{O} = \mathcal{A} \cup \mathcal{M}$ be a set of objects and $\mathcal{E} \subset \mathcal{A} \times \mathcal{M} \cup \mathcal{M} \times \mathcal{A}$ be a set of pairs of objects. Then we call $\mathcal{E}$ the set of edges and the tuple $\mathcal{N} = (\mathcal{O}, \mathcal{E})$ a *network*.

One such network, which can be used as a description method, is the so-called Petri-Net. Within a Petri-Net, attributes are represented by circles, methods by blocks, and edges by directed arcs. Moreover, the denomination extended in Table 2.3 is used.

Table 2.3.: List of Petri-Net symbols

Symbol	Meaning
	Attribute/ place
	Method/ transition
	Edge
	Pre-edge
	Post-edge
	Relation
	Communication/test edge
	Inhibition edge

In this context, a pre-edge is a requirement for a method, while a post-edge is the result of a method. In Table 2.3, we drew both pre-edge and post-edge as a 1-1 connection, yet 1-n connections are possible as well. A simple Petri-Net is illustrated in Figure 2.7.

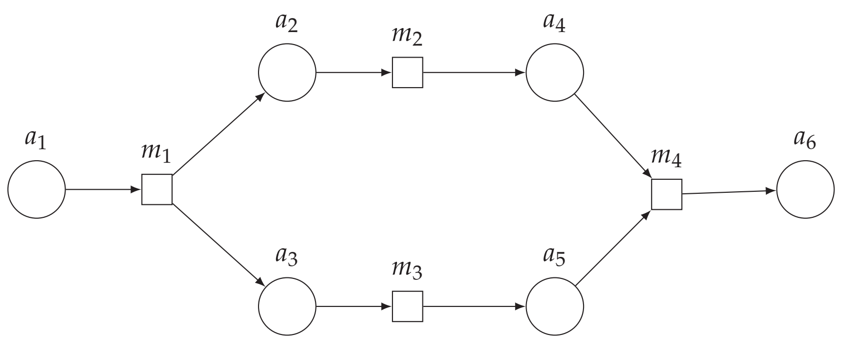


Figure 2.7.: Simple Petri-Net

Petri-Nets are ideal to model parallelism, resource allocation, state-based and event-driven systems. However, as every model is an abstraction of specific system property, Petri-Nets lack of the ability to model properties, such as timing dependencies. This leads to a freedom in implementation but also to erroneous assumptions. To surmount such limitations, extensions such as colored or timed Petri-Nets were proposed. Nonetheless, the complexity can easily increase to a confusing and unclear level.

Remark 2.6
In the computer science or mathematics literature, a network is also called a graph, for which the set of objects is typically referred to as a set of nodes. Also different connections between objects are considered, i.e. aggregation (one object consists of ...) or abstraction (one object is a ...).
In the process automation literature, attributes are also called places indicating the physical position of an object. Methods, on the other hand, are also called transitions, i.e. transportation from start to destination or modifications from initial to the target property.

The edge structure itself must also be stored and may be used for certain computations. To cover both aspects, the network or graph can be summarized in the so called incidence matrix.

Definition 2.7 (Incidence matrix).
For any network $\mathcal{N} = (\mathcal{O}, \mathcal{E})$, we call

$$\mathcal{I} = \left[\mathcal{I}_{jk}\right] \qquad \text{where } \mathcal{I}_{jk} := \begin{cases} 1 & \text{if } \exists \text{ edge connecting } m_j \in \mathcal{M} \text{ and } a_k \in \mathcal{A} \\ -1 & \text{if } \exists \text{ edge connecting } a_j \in \mathcal{A} \text{ and } m_k \in \mathcal{M} \\ 0 & \text{else} \end{cases} \tag{2.2}$$

incidence matrix of the network.

Given the network from Figure 2.7, we obtain the incidence matrix

$$\mathcal{I} = \left[\begin{array}{ccccccccccc} & a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & m_1 & m_2 & m_3 & m_4 \\ a_1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ a_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ a_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ a_4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ a_5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ a_6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ m_1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ m_2 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ m_3 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ m_4 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{array}\right]$$

Remark 2.8
The incidence matrix can be written in a more condensed form if we utilize the methods as columns and attributes as rows only.

The incidence matrix provides a compact and formal representation of a network, capturing the relationships between places (states) and transitions (events). Beyond its descriptive role, it serves as a computational tool for analyzing system behavior. In particular for assessing risks, we can utilize the matrix by applying it repeatedly to a given initial condition, i.e. a vector describing the initial condition of the system. This allows us to check whether certain attributes are necessary/-sufficient or may ever be reached.

Remark 2.9
In the context of an FMEA (failure mode and effects analysis), assessing if and with what probability a risk case may occur is a common question. For such questions, a Petri-Net model can provide a direct answer.

Considering our simple Petri-Net from Figure 2.7 as a flow of products between workstations as methods and storage area as attributes, the split at method m_1 may result in 1 unit moving to attribute a_2 while 2 units are shifted to attribute a_3. If the joint at method m_4 uses only 1 unit from attributes a_4 and a_5, then a stockpile will be rising up unboundedly at attribute a_5.
In most cases, we are interested in parts of a system/process only. To cut these free from the rest of the system/process, we apply the concept of so called *configurations*.

Definition 2.10 (Configuration).
Consider a network $\mathcal{N} = (\mathcal{O}, \mathcal{E})$ with $\mathcal{O} = \mathcal{A} \cup \mathcal{M}$. Then any subset $\mathcal{C} \subseteq \mathcal{A}$ is called a *configuration*. We call the tuple $(\mathcal{N}, \mathcal{C})$ an *elementary network*.

Hence, a configuration is a subnet within a network, that is a use case which we are interested in. Note that since the configuration is a subnet, it may interact with the rest of the network. For a configuration, however, we are only interested in answers for this specific subnet.

Remark 2.11
Loosely speaking, if the entire world were represented as a network, then a configuration is a model of a process/system which interacts with its surroundings and is disturbed by it.

Before we can properly formulate the questions introduced above, it is necessary to introduce a few additional technical details. In a Petri-Net, the notion of a configuration is extended by the concepts of markings and multiplicities (cf. Fig. 2.8). A marking represents the current distribution of tokens within the network and can be interpreted as the number of available units associated with a specific attribute — for instance, the number of motors waiting to be installed into cars. Multiplicities, on the other hand, define how many such units are required for a given method, such as the number of gears needed to assemble a complete gearbox. Together, markings and multiplicities allow us to quantify system states and resource requirements, forming the foundation for analytical reasoning within the Petri-Net framework.

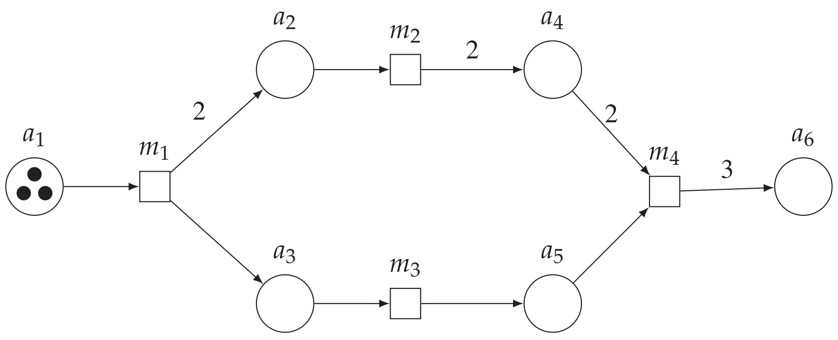


Figure 2.8.: Simple Petri-Net with markings and multiplicities

More formally, we define the following:

Definition 2.12 (Petri-Net).
Consider a network $\mathcal{N} = (\mathcal{O}, \mathcal{E})$ with $\mathcal{O} = \mathcal{A} \cup \mathcal{M}$. Moreover, let

$$\mathcal{C}_{\mathcal{A}} : \mathcal{A} \to \mathbb{N}_0^{\mathcal{A}}, \qquad \mathcal{C}_{\mathcal{A}}(a) = \#(a) \quad \forall a \in \mathcal{A} \tag{2.3}$$

$$\mathcal{C}_{\mathcal{E}} : \mathcal{E} \to \mathbb{N}_0^{\mathcal{E}}, \qquad \mathcal{C}_{\mathcal{E}}(e) = \#(e) \quad \forall e \in \mathcal{E} \tag{2.4}$$

be multisets. Then the triple $(\mathcal{N}, \mathcal{C}_{\mathcal{A}}, \mathcal{C}_{\mathcal{E}})$ is called a *Petri-Net*. The maps $\mathcal{C}_{\mathcal{A}}$ and $\mathcal{C}_{\mathcal{E}}$ are called *marking* and *multiplicity*.

Hence, markings can be considered as storage, whereas multiplicities represent performance requirements for methods/tasks. Here, we already obtain the first interesting questions:

- reachability: Can all markings be set? Which markings can be set?
- coverability: Can specific markings be set?

Continuing, we are interested in how the system/process evolves over time and what are the requirements for such a flow. This leads us to the so-called flow relation.

Definition 2.13 (Flow relation).
Given a Petri-Net $(\mathcal{N}, \mathcal{C}_{\mathcal{A}}, \mathcal{C}_{\mathcal{E}})$. Then we call the set

$$\mathcal{F} := \{e \in \mathcal{E} \mid \mathcal{C}_{\mathcal{E}}(e) > 0\} \tag{2.5}$$

a *flow relation*.

The flow relation says that certain values of attributes are required in order for edges to be executable. This points in the direction of the questions

- liveness: Is a process/system deadlock-free, can all attributes be marked and unmarked?
- consistency: Will all markings be set uniquely?
- boundedness: Will all markings stay bounded?

Regarding these questions, special attention must be paid to inhibitor edges. In this context, a flow is inhibited if the inhibited attribute is marked. This directly leads us to the requirements and results of the methods.

Definition 2.14 (Preset and postset).
Consider a Petri-Net $(\mathcal{N}, \mathcal{C}_{\mathcal{A}}, \mathcal{C}_{\mathcal{E}})$ with flow relation $\mathcal{F}$. For any edge $e \in \mathcal{F}$, we call the set

$$\bullet m := \{a \in \mathcal{A} \mid \mathcal{C}_{\mathcal{E}}(a, m) > 0\} \tag{2.6}$$

preset of a method $m \in \mathcal{M}$ and the set

$$m\bullet := \{a \in \mathcal{A} \mid \mathcal{C}_{\mathcal{E}}(m, a) > 0\} \tag{2.7}$$

postset of a method $m \in \mathcal{M}$.

Preset and postset can be interpreted as input and output of a method, cf. Figure 1.9 where the system corresponds to a method block in the Petri-Net.

Remark 2.15
As stated before, inhibitor edges are somewhat different. These edges enter the set definition (2.6) *with a negation, which can be simplified to $\mathcal{C}_{\mathcal{E}}(a, m) = 0$ if $e = (a, m)$ is an inhibitor edge.*

Following the input/output idea, a method can only be executed if all requirements are satisfied. This allows us to check whether or not a method will ever by applied (e.g., in an FMEA). Apart from that, the presets identify the required chain of methods/attributes, which are elementary for the application of the method under consideration. In the context of Petri-Nets, the application is typically referred to as firing.

Definition 2.16 (Enabling and firing).
Given a Petri-Net $(\mathcal{N}, \mathcal{C}_\mathcal{A}, \mathcal{C}_\mathcal{E})$. If

$$\forall a \in \bullet m : \ \mathcal{C}_\mathcal{A}(a) \geq \mathcal{C}_\mathcal{E}(a, m) \tag{2.8}$$

holds, then a method $m \in \mathcal{M}$ is *enabled*.
If a method $m \in \mathcal{M}$ is enabled, then firing of m is possible i and leads to updating the markings according to

$$\mathcal{C}_\mathcal{A}(a) = \begin{cases} \mathcal{C}_\mathcal{A}(a) - \mathcal{C}_\mathcal{E}(a, m), & \forall a \in \bullet m \\ \mathcal{C}_\mathcal{A}(a) + \mathcal{C}_\mathcal{E}(m, a), & \forall a \in m \bullet \\ \mathcal{C}_\mathcal{A}(a), & \text{else.} \end{cases} \tag{2.9}$$

Remark 2.17
Sometimes a method is called enabled if additionally to (2.8) *the postset is clear, i.e.*

$$\forall a \in m \bullet : \ \mathcal{C}_\mathcal{A}(a) = 0.$$

What makes Petri-Nets particularly interesting is that we can check what will happen when methods may continually fire in arbitrary order. Note that we never mentioned anything about time, sequence, order, etc. Hence, the stated questions can be analyzed without knowledge of time. This holds particularly true for paths, circuits, and deadlocks.

Definition 2.18 (Path and circuit).
Consider a Petri-Net $(\mathcal{N}, \mathcal{C}_\mathcal{A}, \mathcal{C}_\mathcal{E})$ with flow relation $\mathcal{F}$.

- we call a sequence $a_1, \ldots, a_n \in \mathcal{A}$ a *path* if the sequence is finite, nonempty and respective edges are elements of the flow relation $\mathcal{F}$.
- A path is called a *circuit* if $a_1 = a_n$ and additionally from $a_j = a_k$ we have $j = k$ for all $1 < j, k < n$.

Paths and circuits can be obtained directly by the incidence matrix for Petri-Nets. Here, it is common to separate between the preset and postset incidence matrix. The preset one allows seeing circuits, which may get lost if the complete incidence matrix is computed.

Definition 2.19 (Petri-Net incidence matrix).
For any network Petri-Net $(\mathcal{N}, \mathcal{C}_{\mathcal{A}}, \mathcal{C}_{\mathcal{E}})$, we call $\mathcal{I} = \mathcal{I}^- + \mathcal{I}^+$ with

$$\mathcal{I}^- = [\mathcal{I}_{jk}] \qquad \text{where } \mathcal{I}_{jk} := \begin{cases} -\mathcal{C}_{\mathcal{E}}(a_j, m_k) & a_j \in \bullet m_k \\ 0 & \text{else} \end{cases} \tag{2.10}$$

$$\mathcal{I}^+ = [\mathcal{I}_{jk}] \qquad \text{where } \mathcal{I}_{jk} := \begin{cases} \mathcal{C}_{\mathcal{E}}(m_k, a_j) & a_j \in m_k \bullet \\ 0 & \text{else} \end{cases} \tag{2.11}$$

incidence matrix of the Petri-Net.

The incidence matrix is a central tool for computing the properties of networks and initial markings. To qualify such a network, we adapt the definition of a key performance indicator (Definition 1.13) to the network setting.

Definition 2.20 (Cost function for networks).
Consider a network $(\mathcal{O}, \mathcal{E})$. Then we call $J : \mathcal{I} \times \mathcal{I}^2 \to \mathbb{R}_0^+$ *cost function*.

While Definition 2.20 is generic, the following indicators specify the idea of qualifying a network in practical terms.

2.4.1. Reachability and coverability

Regarding our question on uniqueness, it is already evident that the resulting marking in a Petri-Net depends on the specific sequence of transition firings; consequently, uniqueness of the resulting state cannot generally be expected. Different firing sequences may lead to distinct but valid markings, even if they start from the same initial configuration. This non-uniqueness reflects the inherent concurrency and nondeterminism within distributed systems modeled by Petri-Nets.
Nevertheless, the concept of markings allows us to systematically investigate two fundamental properties: reachability and coverability. Reachability describes whether a particular marking can be obtained from the initial marking by some sequence of method firings. Coverability, in contrast, asks whether a marking can be reached that equals or exceeds another marking component-wise — i.e., whether a desired or critical system state can ever occur. For example, in a manufacturing Petri-Net, one might ask whether a state in which multiple assembly stations

are simultaneously occupied is reachable, or whether a situation representing resource overload (e.g., too many motors waiting in a buffer) is coverable.
To check for a specific marking (i.e. conditional state of a production system) with regards to reachability and coverability, we can apply the following algorithm:

Algorithm 2.21 (Coverability and reachability)
Consider a Petri-Net $(\mathcal{N}, \mathcal{C}_{\mathcal{A}}, \mathcal{C}_{\mathcal{E}})$ together with an initial marking $c_0 \in \mathbb{N}_0^{\mathcal{A}}$ and a terminal marking $c \in \mathbb{N}_0^{\mathcal{A}}$ to be given.

```
1. For each $a \in \mathcal{A}$ with $c_0(a) > 0$
        Set $\mathcal{R}(c_0) := \mathcal{R}(c_0) \cup \{a\}$.
2. While there exists a fireable method $m$ do
    a) For each method $m \in \mathcal{M}$
        i. If $m$ is fireable as defined in (2.8)
            A. Update marking $\mathcal{C}_{\mathcal{A}}$ according to (2.9)
            B. Set $\mathcal{R}(c_0) := \mathcal{R}(c_0) \cup \{a\}$ for all $a \in m\bullet$
    b) If $\mathcal{C}_{\mathcal{A}} \geq c$, then $c$ is coverable and stop.
3. If $\mathcal{C}_{\mathcal{A}} \not\geq c$, then $c$ is not coverable.
```

Within Algorithm 2.21, the set $\mathcal{R}(c_0)$ denotes all markings that can be reached from the initial marking c_0 through any valid sequence of method firings. In the context of the production Petri-Net introduced earlier, this set represents all possible configurations of the manufacturing process that can arise from the initial availability of components — for instance, different combinations of motors, gears, and partially assembled powertrains. Each element of $\mathcal{R}(c_0)$ therefore corresponds to a feasible production state, such as „motors installed but gearbox incomplete“ or „assembly finished and awaiting quality check“. Determining this reachable set allows us to analyze whether critical states — e.g., resource shortages, buffer overflows, or deadlocks — can occur during operation, thereby supporting the verification of process robustness and resource allocation strategies.
If we consider the production system given in Figure 2.9, we observe that for the given tokens c_0 we can produce either one assembled gearbox or one assembled motorbox.
Hence, if we want to check

- if $c = \{a_4\}$, then c is covered,
- if $c = \{a_5\}$, then c is covered,

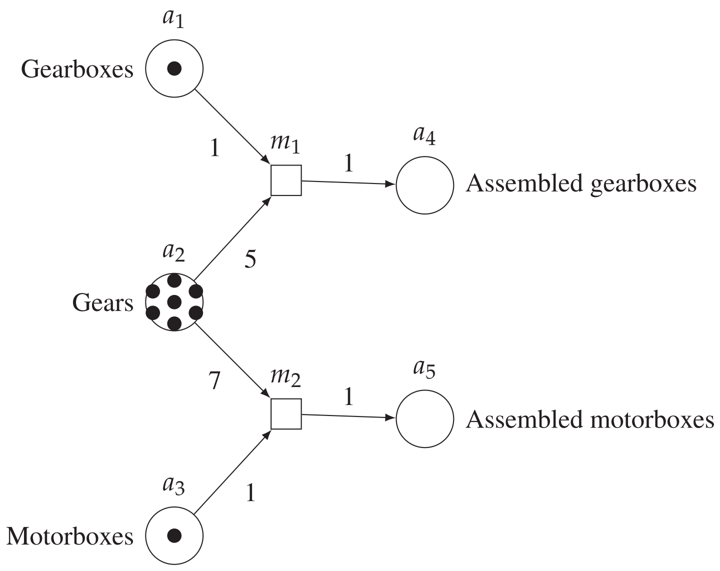


Figure 2.9.: Petri-Net of gearbox and motorbox assembly

- if $c = \{a_4, a_5\}$, then c is not covered,
- but $\mathcal{R}(c_0) = \{a_1, a_2, a_3, a_4, a_5\}$.

From this, we obtain that $\mathcal{R}(c_0)$ is something different than the covered set. It combines all possibilities to the so called *reachability set*.

Definition 2.22 (Reachability set).
For a Petri-Net $(\mathcal{N}, \mathcal{C}_{\mathcal{A}}, \mathcal{C}_{\mathcal{E}})$ with given initial marking $c_0 \in \mathbb{N}_0^{\mathcal{A}}$ we call $\mathcal{R}(\mathcal{A})$ *reachability set* of the Petri-Net if it is maximal and for each attribute $a \in \mathcal{R}$ there exists a path within the flow relation $\mathcal{F}$. We call $\mathcal{R}(c_0)$ *reachability set of the initial marking* if it is the maximal set of attributes a such that there exists a path within the flow relation $\mathcal{F}$ and each method along this path is enabled.

In the production Petri-Net, the reachability set $\mathcal{R}(\mathcal{A})$ includes all markings that can be reached from the initial marking $c_0 \in \mathbb{N}_0^{\mathcal{A}}$ by firing any sequence of methods. Each marking represents a possible configuration of the production process, showing how many components are available, being processed, or completed at a given time.
For instance, in our example network from Figure 2.9 if the initial marking indicates that motorboxes, gearboxes and gears are available but only either gearboxes and motorboxes can be assembled, then $\mathcal{R}(\mathcal{A})$ still contains all states that can result from executing process methods. In this way, the reachability set describes the dynamic evolution of the manufacturing system and all achievable combinations of partial and completed assembly stages.

Remark 2.23
Note that the reachability set depends on the initial marking. Moreover, not the entire coverability search must be executed to check if a specific attribute can be reached.

A short way to compute whether or not a marking is coverable is given by the incidence matrix.

Theorem 2.24 (Coverability).
Consider a Petri-Net $(\mathcal{N}, \mathcal{C}_{\mathcal{A}}, \mathcal{C}_{\mathcal{E}})$ together with an initial marking $c_0 \in \mathbb{N}_0^{\mathcal{A}}$ and a terminal marking $c \in \mathbb{N}_0^{\mathcal{A}}$ to be given. Then c is coverable iff

$$\exists x \in \mathbb{N}_0^{\mathcal{E}} : \; c = c_0 + \mathcal{I} \cdot x. \tag{2.12}$$

From equation (2.12) we directly obtain that a marking is coverable if

$$x = \mathcal{I}^{-1} (c - c_0)$$

is solvable. The result corresponds a sequence of methods $m \in \mathcal{M}$ which need to be fired resulting in a path of intermediate attributes.
To further answer the question which attributes can be set, i.e. what is reachability set of a initial marking, we directly obtain that it is identical to the maximal coverable set. Hence, we have the following:

Theorem 2.25 (Reachability).
For a Petri-Net $(\mathcal{N}, \mathcal{C}_{\mathcal{A}}, \mathcal{C}_{\mathcal{E}})$ and an initial marking $c_0 \in \mathbb{N}_0^{\mathcal{A}}$, the reachability set is given by

$$\max_{c \in \mathbb{N}_0^{\mathcal{A}}} c : \; \exists x \in \mathbb{N}_0^{\mathcal{E}} : \; x = \mathcal{I}^{-1} (c - c_0). \tag{2.13}$$

2.4.2. Liveness

Having answered the questions regarding reachability and coverability, we next address liveness. The latter is a property that expresses whether the system modeled by a Petri-Net can continue to operate indefinitely without entering a deadlock or a terminal state. In other words, no method becomes permanently disabled. Hence, a Petri-Net is live if, from every reachable marking, it is always possible — through some sequence of method firings — to eventually enable any method in the network.

Definition 2.26 (Liveness).
Consider a Petri-Net $(\mathcal{N}, \mathcal{C}_{\mathcal{A}}, \mathcal{C}_{\mathcal{E}})$ and an initial marking $c_0 \in \mathbb{N}_0^{\mathcal{A}}$. Then we call a method $m \in \mathcal{M}$

- *dead* if no path exists such that m can be fired,
- *quasi-live* if m can be fired at least once in a path,
- *live* if it is quasi-live for every marking $c \subset \mathcal{R}(c_0)$.

The first case is self-speaking and illustrated in Figure 2.10. Here, since it is not possible to provide a token to attribute a_1 we have that method m_2 is dead.

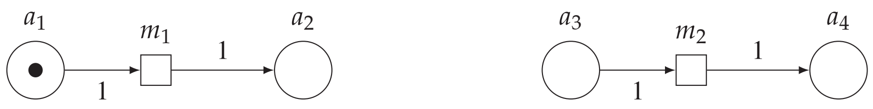


Figure 2.10.: Example of a dead Petri-Net

The other two cases show a fundamental difference. While quasi-live means that the method may be executed but is potentially not executed, live means that the Petri-Net is deadlock-free independent of the choice of the path/firing sequence. To illustrate this, we modify the previous example slightly to Figure 2.11.

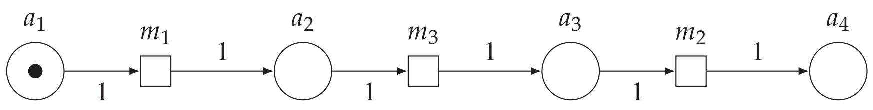


Figure 2.11.: Example of a quasi-live Petri-Net

For this example, we have that all methods are quasi-live for $c = \{a_1\}$, but not live if $c \subset \mathcal{A} \setminus \{a_1\}$. In the latter case, it is not possible to fire method m_1.
A typical structure to obtain a live Petri-Net is to create a circle as sketched in Figure 2.12.
By having such a circle, it is ensured that any attribute $a \in \mathcal{A}$ may be set by some sequence of method firing, even if methods are added which range outside the circle such as attribute a_7 in the example setting. Practical applications of such circles are routines that sequentially check whether subprocesses are still operational
To achieve such a property in general, the following two cases must at least be avoided such that the system does not get stuck:

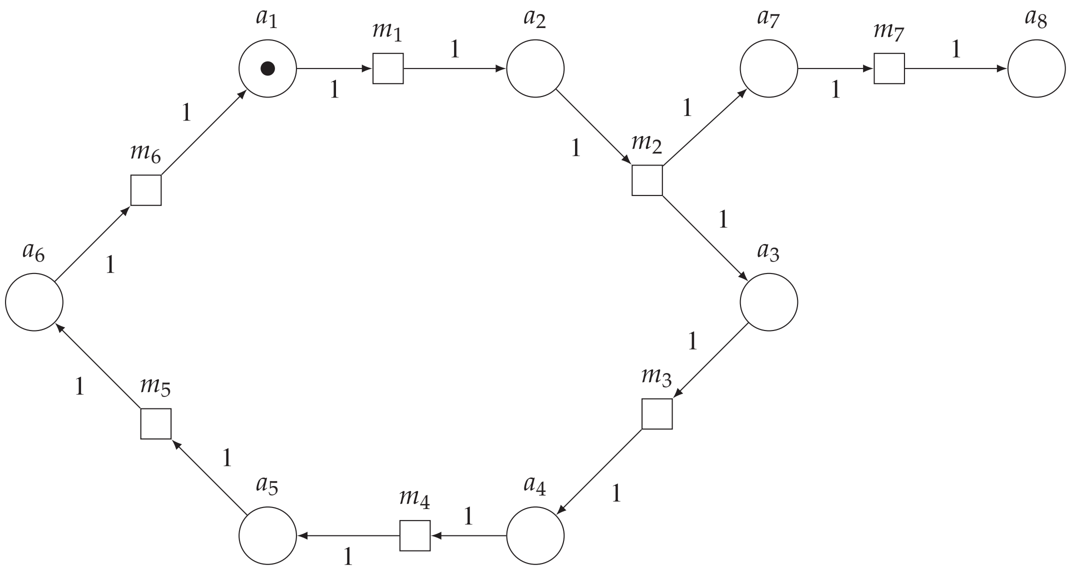


Figure 2.12.: Example of a live Petri-Net

Theorem 2.27 (Liveness conflicts).
Consider a Petri-Net $(\mathcal{N}, \mathcal{C}_{\mathcal{A}}, \mathcal{C}_{\mathcal{E}})$. *Then the network is live if no directional conflicts exist*

$$\bullet m_1 \cap \bullet m_2 = \emptyset, \tag{2.14}$$

$$m_1 \bullet \cap m_2 \bullet = \emptyset. \tag{2.15}$$

Remark 2.28
Note that conditions (2.14)*,* (2.15) *are necessary only.*

A conflict in predecessor attributes, i.e., a violation of (2.14), may occur when an attribute is required by multiple methods, but its marking is insufficient to satisfy the multiplicities of all requesting methods. In the context of the production Petri net, such a situation arises when a limited number of motors or gears are simultaneously needed for different assembly steps. For instance, if both the install motor and mount gearbox methods require access to the same set of motors, but only one token (motor) is available, a resource conflict occurs, preventing one of the methods from firing.
Similarly, a conflict in successor attributes, corresponding to a violation of (2.15), may occur when the firing of one method blocks another method by occupying or exhausting a shared output attribute. In the production Petri net, this could happen if the storage area for completed powertrains has limited capacity. When the finish assembly method fires and places a token into

this storage attribute, the space may become full, preventing the install motor method (and therefore subsequent assembly processes) from proceeding.
Both types of conflicts reduce the liveness of the system, as they create conditions where certain methods become permanently disabled due to resource competition or capacity constraints. Analyzing these conflicts helps identify potential bottlenecks in production and supports the design of balanced workflows and resource management strategies.

2.4.3. Safeness

Finally, we address the property of boundedness, which is closely related to the concept of safeness in Petri nets. Boundedness ensures that the number of tokens in each place remains within a finite limit for all possible firing sequences, thereby guaranteeing that the modeled system cannot overflow or accumulate resources without bound. A Petri net is called k-bounded if no place ever contains more than k tokens in any reachable marking.
In the context of the production Petri-Net, boundedness means that storage areas, buffers, or resource pools — such as bins for gears or waiting zones for assembled powertrains — have finite capacities that are never exceeded, regardless of how methods (e.g., install motor, mount gearbox, finish assembly) are fired.
For ease of exposition, we consider the simplest case of 1-boundedness[1] here:

Definition 2.29 (Safeness / 1-boundedness).
A Petri-Net $(\mathcal{N}, \mathcal{C}_{\mathcal{A}}, \mathcal{C}_{\mathcal{E}})$ with initial marking $c_0 \in \mathbb{N}_0^{\mathcal{A}}$ is *1-bounded* or *safe* if

$$\forall c \in \mathcal{R}(c_0): \; c(a) \leq 1 \; \forall a \in \mathcal{A}. \tag{2.16}$$

Basically, this definition says that each attribute may at most have one marking. We then directly obtain the following:

Theorem 2.30 (Sufficient conditions for a safeness).
A Petri-Net $(\mathcal{N}, \mathcal{C}_{\mathcal{A}}, \mathcal{C}_{\mathcal{E}})$ is safe iff c_0 has at most one unit element. A live network is safe iff c_0 has exactly one unit element.

In the first part, this results says that a Petri-Net is safe if the initial marking c_0 assigns at most one token to each attribute — that is, no place can contain more than one unit at any time. This condition ensures that the modeled production system behaves realistically, without multiple overlapping resources or process instances occupying the same logical position. For example, if each

[1] also referred to as safeness

place in the Petri-Net represents a physical location or a resource — such as motor available, gear ready, assembly station occupied, or powertrain stored — then safeness guarantees that there is never more than one motor in a single availability slot or more than one powertrain in the same storage area. This prevents unphysical states like multiple assemblies sharing one workstation simultaneously.

The second part of the definition specifies that a live and safe Petri-Net requires exactly one token in each initially marked attribute. In the production context, this means the process begins with precisely one available unit per required resource — for example, one motor, one gear, and one free assembly station. This configuration ensures that every method (such as install motor, mount gearbox, or finish assembly) can eventually fire, keeping the system operational without deadlocks.

In summary, safeness ensures that the production Petri-Net respects physical and logical resource limits, while liveness ensures that the process continues indefinitely. A live and safe production system thus corresponds to a balanced process where resources circulate correctly and no step ever exceeds its designed capacity, cf. Figure 2.13 for an example.

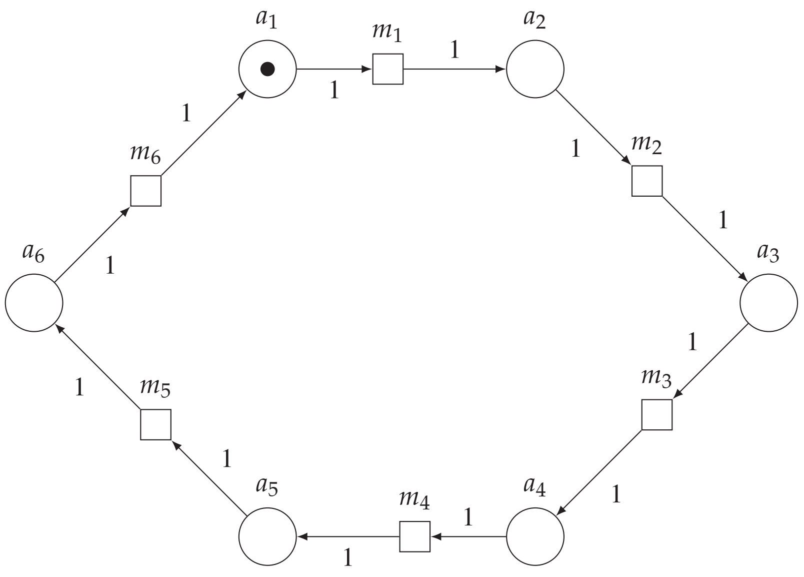


Figure 2.13.: Example of a live and safe Petri-Net

While Theorem 2.30 states sufficient conditions, also necessary conditions are available:

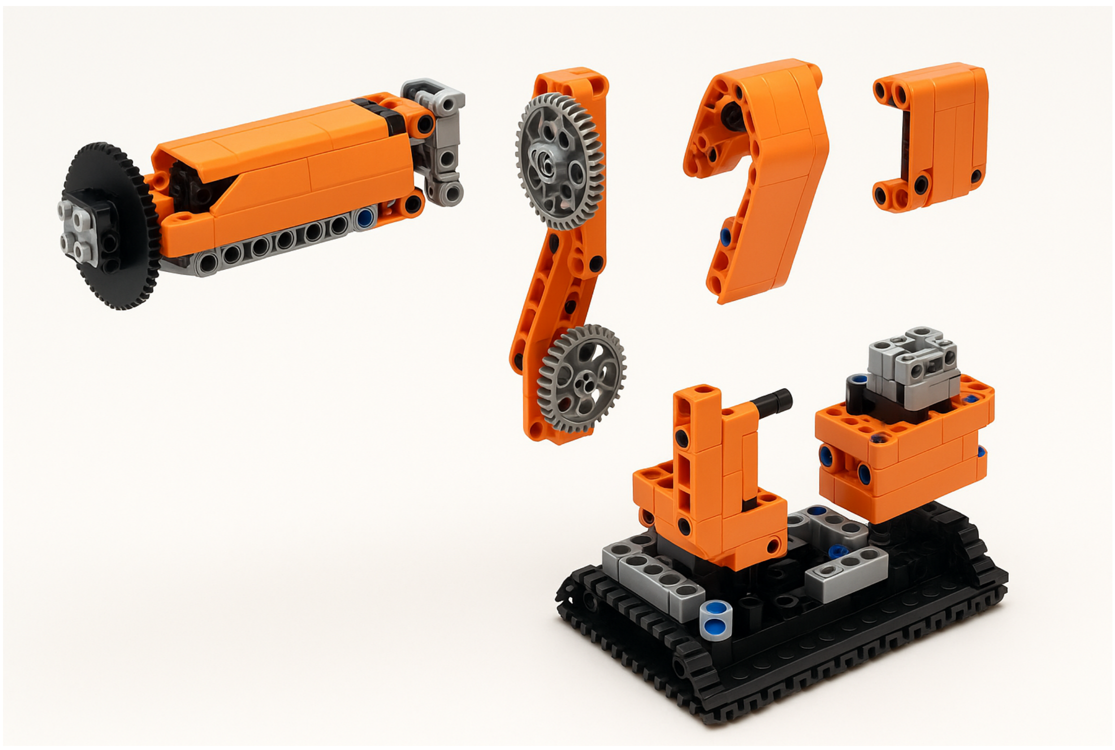
Theorem 2.31 (Necessary conditions for safeness).
Given a Petri-Net $(\mathcal{N}, \mathcal{C}_\mathcal{A}, \mathcal{C}_\mathcal{E})$, there exists a safe and live marking iff $\mathcal{N}$ is strongly connected.

In the production context, strong connectivity implies that all process stages and resources are mutually reachable through some sequence of methods. For example, tokens representing motors,

gears, or assembled powertrains can circulate through the entire network — from component availability, through assembly, to storage, and back to resource release — without any isolated subprocess. If this connectivity is broken (e.g., one assembly line or buffer is disconnected), tokens can become trapped in a subsystem. In such a case, certain methods — like install motor or finish assembly — would eventually be disabled because the necessary resources or outputs can no longer be reached. The system would then lose liveness, and the disconnected part might accumulate tokens, violating safeness.
The ability to check safeness is one of the major applications, for which Petri-Nets are used in industrial practice. Yet, respective algorithms require an in-depth connection between the safety property and the incidence matrix, which is outside the scope of this lecture. For further details, we refer to the work of Murata [25].

In the upcoming chapter, we will apply the idea of configurations to split the system/process into smaller parts. The idea behind the latter is to derive standardized parts with defined interfaces, for which conditions such as safeness have already been shown. Such parts will then allow us to build up more complex systems.

CHAPTER 3 SEPARATION


Generated with chatgpt.com

Modularity is a chunky word for the elegant idea of big things made from small things.

Bent Flyvbjerg

In the past chapter, we discussed networks to represent systems and processes. Networks are a very general tool to indicate connection and evaluate systemic properties such as reachability, liveness, and safeness. In the set used here, it is particularly difficult to add time. While the latter can be done, it may not be in the user's best interest to do so as the description becomes more complex. Instead, in this chapter we discuss the idea to further reduce complexity via a divide–and–conquer strategy. In particular, our aim is to generate modules, which can be used in various settings or applications, and which provide a standard interface. Regarding the latter, we are interested in interfaces, which are simple, i.e. a low number of edges connecting modules. Reconsidering our overall path of automation, Figure 3.1 indicates the focus of the present chapter.

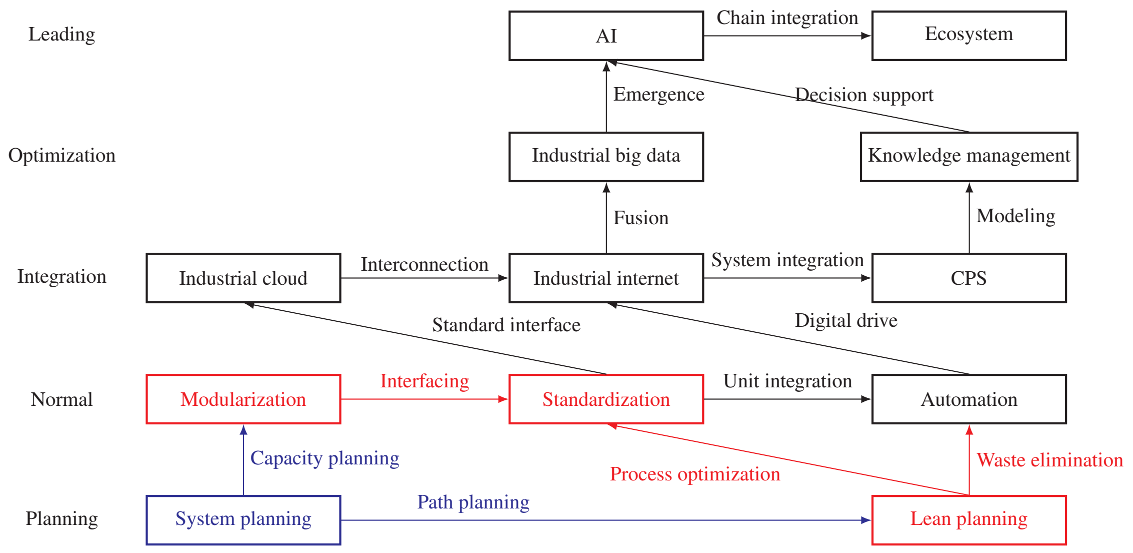


Figure 3.1.: Integrated implementation path of automation

To this end, we first discuss modularization, i.e., how to precisely cut down the system/process into smaller ones and what is a good cut [5, 22]. In the second step, we consider the interfaces between these modules and how these can be standardized. Note that an interface can be with respect to information, energy, goods, and people. Both standardization and modularization need to be considered from a global perspective to generate efficiency and avoid waste. These aims stand at the core of lean planning, which we discuss at the end of this chapter.

3.1. Modularization

As outlined, the aim of modularization is to divide the system/process into smaller parts. We will use the network description derived in the previous chapter to identify similar objects. In network theory, these are known as clusters or communities, and similarity is based on topology,

i.e. connectedness.
We would like to point out that modularization is not only applicable to mechanical systems, as illustrated by the introductory picture in this chapter. This idea can also be applied to systems such as the Braunschweig Tram System (see Figure 3.2). In this example, attributes represent stations and edges represent rail tracks.

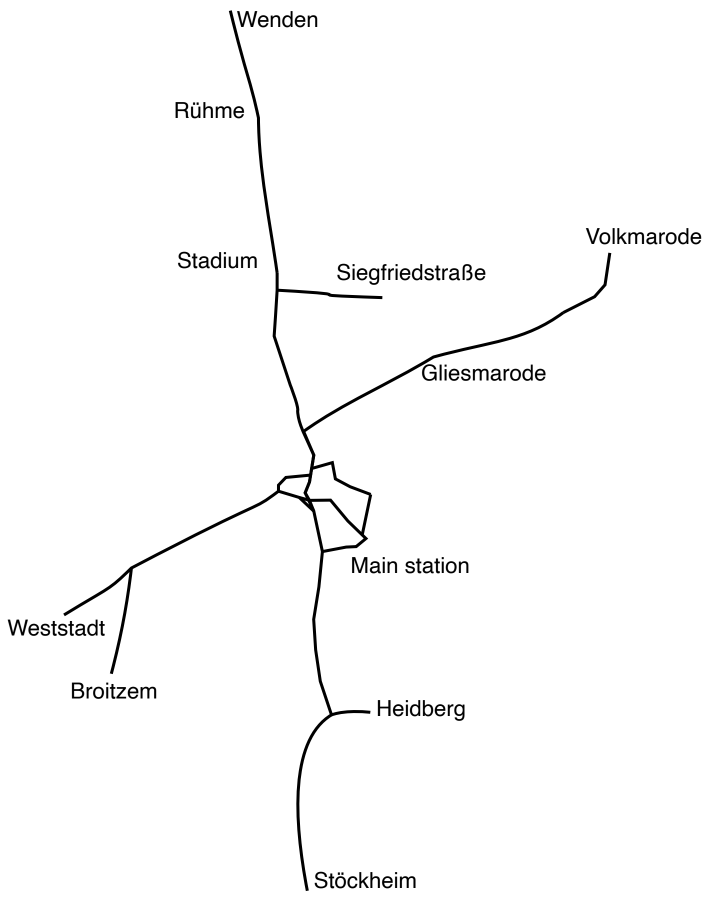


Figure 3.2.: Example of the Tram Network Plan for Braunschweig

If such a system should be operated by two different companies, we would be interested in a suitable cut, which allows a good cooperation between the companies and good transfer of people. Abstracting from this idea, modularization bears a variety of advantages:

- Complexity: If only a module needs to be treated, the overall system complexity can be reduced drastically, resulting in lower costs and faster development cycles.
- Replacement: If modules can be exchanged, cross usage (like cross products) can be applied to assure quality or accelerate technical iteration/innovation.
- Diversity: As modules may be substituted, this leads to diverse/customizable solutions.

However, compared with integrated architecture, on a management level, modular architectures often lack overall coordination and performance optimization. Hence, its weak link is often found in the interface between modules.
In order to modularize a system/process, we first introduce the concept of a cluster, which is basically an intersection free coverage of the network by subsets of it. More formally:

Definition 3.1 (Clustering/Modularization).
Consider a network $(\mathcal{O}, \mathcal{E})$. Then we call $S = \{S_1, \ldots, S_{\#S}\}$ a *clustering* or *modularization* if

$$\begin{aligned} S_j &\neq \emptyset \ \forall j \in \{1, \ldots, \#S\} \\ S_j \cap S_k &= \emptyset \ \forall j, k \in \{1, \ldots, \#S\} \\ \bigcup_{j=1}^{\#S} S_j &= \mathcal{O}. \end{aligned} \tag{3.1}$$

The first part essentially means that no part of a clustering, i.e. a module, shall be empty. The second part calls for a division between each pair of modules; in other words, no attribute can be part of more than one module. The final part shows that all attributes must be covered by the clustering; that is, each attribute must be an element of a module.
To derive clusters, we need to consider several quality measures, which will be our key performance criterion (see Definition 1.13). To describe a network topology, we first need to distinguish between undirected and directed ones.

Definition 3.2 (Un-/directed networks).
Suppose a network $(\mathcal{O}, \mathcal{E})$ to be given. Then we call the network to be directed if the edges $e = (e_1, e_2) \in \mathcal{E}$ are directed, i.e. $e_1 \in \mathcal{O}$ is the starting point and $e_2 \in \mathcal{O}$ is the endpoint of the edge.

In terms of drawing, the difference between undirected and directed networks simply reduces to adding an arrow to an edge in the directed case. Therefore, in the undirected case, an edge represents two arrows, a fact that will be important for us when counting connections later on.
In the undirected case, we then obtain the feature of modularity via the following:

Definition 3.3 (Modularity measure).
Given a network $(\mathcal{O}, \mathcal{E})$ where $e \in \mathcal{E}$ are undirected, let S be a clustering of it. Then we call

$$J_{\text{Modularity}}(S) := \sum_{j=1}^{\#S} \left(P(S_j \mid S_j) - P(S_j \mid \mathcal{O})^2 \right) \tag{3.2}$$

the *modularity* of a clustering S where $P(S_j \mid S_j)$ denotes the probability of intra-cluster edges $e = (e_1, e_2)$ in cluster S_j, that is

$$P(S_j \mid S_j) := \frac{\#\left\{(e_1, e_2) \mid e_1 \in S_j, e_2 \in S_j\right\}}{\#\mathcal{E}} \tag{3.3}$$

and $P(S_j \mid \mathcal{O})$ the probability of either an intra-cluster edge in cluster S_j or of an inter-cluster edge incident from cluster S_j, i.e.

$$P(S_j \mid \mathcal{O}) := \frac{\#\left\{(e_1, e_2) \mid e_1 \in S_j, e_2 \in \mathcal{O}\right\}}{\#\mathcal{E}} \tag{3.4}$$

Similarly, in the directed case, modularity reads

Definition 3.4 (Modularity measure for directed networks).
Suppose an directed network $(\mathcal{O}, \mathcal{E})$ to be given an let S be a clustering of it. Then modularity of clustering S is given by

$$J_{\text{Modularity}}(S) := \sum_{j=1}^{\#S} \left(P(S_j \mid S_j) - P(S_j \mid \mathcal{O})P(\mathcal{O} \mid S_j)\right) \tag{3.5}$$

where the probability of either an intra-cluster edge in cluster S_j or of an inter-cluster edge incident from cluster S_j is given by

$$P(S_j \mid \mathcal{O}) := \frac{\#\left\{(e_1, e_2) \mid \left(e_1 \in S_j, e_2 \in \mathcal{O}\right)\right\}}{\#\mathcal{E}} \tag{3.6}$$

$$P(\mathcal{O} \mid S_j) := \frac{\#\left\{(e_1, e_2) \mid \left(e_1 \in \mathcal{O}, e_2 \in S_j\right)\right\}}{\#\mathcal{E}} \tag{3.7}$$

reflecting both inbound and outbound edges.

Modularity is a value (or score), which can be used to assess clustering. It formalizes the intuitive idea that good clusters contain more internal edges than expected in a comparable random graph. In terms of modularity, we are interested to identify a clustering S such that the links between clusters are minimized, that is each cluster S_j is more or less self-contained. The idea behind self-containedness is to allow us to treat such a cluster as standalone or *embedded*, i.e., to treat it separately from the remaining network. Hence, a clustering S is considered to be good if the value of a given metric $J(S)$ is maximized (and at best 1).

Task 3.5
Consider the network given in Figure 3.3. Compute modularity of the clustering $S = \{\{a,b,c\},\{d,e,f,g\},\{h,i\}\}$.

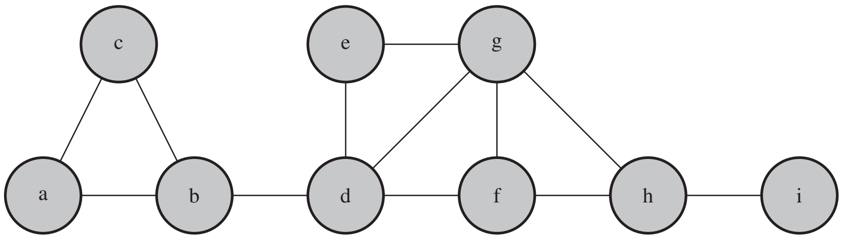


Figure 3.3.: Cluster of a network

Solution to Task 3.5: We obtain

$$J_{\text{Modularity}}(S) = \left(\frac{6}{24} - \left(\frac{7}{24}\right)^2\right) + \left(\frac{10}{24} - \left(\frac{13}{24}\right)^2\right) + \left(\frac{2}{24} - \left(\frac{4}{24}\right)^2\right) \approx 0.34.$$

A different measure is the so called conductance, which is a measure to check whether a network has a bottleneck. Conductance captures how „tight“ or „leaky“ a cluster is. Instead of comparing the cluster to a null model (like modularity), conductance uses a boundary-to-volume ratio: few boundary edges relative to internal total degree imply strong structure.

Definition 3.6 (Conductance of undirected network).
Given a undirected network $(\mathcal{O},\mathcal{E})$ let S be a clustering of it. Then we call

$$J_{\text{Conductance}}(S_j) := \frac{Q(S_j,\overline{S}_j)}{\min\{Q(S_j,\mathcal{O}),Q(\overline{S}_j,\mathcal{O})\}} \tag{3.8}$$

conductance of a cluster S_j where $\overline{S}_j = \mathcal{O} \setminus S_j$ and

$$Q(S_j,\overline{S}_j) = \#\left\{(e_1,e_2) \mid e_1 \in S_j, e_2 \in \overline{S}_j\right\}$$

denotes the quantity of edges connecting the cluster S_j to its network complement $\overline{S}_j$,

$$Q(S_j,\mathcal{O}) = \#\left\{(e_1,e_2) \mid e_1 \in S_j, e_2 \in \mathcal{O}\right\}$$

denotes the quantity of edges within and connecting the cluster S_j to its network complement, and

$$Q(\overline{S}_j, \mathcal{O}) = \#\left\{(e_1, e_2) \mid e_1 \in \overline{S}_j, e_2 \in \mathcal{O}\right\}$$

denotes the quantity of edges outside and connecting the cluster S_j to its network complement. Moreover, we call

$$J_{\text{Conductance}}(S) := 1 - \frac{1}{\#S} \sum_{j=1}^{\#S} J_{\text{Conductance}}(S_j) \tag{3.9}$$

conductance of a clustering S.

Similarly, we obtain the following for a directed network.

Definition 3.7 (Conductance of directed network).
Suppose a directed network $(\mathcal{O}, \mathcal{E})$ with clustering S to be given. Then conductance of a cluster S_j is given by (3.8) where $\overline{S}_j = \mathcal{O} \setminus S_j$ and

$$Q(S_j, \overline{S}_j) = \#\left\{(e_1, e_2) \mid \left(e_1 \in S_j, e_2 \in \overline{S}_j\right) \vee \left(e_1 \in \overline{S}_j, e_2 \in S_j\right)\right\}$$

denotes the quantity of edges entering and leaving the cluster S_j,

$$Q(S_j, \mathcal{O}) = \#\left\{(e_1, e_2) \mid \left(e_1 \in S_j, e_2 \in \mathcal{O}\right) \vee \left(e_1 \in \mathcal{O}, e_2 \in S_j\right)\right\}$$

denotes the quantity of edges within as well as entering and leaving the cluster S_j, and

$$Q(\overline{S}_j, \mathcal{O}) = \#\left\{(e_1, e_2) \mid \left(e_1 \in \overline{S}_j, e_2 \in \mathcal{O}\right) \vee \left(e_1 \in \mathcal{O}, e_2 \in \overline{S}_j\right)\right\}$$

denotes the quantity of edges outside and entering/leaving the cluster S_j.

The idea of detecting bottlenecks is totally different from separability/embeddedness. Here, we are interested to identify those cuts, which put a limit on the flow of goods/information/people/energy within the network. To similarly be able to assess the network in terms of required flow, conductance is inverted in Equation (3.9), again resulting in a maximization problem with aim 1.

Task 3.8
Reonsider the network given in Figure 3.3. Compute conductance of the clustering $S = \{\{a,b,c\},\{d,e,f,g\},\{h,i\}\}$.

Solution to Task 3.8: For $S_1 = \{a,b,c\}$, $S_2 = \{d,e,f,g\}$, $S_3 = \{h,i\}$, we obtain

$$J_{\text{Conductance}}(S_1) = \frac{1}{7}, \qquad J_{\text{Conductance}}(S_2) = \frac{3}{11}, \qquad J_{\text{Conductance}}(S_3) = \frac{2}{4}$$

and

$$J_{\text{Conductance}}(S) = 1 - \frac{1}{3} \cdot \left(\frac{1}{7} + \frac{3}{11} + \frac{2}{4}\right) \approx 0.69.$$

Finally, the coverage of a network is also typically considered a marker for modularization. Here, coverage differs from coverability, which we considered in Chapter 2 to be the ability to set certain attributes. Coverage describes the property of subsets within a network; in particular, it quantifies the proportion of edges within clusters rather than between them.

Definition 3.9 (Coverage).
Given a network $\mathcal{N} = (\mathcal{O}, \mathcal{E})$ and a clustering S of it, the quotient

$$J_{\text{Coverage}}(S) := \frac{\sum_{j=1}^{\#S} Q(S_j, S_j)}{\#\mathcal{E}} \tag{3.10}$$

is called *coverage* of a cluster S_j where

$$Q(S_j, S_j) = \#\left\{(e_1, e_2) \mid e_1 \in S_j, e_2 \in S_j\right\}$$

denotes the quantity of edges inside of cluster S_j.

Coverage is a simple, edge-based KPI. Unlike modularity, it does not use a probabilistic null model, and unlike conductance, it does not use a boundary-to-volume ratio. Instead, it simply counts how many edges the clustering preserves internally. This makes it easy to interpret, but also limits its usefulness (it may reward overly coarse clusterings).
Therefore, coverage tells us what percentage of edges are contained within the clustering and what percentage are neglected. Therefore, it is irrelevant whether we consider directed or undirected

networks; the aim is to maximize this number to 1. However, over-optimisation of this metric is not useful, as the optimal result is to define the cluster as identical to the set of objects $\mathcal{O}$.

Task 3.10

Reconsider the network given in Figure 3.3. Compute coverage of the clustering $S = \{\{a,b,c\},\{d,e,f,g\},\{h,i\}\}$.

Solution to Task 3.10: We obtain

$$J_{\text{Coverage}}(S) = \frac{6+10+2}{24} = 0.75.$$

Remark 3.11

In the literature, additional metrics like information recovery or normalized mutual information are considered as well, which are outside the scope of this lecture, cf., e.g., [5].

One method to derive a clustering that (approximately) maximizes any of the above metrics is the so-called Louvain algorithm. The central idea of this method is to approximate a maximum through an iterative procedure in which the chosen metric is repeatedly evaluated to determine whether relocating an object (i.e., a node) from one cluster into another yields an improvement. Each node is considered individually, and the algorithm examines whether moving it to the community of one of its neighbors increases the clustering quality measure (e.g., modularity, coverability or coverage). Nodes are reassigned greedily to the cluster that provides the highest positive gain. Once no further improvement can be achieved through local movements, the resulting clusters are collapsed, thereby producing a reduced network in which edges represent aggregated connectivity patterns. The algorithm then repeats the same local reassignment procedure on this smaller graph. This hierarchical process effectively builds a tree-like structure and reveals clusters at different resolutions. However, due to its greedy nature and the absence of a global search mechanism, the resulting solution is only locally optimal and may depend on the order in which nodes are processed [1].

Algorithm 3.12 (Louvain algorithm for clustering)
Consider a network $(\mathcal{O}, \mathcal{E})$.

1. For $j = 1, \ldots, \#\mathcal{O}$ set $S_j = \{\mathcal{O}_j\}$

```
2. Compute metric (modularity (3.2) or conductance (3.9) or coverage (3.10))
3. Do
    a) For $j = 1, \ldots, \#\mathcal{O}$
        i. For each edge $(j,k)$ or $(k,j)$ of object $\mathcal{O}_j$ with $k \in \mathcal{O}_k$, $\mathcal{O}_j \neq \mathcal{O}_k$
            ■ Remove $\mathcal{O}_j$ from its cluster and add it to cluster of $\mathcal{O}_k$
            ■ Recompute metric
            ■ If metric not improved, revert move of $\mathcal{O}_j$
        ii. Remove empty clusters
   while metric is improved
```

Having the possibility to cluster a system/process at hand, the question is which clustering is appropriate regarding the strategy and goals of the system/process.

3.2. Standardization

As we have seen in Table 1.1, there are two main strategies: cost leadership and differentiation. Regarding cost leadership, standardization aims to develop components, subassemblies, processes, structures, types, units, goods, services, or programs that can be interchanged. A typical example is the use of a common vehicle platform in the automotive industry, where multiple car models share the same chassis, battery pack geometry, or sensor architecture. Such a standardized module can then be produced in high volumes, reducing unit costs. Reconsidering the previous Section 3.1, this corresponds directly to applying modularity: modules are designed to be internally coherent but externally decoupled, allowing reuse across product variants. In a network sense, this is the opposite of customization, which is the aim of the differentiation strategy and corresponds to maximizing coverage. For instance, offering individualized interior configurations, multiple motor options, or customer-specific software packages increases the number of edges within variant-specific sub-networks and decreases interchangeability.

Remark 3.13
Note that in the context of customization, alternatives are treated as a different set of sub-networks, which all need to be present in the network. For example, if a product family supports three different types of charging interfaces, the complete network must maintain three distinct interface branches, even though each customer only uses one of them.

Since standardization can only provide a cost advantage if options are dropped or consolidated, both strategies naturally work against one another. A company cannot simultaneously maximize shared components and maximize customer-specific differentiation; increasing one reduces the other. For example, a smartphone manufacturer could standardize its camera module across all models to reduce cost, but this directly limits the degree of differentiation it can offer between entry-level and premium products.

An essential component of meaningful standardization is the definition of module interfaces. To enable flexible reuse, the interfaces must be as independent as possible from special customer requirements and from the internal implementation of each module. A classical example is the standardized USB-C port: regardless of the device it is embedded in, the interface remains identical, allowing modules (cables, chargers, peripheral devices) to be interchanged freely. Similarly, in software architectures, a clean API allows internal components to change without affecting downstream modules.

The standardization procedure typically consists of the three steps shown in Figure 3.4.

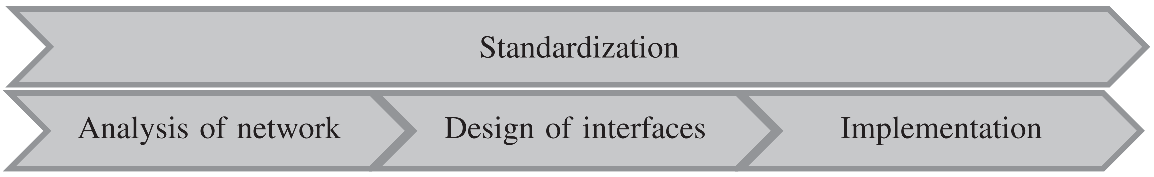


Figure 3.4.: Structure of the standardization process

For example, in the automotive context, these steps may involve:

- Identifying similarities across product variants (e.g., shared mounting points for traction motors).
- Defining standardized modules (e.g., a universal inverter module with a fixed mechanical and electrical interface).
- Eliminating redundant variant-specific designs (e.g., reducing five brake-controller versions to two standardized ones).

This process ensures that modules remain interchangeable, production complexity decreases, and economies of scale can be exploited—while still allowing a controlled degree of differentiation through combinable modules.

In the previous Section 3.1, we discussed ideas and methods for how to modularize a network. Now, our task is the proper integration of the criteria and requirements put to a modularization to be optimized/satisfied, which is termed standardization.

Definition 3.14 (Standardization).
Consider a network $(\mathcal{O}, \mathcal{E})$. We call a clustering S a *standardization* if

- S is optimal with respect to a defined metric $J(\cdot)$, and
- S satisfies all constraints defined by requirements.

Task 3.15
Reconsider the network from Figure 3.3. Suppose we require object f to be held singularly in one cluster. Plot the standardization by adapting the clustering S from Task 3.5 respectively.

Solution to Task 3.15: Results are given in Figure 3.5 with

$$\begin{aligned} J_{\text{Modularity}}(S) = &\left(\frac{6}{24} - \left(\frac{7}{24}\right)^2\right) + \left(\frac{6}{24} - \left(\frac{10}{24}\right)^2\right) \\ &+ \left(\frac{0}{24} - \left(\frac{3}{24}\right)^2\right) + \left(\frac{2}{24} - \left(\frac{4}{24}\right)^2\right) \approx 0.24 \end{aligned}$$

Remark 3.16
Note that constraints can also be put on edges and are not restricted to objects.

The main idea we can take from Task 3.15 is that requirements (as they represent restrictions) will lead to a reduction in optimality. In other words, every requirement imposes a constraint on the design space, thereby limiting the degree to which an architecture, product, or system can be optimized with respect to performance, cost, or modularity. Fixed interfaces are a particularly influential type of requirement, because once an interface is defined, it restricts the allowable variation of the connected modules. Hence, special attention must be paid to how these interfaces are defined and which degrees of freedom remain.
On an exemplary basis, we can consider a modular drivetrain in which different battery packs and motors should be combinable. If the mechanical interface between battery and inverter is fixed prematurely — e.g., specifying a certain bolt pattern or connector geometry — then any alternative component that does not satisfy this exact interface must either be redesigned or excluded. This restricts the range of feasible configurations and may reduce the achievable modularity or cost savings. Again, the approach is not limited to the physical world. In software systems, fixing

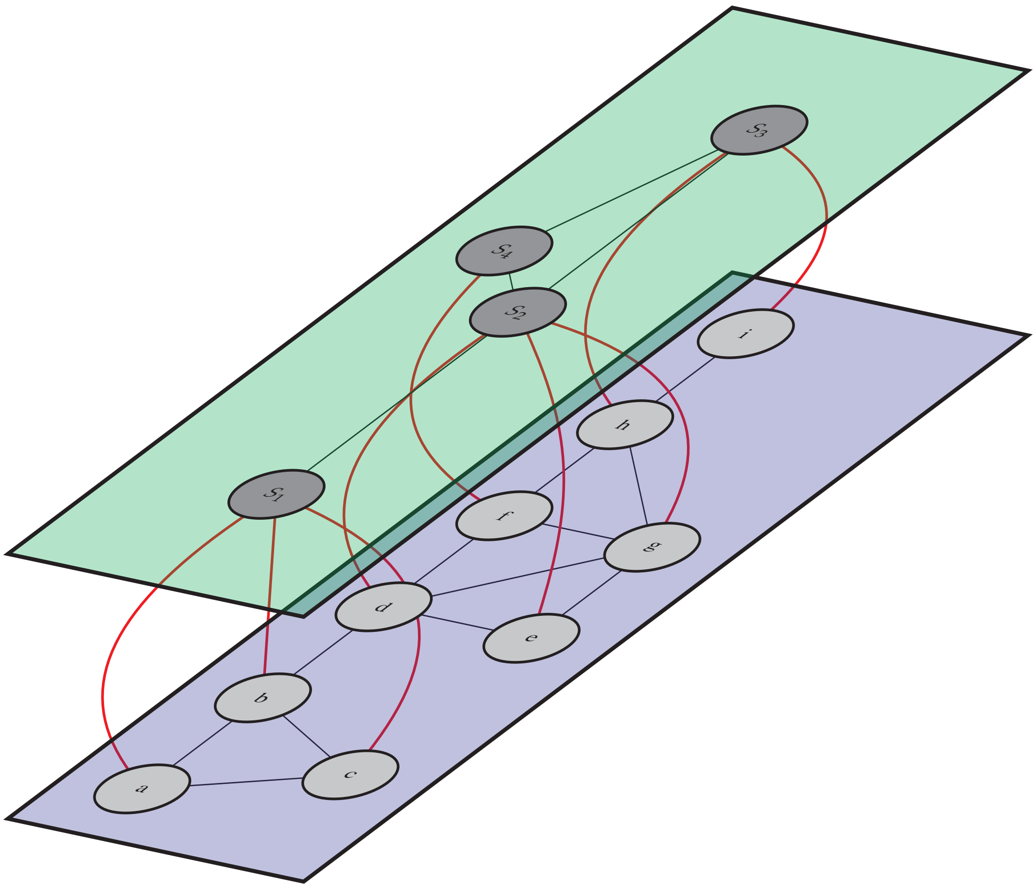


Figure 3.5.: Multi-layer Cluster map using standardization

an API too early may prevent later design choices that would have provided better performance, security, or maintainability.
Considering the definition of a requirement from [17], we can formalize the following:

Definition 3.17 (Requirement).
Suppose a network $(\mathcal{O}, \mathcal{E})$ to be given. Then the conditions

$$\bigcup_{j \in \mathcal{I}} \mathcal{O}_j \in S_k \qquad \text{(Union)} \qquad (3.11)$$

$$\mathcal{O}_i \in S_j \cup \mathcal{O}_k \in S_l \qquad \text{(Split)} \qquad (3.12)$$

$$(e_1, e_2) \in \mathcal{E} : (e_1, e_2) \in S_j \qquad \text{(Containment)} \qquad (3.13)$$

$$(e_1, e_2) \in \mathcal{E} : \ e_1 \in S_j \wedge e_2 \in S_k \qquad \text{(Interface)} \qquad (3.14)$$

can be used to impose *requirements* on a clustering S. The *set of requirements* is denoted by $\mathcal{R}$.

Within the Louvain Algorithm 3.12, these conditions can be integrated easily in the greedy strategy by checking whether or not a requirement needs to be enforced.

Algorithm 3.18 (Louvain algorithm for clustering with requirements)
Consider a network $(\mathcal{O}, \mathcal{E})$ and requirements $\mathcal{R}$.

```
1. For $j = 1, \ldots, \#\mathcal{O}$ set $S_j = \{\mathcal{O}_j\}$
2. Compute metric (modularity (3.2) or conductance (3.9) or coverage (3.10))
3. Do
     a) For $j = 1, \ldots, \#\mathcal{O}$
          i. For each edge $(j,k)$ or $(k,j)$ of object $\mathcal{O}_j$ with $k \in \mathcal{O}_k$, $\mathcal{O}_j \neq \mathcal{O}_k$
               ■ If $\mathcal{O}_j$ can be removed from $S_j$ and added to $S_k$ according to $\mathcal{R}$
                    ▪ Remove $\mathcal{O}_j$ from its cluster and add it to cluster of $\mathcal{O}_k$
                    ▪ Recompute metric
                    ▪ If metric not improved, revert move of $\mathcal{O}_j$
          ii. Remove empty clusters
   while metric is improved
```

Remark 3.19
Note that internal interfaces can be set freely and should be designed to have no impact on the optimality of the clustering given a defined metric. External interfaces on the other hand are subject to negotiation. A quantification of optimality degradation is given by so-called Lagrangian variables, also called shadow prices, which are beyond the scope of this lecture. For details, we refer to [27].

Still, standardization (of modules) offers a number of advantages for both users and manufacturers, cf. Table 3.1.

Table 3.1.: Advantages of standardization for users and manufacturers

User	**Manufacturer**
• Becomes more transparent and easier to use	• Fulfills requirements of existing users
• Reduces error sources by using proven modules	• Allows maintenance and expanding competitiveness
• Minimizes effort for creation and commissioning	• Increases engineering efficiency
• Simplifies diagnosis and troubleshooting	• Simplifies management of component variants (flexibility)
• Clarifies documentation of modules and behavior	• Allows division of tasks into work packages
• Defines interfaces	• Allows virtual commissioning (digital twin)

On the other hand, disadvantages arise in particular for the differentiation strategy. In production, the latter is also called mass customization. In particular, we see the following:

Table 3.2.: Disadvantages of standardization for users and manufacturers

User	**Manufacturer**
• Increases wait time from order placement to shipment	• Renders forecasting of trends/spikes difficult
• Inhibits innovation outside standard ranges	• Increases cost to maintain variety of machinery/tools
• Affects flow of supply chains with partners	• Makes build-up of stock impossible
• Renders tracking of orders/projects difficult	• Increases complexity for returned components

Requirements for standardization may arise from different sources and are sector-specific. Examples of sector-specific norms are ISO/IEC 81346 [12] for an industrial plant or IEC 61512 [9] for batch processes. Here, we focused on the general setting of requirements and how these impact planning.

Link: For further details on standardization in production, we refer to the lecture *Computer Integrated Manufacturing*.

In the following Chapters 4 and 5, we additionally consider the information (IEC 61131 [11]) and control aspect (DIN IEC 60050-351 [3]). Before coming to that, we next consider waste within the planning and how waste may be identified.

3.3. Lean planning

So far, we have examined the objectives of modularization, conductance, coverage and standardization. While all of these aspects are fundamental to breaking down a specific work task, for example by structuring the system, grouping related elements, defining reusable modules or reducing unnecessary variation, lean planning takes an additional conceptual step. Rather than just optimizing how the work is divided, it also questions whether each part of the task is necessary. The primary objective of lean planning is to identify and reduce waste, i.e. any activity, process or structural element that does not contribute to value creation. This approach shifts the focus from organizing complexity to eliminating it altogether. Unlike modularization or standardization, which reorganize or simplify the system, lean planning fundamentally questions the necessity of certain tasks, dependencies, or interfaces. By thinking in terms of networks (Definition 2.5) and key performance indicators (Definition 2.20), waste can be reinterpreted as a structural property of the network itself. Nodes or edges that do not contribute to the intended objective — for example, redundant process steps, unnecessary interactions, excessively tightly coupled components, or rarely used variant-specific branches — represent waste from the perspective of lean planning. Thus, in the network-theoretic framework used throughout this chapter, waste can be defined as follows:

Definition 3.20 (Waste).
Given a network $(\mathcal{O}, \mathcal{E})$ together with a key performance criterion $J : \mathcal{I} \times \mathcal{I}^2 \to \mathbb{R}_0^+$. We call any object $\mathcal{O}_j$ or edge $\mathcal{E}_k$ *waste* if the condition

$$J(\mathcal{O} \setminus \mathcal{O}_j, \mathcal{E} \setminus \mathcal{E}_k) \leq J(\mathcal{O}, \mathcal{E}) \tag{3.15}$$

holds.

On the planning level, our task is to quantify elements with respect to their impact on the considered key performance indicator. Here, the latter definition says that an object or an edge exhibits a neutral or negative contribution toward the key performance criterion. If these objects or edges are not necessary within the network, they should be removed.
Waste is typically characterized by the word „DOWNTIME“, which assembles different types of waste illustrated in Figure 3.6.

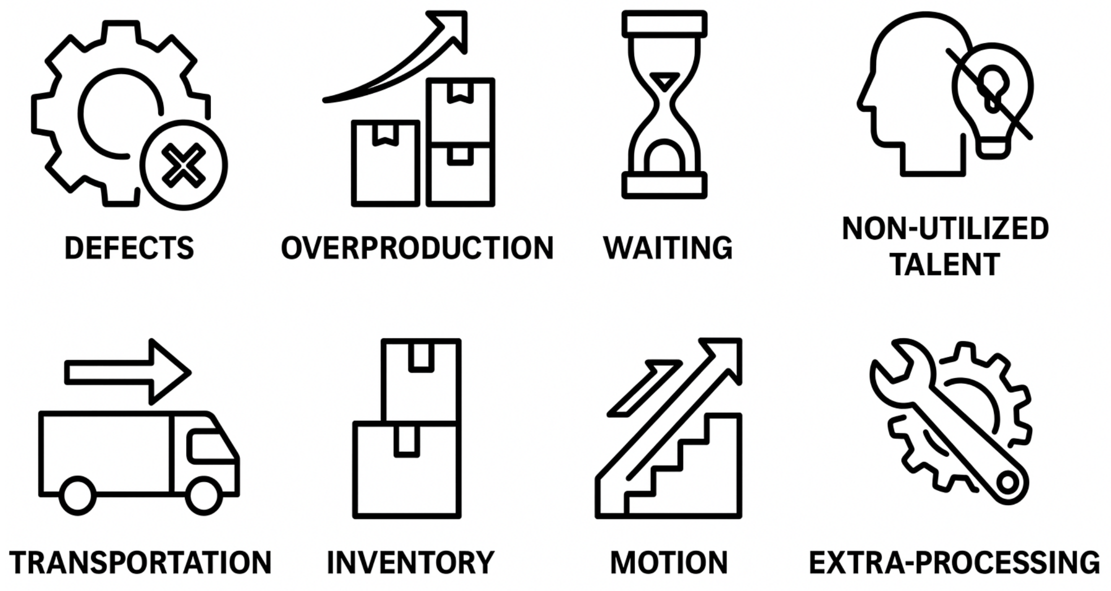


Figure 3.6.: „DOWNTIME“ — 8 types of waste, generated with chatgpt.com

In industrial practice, waste is only one indicator, yet it may already reveal an unwanted interplay between product innovation and process innovation. For instance, an organization that invests heavily in product innovation may introduce highly sophisticated features, novel materials, or cutting-edge technologies. While these innovations may improve performance or differentiation, they often require new manufacturing capabilities, tighter tolerances, or specialized suppliers. As a consequence, production costs may increase substantially, making the product economically unattractive despite its technical superiority. Conversely, focusing too strongly on process innovation may lead to streamlined, highly optimized production workflows — such as extensive automation, reduced variant diversity, or rigidly standardized work sequences. Although this improves efficiency and reduces waste, it can also constrain the design space for new products. Over time, the organization may become locked into its existing processes, making it difficult to introduce radical product innovations. This may result in products that are reliable and inexpensive to manufacture but technologically outdated or insufficiently differentiated in the market. A typical example is a company that prioritizes stable, high-volume manufacturing and therefore postpones or avoids adopting new technologies such as advanced sensors or digital services. As illustrated in Figure 3.7, this tension shows that innovation in either dimension can have unintended consequences if pursued in isolation. Product innovation that outpaces process flexibility leads to inefficiencies and cost overruns, whereas process innovation that becomes too rigid suppresses the organization's ability to adapt its products to changing market demands.

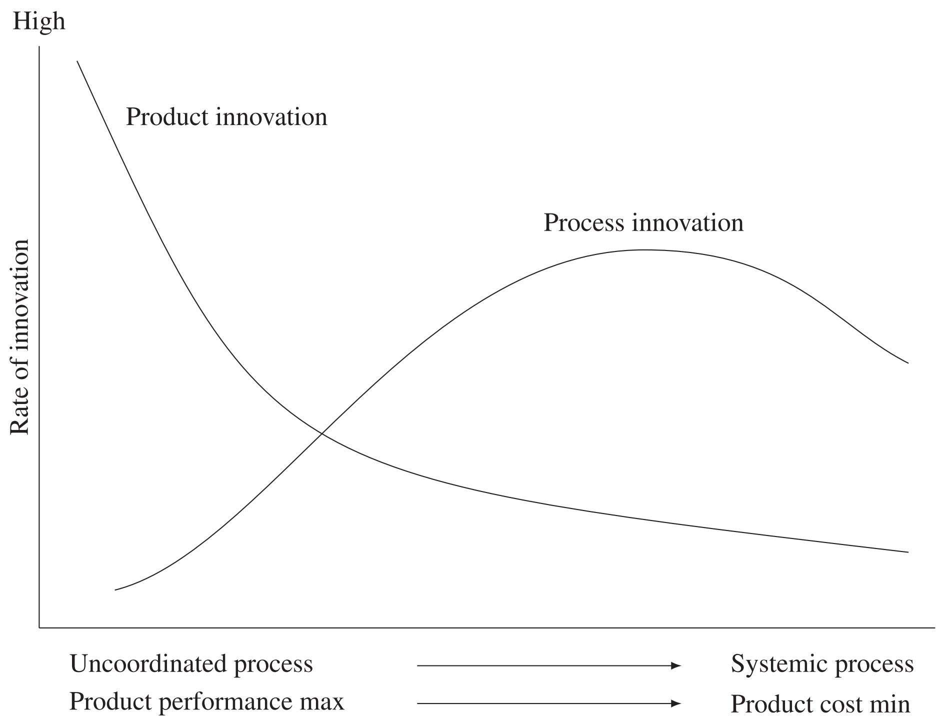


Figure 3.7.: Innovation and stage of development from [30]

Apart from avoiding waste, other measures can be taken, including:

- Align all tasks with customer needs
- Focus on strengths
- Optimize processes/networks
- Improve quality continuously
- Implement costumer centric company principle
- Strengthen self responsibility, empowerment, and team work
- Implement decentralized and costumer focused structures

- Live leadership as service for coworkers
- Implement open information and feedback processes
- Improve culture and attitude within the company

Most of the latter are qualitative and not engineering-related issues. In order to lead a company/process, it is still necessary to be familiar with these tasks but outside the scope of this lecture.

> **Link:** For further details on lean planning considering management and human resource tasks, we refer to the lecture *Operations Management*.

While waste can easily be quantified for a given setting/network, finding an improved one is much more involved and will be the content of Chapter 7 focusing on optimization and leading.

CHAPTER 4 INFORMATION AND COMMUNICATION

Generated with chatgpt.com

Effective teamwork begins and ends with communication.

Mike Krzyzewski

Coming back to the network of systems and processes discussed in Chapter 2, we observe that the links connecting these systems/processes may consist of matter, energy, or information (cf. Definition 1.4 of a process in [3]). Material flows could be physical goods moving through a production line, energy flows could represent electrical or thermal power, and information flows describe data exchanged to coordinate the system's behavior. Since our overarching aim is automation, the present chapter focuses specifically on the information flow required to operate and steer the system. The question of how such information is later processed, interpreted, and transformed into control actions will be addressed in Chapter 5.
In industrial practice, such an information network can consist of a wide variety of heterogeneous elements: software programs running on different platforms, distributed computers, communication lines, machine controllers, sensors, actuators, and supervisory systems. Each of these components may use its own protocols, data formats, and communication standards. The intention of this chapter is therefore to illustrate how these diverse objects — each with their own „language" — can be integrated into a coherent communication structure. To achieve this, we adopt the principle of modularization and apply it to communication itself. Just as modularization in product design separates functions into interoperable modules, modularization in communication separates the information exchanged into well-defined, interchangeable interfaces. This reduces complexity, increases flexibility, and supports scalability in automated environments. Reconsidering our overall path of automation, Figure 4.1 shows the position of the present chapter within that trajectory. We transition from understanding what information is present in the system to structuring and exchanging it in a modular, consistent way—an essential step before employing the control mechanisms introduced in the subsequent chapter.

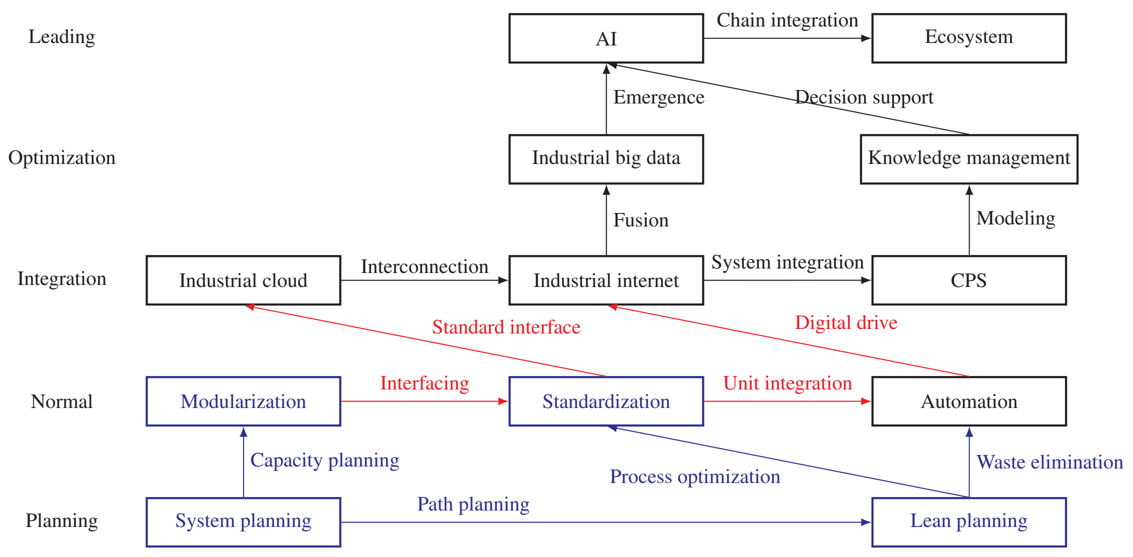


Figure 4.1.: Information in the integrated implementation path of automation

We first address what should be understood by the term *information*. Subsequently, we move towards different transmission networks and communication structures before actually modularizing the information transfer itself. After that, we swiftly address the ideas of channel access and message formats. This structure lays the foundation for the idea of actually controlling the system/process/network at hand in the following chapter.

4.1. Information processing

As outlined earlier, information is one of the fundamental linking components among systems, processes, and networks. However, in Chapter 1, our discussion referred only to the term signal as defined in DIN IEC 60050-351 (2014) [3]. Recall the definition:

> A signal is a physical quantity that conveys information about one or more variable quantities using one or more of its parameters.
>
> DIN IEC 60050-351 (2014)

This definition underlines that a signal is merely the carrier of information—a physical phenomenon such as voltage, current, pressure, displacement, optical intensity, or electromagnetic radiation. In network terminology, a signal corresponds to an edge that provides a channel through which something can be transmitted. The signal itself does not guarantee meaning; it simply enables transport. By contrast, information has no universal formal definition, and its interpretation varies across disciplines such as information theory, computer science, linguistics, cognitive science, and systems engineering. Nevertheless, several properties commonly attributed to information include:

- Reduction of uncertainty (the information changes what we believe about a system)
- Relevance (the information is meaningful in a given context)
- Redundancy (the information may contain repeated or unnecessary parts)
- Freeness (ease of access or sharing)
- Newness (the degree to which the information adds something not previously known)

In practice, information is not identical to raw data. Information emerges only when:

1. context is known, and

2. primary signals are processed and interpreted.

Raw data such as the voltage output of a temperature sensor, the torque reading of an electric motor, or pixel values from a camera is not inherently meaningful. It becomes information only once it is evaluated in relation to a model or purpose.
In practice, information is gained by knowledge of context and respective processing of the primary signal, i.e., raw data from objects such as sensors. Hence, information stands between data and knowledge. Formally, we define the following:

Definition 4.1 (Signal processing).
Reconsider a system

$$\begin{aligned} \dot{\mathbf{x}}(t) &= f(\mathbf{x}(t), \mathbf{u}(t), t), \quad \mathbf{x}(t_0) = \mathbf{x}_0 \\ \mathbf{y}(t) &= h(\mathbf{x}(t), \mathbf{u}(t), t) \end{aligned} \tag{1.1}$$

with output $\mathbf{y}$. Let $\mathcal{D}$ denote the *set of output data*. Then any map/method $m : \mathcal{D} \to \mathcal{I}$ is called *signal processing* with $\mathcal{I}$ denoting the *information set*.

Examples of data streams in the realms of vehicles and production are given in Figure 4.2 and 4.3, respectively. As we can see, the means of transfer and the context of data streams in an application may vary significantly.

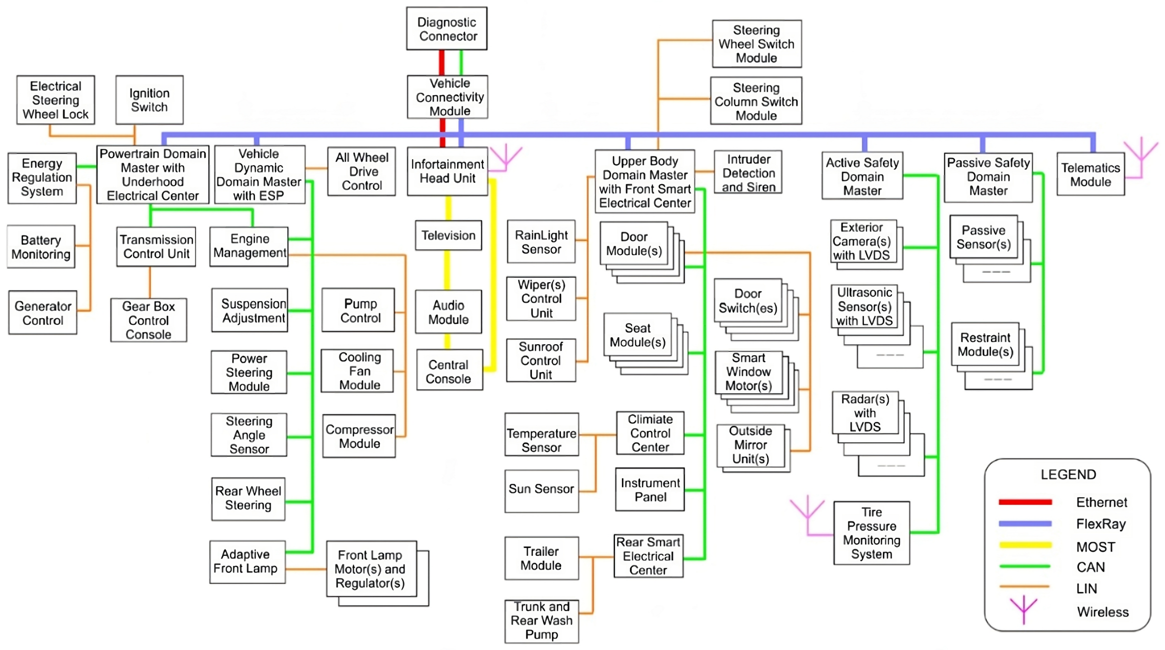


Figure 4.2.: Exemplary topology of in-vehicle network topology in 2016 [32]

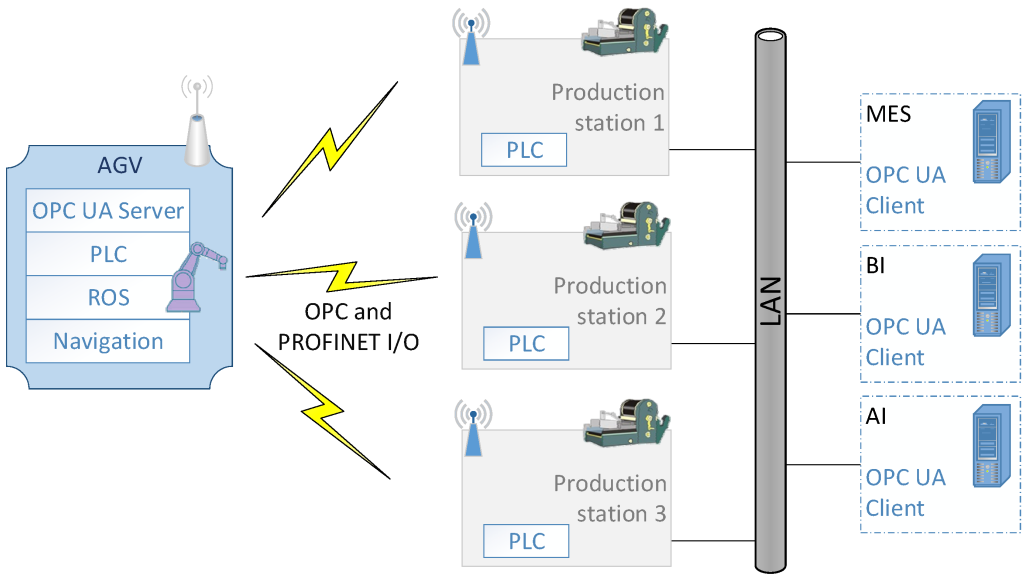


Figure 4.3.: Examples of data streams for a production system

In applications, signal processing consists of several steps and depends on what information is used. The basic step is always to measure with some sensing equipment. Using mechanics only, such a signal can already be used to take action on the process. This was the standard approach in the first industrial revolution, e.g., via the steam engine governor; see again Chapter 1. The mechanical signal can also be transduced, i.e., pushed into an electric form. This allows for monitoring processes, as in the second industrial revolution. The third step is to apply a converter and shift the signal into digital form, which already allows to apply logic units to it (third industrial revolution). Last, coding reveals state information, which can be included in higher computing tasks.
The following Table 4.1 captures these steps.

Table 4.1.: Steps of signal processing

Attribute	**Method**	**Attribute**
State of system/process	Sensing	Measurement signal (arbitrary form)
Measurement signal	Transducing	Electric continuous signal
Electric continuous signal	Converting	Electric digital form
Electric digital signal	Coding	System/process state information

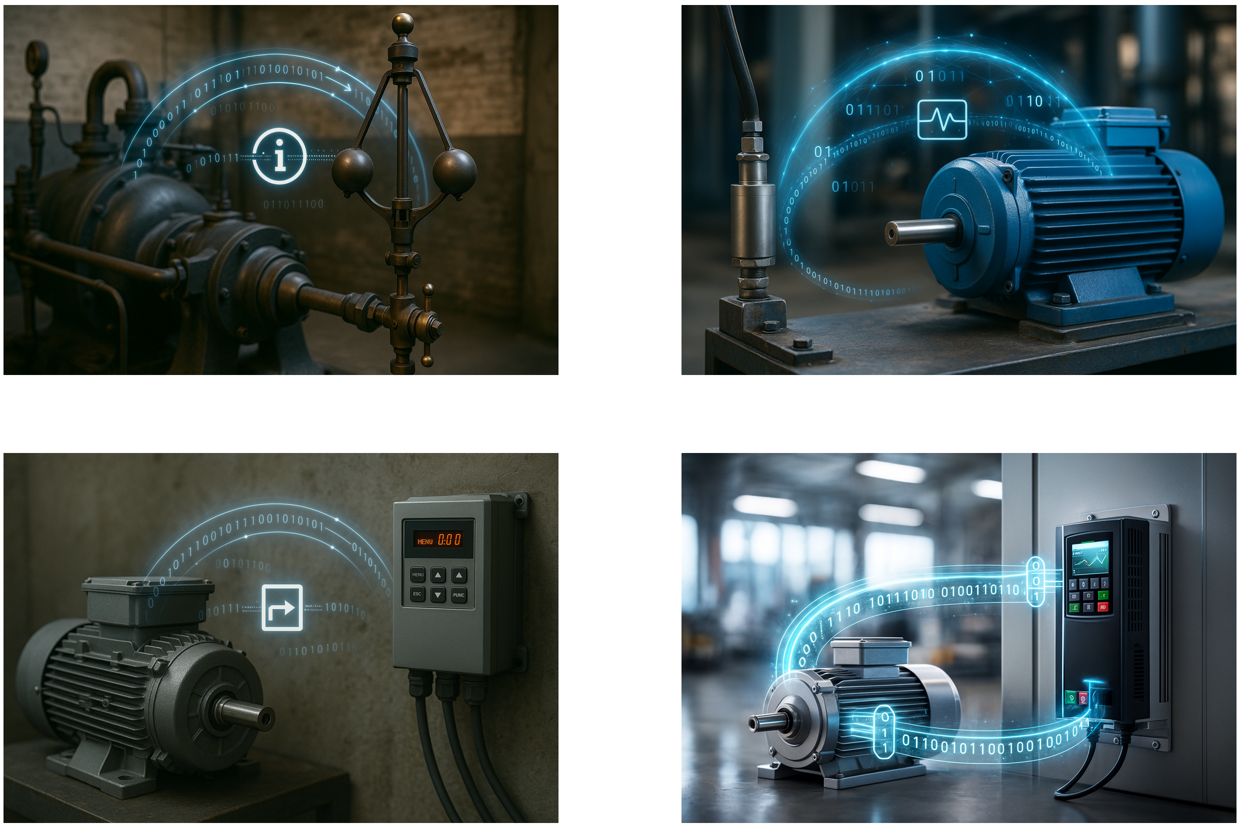


Figure 4.4.: Illustration of sensing, transducing, converting and coding, generated with chatgpt.com

Task 4.2
An example of such signal processing is a gyro, a device nowadays found in almost any smartphone or motion platform. Assign the object's amplifier, stator, A/D converter, and Kalman filter in the correct sequence for a signal processing unit.

Solution to Task 4.2: The gyro sensor is built up as stator $\rightarrow$ amplifier $\rightarrow$ A/D converter $\rightarrow$ Kalman filter.

Remark 4.3
Note that the reverse path from gathered system/process information to influencing the system/process is equivalent but inverted in its sequence.

In today's practice, various sensors can be found, which may to some extend already integrate all of the steps above. In particular, we distinguish between

- a sensor, which is a basic device that measures a physical quantity and outputs a raw signal without any internal processing. It performs only the transduction step by converting a physical phenomenon into an electrical signal.

- an integrated sensor, which includes not only the sensing element but also basic signal conditioning electronics – typically amplification, filtering, linearization, and sometimes normalization. It still does not „understand“ the process, but it provides a clean, ready-to-use signal, and
- an intelligent sensor, which integrates the sensing element, signal conditioning, and embedded microelectronics for data processing, communication, and self-diagnostics. As such, it generates information, not just conditioned signals.

These three types are schematically sketched in Figures 4.5–4.7.

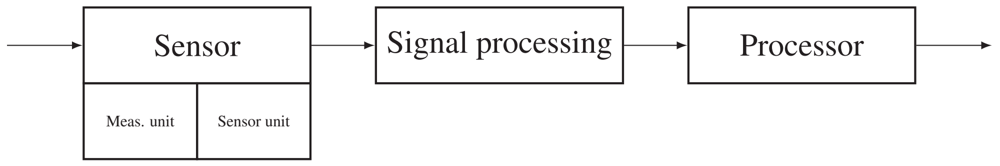


Figure 4.5.: Standard sensor configuration

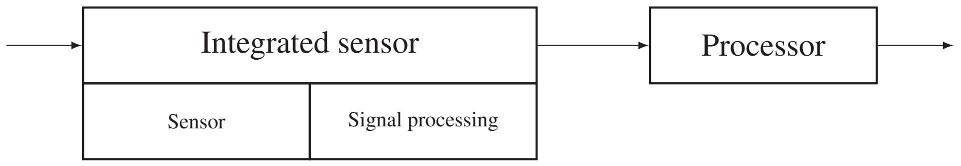


Figure 4.6.: Integrated sensor

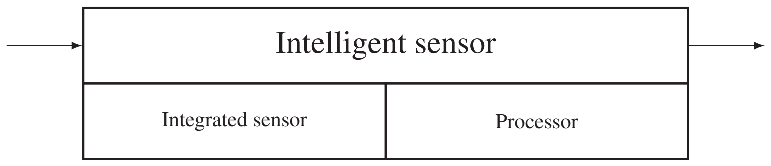


Figure 4.7.: Intelligent sensor

The choice of sensor depends on how deeply the underlying process is to be integrated into higher-level management or supervisory systems. If a control system is intended to access not only a particular machine or subsystem but a broader segment, then sufficiently rich and structured information about the respective object must be available to the control layer. This requirement naturally leads to the question of how such information can be transmitted, both reliably and in a form that remains meaningful across different levels of the automation hierarchy.

4.2. Transmission networks and communication structures

In the previous Chapter 2, we have already seen how edges in a network can connect objects. While in that chapter, we thought of networks more in a physical way, i.e., goods to be transported or tasks to be done, networks for information focus on the transmission of data/information.

Remark 4.4
Data and information are regularly used equivalently, although they are not. This confusion stems from the problem that information gained from a single system/process, i.e., on a low level, may only be primary data for the integration of systems/processes, i.e., on a higher level. Hence, the same object is information for a control engineer on an operational (machine) level but data for a planner on a tactical (layout) level or information for a planner but data for a data scientist on a strategic (leading) level.

By definition of a network (cf. Definition 2.5), the connection of objects by edges depends on the application to be modeled. There exist two trivial cases as shown in Figure 4.8:

1. If the network exhibits zero edges, then the network is completely disconnected.
2. If for each pair of distinct objects $\mathcal{O}_j$, $\mathcal{O}_k$ with $\mathcal{O}_j \neq \mathcal{O}_k$ there exists an edge $e = (\mathcal{O}_j, \mathcal{O}_k)$, then the network is fully connected.

Figure 4.8.: Examples of fully connected and completely disconnected networks

In between these two extreme cases, a wide variety of possible topologies exist, which exhibit certain advantages and disadvantages when it comes to the transmission of data/information. Prototypes of such topologies are given in Figure 4.9.
Regarding practically relevant topologies, we obtain the (dis-)advantages shown in Table 4.2.

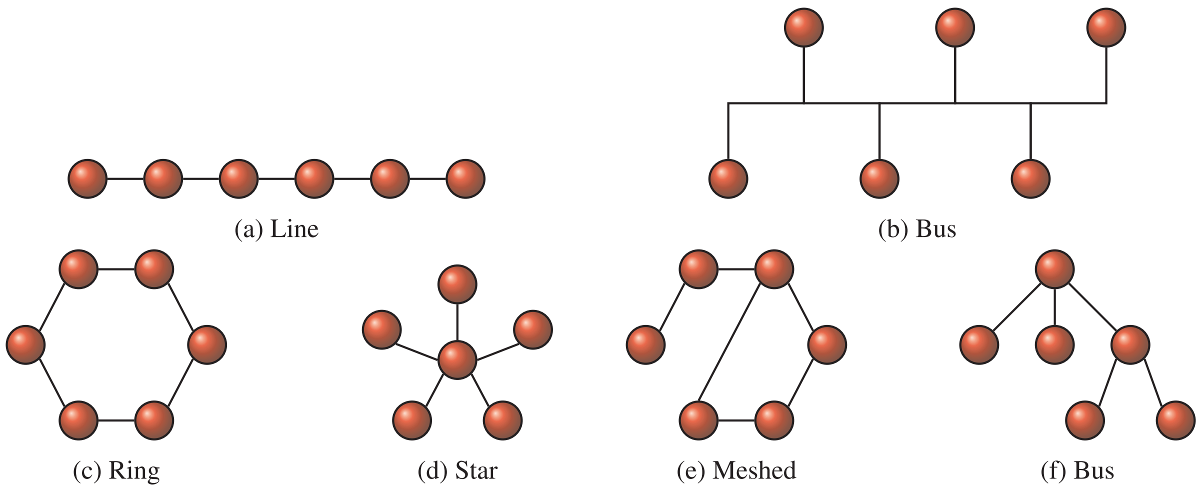


Figure 4.9.: Different network topologies

Table 4.2.: Properties of communication topologies

Topology	**Advantages**	**Disadvantages**
Bus	• Scalable • No additional network components	• Not secure
Star	• Easily reducable • Secure • No routing	• Large number of lines • SPOF
Ring	• Easily extendable • No additional network components	• Not secure • Each object is SPOF • Long transmission times

For the physical realization of such networks, a wide range of tethered and wireless options are available. These are referred to as media.

Definition 4.5 (Network medium).
Consider a network $(\mathcal{O}, \mathcal{E})$. Then we call the physical possibility to realize edges $e \in \mathcal{E}$ *network medium* $\mathcal{N}_M$.

Here, we consider only the different transmission media and refrain from discussing the numerous protocol level or technology specific options that may operate on top of them. Nevertheless, even at this fundamental level, the choice of medium has a significant impact on system behavior, as each exhibits distinct physical characteristics, performance limits, and susceptibility to disturbances. These differences directly influence achievable bandwidth, latency, robustness, installation complexity, and cost. For instance, wireless media may offer high flexibility but reduced reliability in harsh industrial environments, whereas optical fibers provide excellent immunity to electromagnetic interference at the expense of higher installation effort. The following Table 4.3 summarizes some basic properties and relates each medium to typical communication topologies, highlighting how certain media naturally support or constrain the organization of networks within automated systems.

Table 4.3.: Properties of communication media

Medium	**Advantages**	**Disadvantages**	**Topology**	**Example**
UTP[1]	Low costs Simple patching	Not secure Low throughput Low range	Bus, ring, star	ISDN Ethernet
S/STP[2]	Simple patching Medium throughput	High costs Not secure	Star	Ethernet
Coaxial	High throughput High range	Difficult patching Medium costs	Bus	BNC (Car) F-plug (TV)
Fiber optic	High throughput High range	High costs Difficult patching Not bus-compatible	Star, ring	LC (Router) Soundbar
Wireless	No cables No patching	High costs Not secure	Meshed, Star	LTE/5G 802.11/15/16

As can be seen from the different media and topologies available, the technologies applied in the examples shown in Table 4.3 can diverge quickly. To prevent this, the so-called ISO/OSI standard

[1]Unshielded twisted pair such as CAT3 to CAT5
[2]Screend shielded twisted pair such as CAT6/7

has been developed to ensure that all technologies and media can be used interchangeably at the cost of additional components

Remark 4.6
Additional hardware components may be required if the selected media and topologies are not inherently compatible. For instance, transitioning from a wired bus segment to an optical fiber backbone necessitates the use of repeaters or media converters to bridge the physical-layer incompatibility. Similarly, integrating a wireless mesh segment into an otherwise line- or star-structured wired network may require gateways that translate the physical medium, addressing and timing behavior characteristic of the respective topology. These conversion points introduce additional costs, latency and potential failure modes, meaning that the choice of media and network topology cannot be considered in isolation. Instead, they must be evaluated together to ensure seamless communication across heterogeneous system components.

4.3. Open system interconnection

The OSI (Open Systems Interconnection) reference model was developed to standardize communication structures and behaviors across heterogeneous systems, thereby enabling interoperability among devices, software components, and network technologies. This standardization provides significant advantages to both users and developers/manufacturers of communication components: users benefit from predictable behavior and interchangeability, while developers can design modular communication interfaces without needing to tailor each solution to every possible application scenario. However, as we learned in Chapter 3, any form of standardization inherently introduces constraints and may reduce system optimality. The OSI model is no exception. In the context of information and communication, this degradation typically stems from the fact that the reference model prescribes application-independent communication patterns. These patterns are designed to be generic and broadly applicable, which means they may not perfectly fit the specific timing, bandwidth, or reliability requirements of a particular industrial or embedded application. As a result, additional overhead—such as header structures, protocol negotiation, encapsulation, or abstraction layers—may be introduced solely to comply with a standardized structure. The OSI model divides communication into seven distinct layers [13], each responsible for a specific set of functions within the overall communication process. This layered approach supports modularization and separation of concerns: higher layers can operate independently of the underlying physical technology, while lower layers remain unaffected by application logic. Figure 4.10 illustrates how the layers build on one another, forming a conceptual pipeline that governs the transformation of data into signals, and ultimately into meaningful information exchanged between systems.

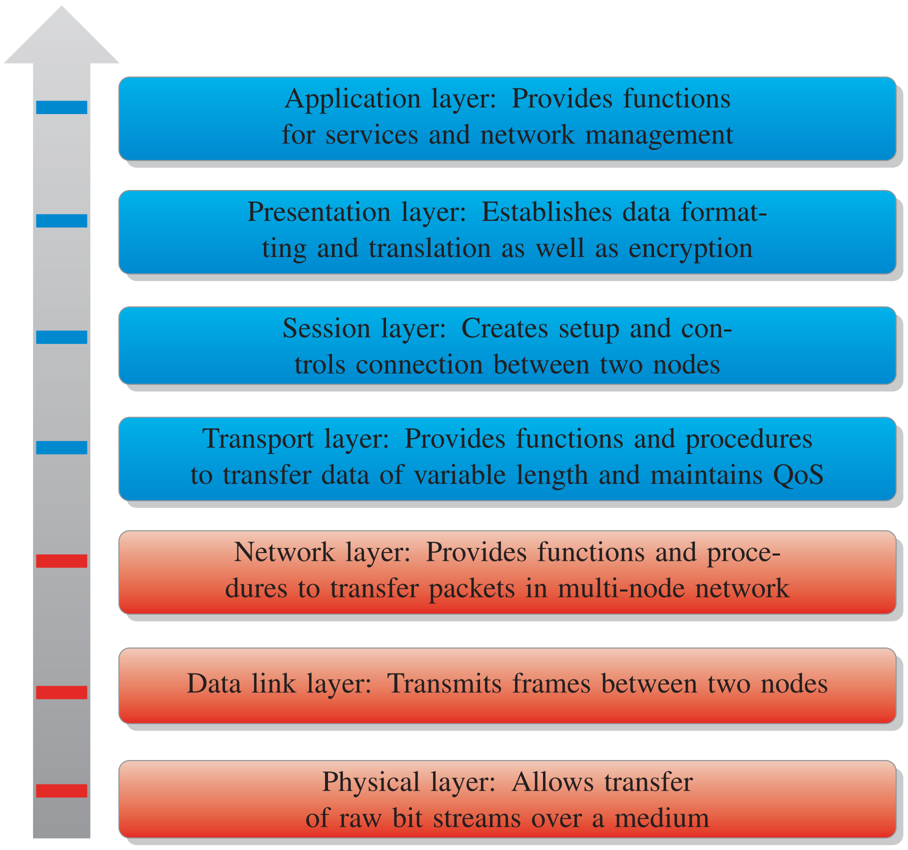

Application layer: Provides functions for services and network management
Presentation layer: Establishes data formatting and translation as well as encryption
Session layer: Creates setup and controls connection between two nodes
Transport layer: Provides functions and procedures to transfer data of variable length and maintains QoS
Network layer: Provides functions and procedures to transfer packets in multi-node network
Data link layer: Transmits frames between two nodes
Physical layer: Allows transfer of raw bit streams over a medium

Figure 4.10.: Structure of the standardization process

As such, the OSI reference model is a network that upholds certain rules, which pay special attention to the allowable links between objects.

Definition 4.7 (Layer).
Consider a network $(\mathcal{O},\mathcal{E})$ with clustering S. Then we call the elements $\mathcal{L}_j \in S$ *layers* if

$$\mathcal{L}_j \cap \mathcal{L}_k = \emptyset \tag{4.1}$$

$$\forall \mathcal{O}_{j_1} \in \mathcal{L}_k, k < \#S : \exists \mathcal{O}_{j_2} \in \mathcal{L}_{k+1} \wedge (\mathcal{O}_{j_1},\mathcal{O}_{j_2}) \in \mathcal{E} \tag{4.2}$$

$$\forall \mathcal{O}_{j_1} \in \mathcal{L}_k, k > 1 : \exists \mathcal{O}_{j_2} \in \mathcal{L}_{k-1} \wedge (\mathcal{O}_{j_1},\mathcal{O}_{j_2}) \in \mathcal{E} \tag{4.3}$$

$$\forall \mathcal{O}_{j_1} \in \mathcal{L}_k : \nexists (\mathcal{O}_{j_1},\mathcal{O}_{j_2}) \in \mathcal{E} : \mathcal{O}_{j_2} \notin \mathcal{L}_{k-1} \cup \mathcal{L}_{k+1} \tag{4.4}$$

$$\forall \mathcal{O}_{j_1} \in \mathcal{L}_k : \nexists \mathcal{O}_{j_2} \in \mathcal{L}_k \cap (\mathcal{O}_{j_1},\mathcal{O}_{j_2}) \in \mathcal{E}. \tag{4.5}$$

In this sense, the OSI reference model is not a physical network but a rule-based structural network. Its operation depends on these rules in the same way technical systems depend on physical laws or organizational networks depend on coordination rules.

Note that the definition of layers within the OSI reference model is formulated from the perspective of a single, distinct node, whether this exists as a physical device, such as a sensor, controller or server, or as a virtualized entity, such as a software-defined communication endpoint. The model describes how this node structures, processes and exchanges information through its internal layer stack. Each layer is therefore a function of the node rather than a property of the network as a whole. In a broader network-theoretic sense, however, such nodes are still treated as interacting objects. The OSI layers define how each object behaves internally and prepares data for interaction. However, the network emerges only from the interplay of many such objects connected through communication links. Therefore, although the OSI model is node-centric, communication is a system-level phenomenon shaped by the collective behavior of all interconnected objects. As such, the OSI network links different nodes.
Hence, due to the layer structure in each of the nodes, there actually exist two networks. Abstracting the latter, we obtain the following:

Definition 4.8 (OSI network).
Consider a network $(\mathcal{O}, \mathcal{E})$. Then we call it *OSI network* if

- each object $\mathcal{O}_j \in \mathcal{O}$ is itself a network consisting of 7 layers in the sense of Definition 4.7 and Figure 4.10,
- for each edge $(\mathcal{O}_j, \mathcal{O}_k) \in \mathcal{E}$ there exist edges $(\mathcal{O}_{j_z}, \mathcal{O}_{k_z})$, $z = 1, \ldots, 7$ connecting the layers, and
- for each communication process including more than two objects, only layers 1-3 exist for intermediate objects.

Remark 4.9
The last condition in Definition 4.8 explicitly deals with the communication process, not the network. In fact, Definition 4.8 shows us two networks, one for communication and one within the objects for the respective layers.

For developers of programs, the first three layers are relevant as they can be adapted to application specifications, whereas the lower four layers address the transport of information. In terms of connectivity, the first five levels consider multi-hop and multi-destination problems, whereas the lower two layers implement point-to-point communication, cf. Table 4.4.
The top four layers within the OSI reference are so-called host layers and deal with issues of the host. On the other hand, the lower three layers are called media layers and focus on the specifics of the respective media. In Figure 4.10, these layers are indicated by the different colors.

Table 4.4.: Layers in the OSI reference - **P**lease **D**o **N**ot **T**hrow **S**alami **P**izza **A**way

Layer	Classification	Connection	Examples
Application			DHCP, FTP, DNS
Presentation	Application oriented		MQTT, LDAP,
Session		Multihop	HTTP/S
Transport			TCP, UDP
Network			IP, IPX
Data link	Transport oriented		802.11, MAC
Physical		Point-2-point	ARCNET

Here, we want to highlight some of the key properties of the respective layers:

1. The physical layer provides „physical“ means for communication, that is, mechanical, electrical, optical, or other components to transmit bits. Hence, only 0 and 1 can be transmitted, which induces some kind of coding.
2. On the data link layer, we are already dealing with frames, which are sets of bits combined with being transmitted. The aim is to guarantee the most error-free transmission and ability to access the medium. To this end, checks for successful transmission are made on the receiving end, yet no corrections are considered.
3. The network layer deals with packets of frames and aims to choose the right way to the destination while already avoiding congestion on the medium. To this end, this layer also considers different ways to reach the target, which is called routing.
4. The transport layer slices the information to be transmitted into packets and serves as an internal router by assigning a port for communication. By assigning a port, this layer also converts data into technology dependent formats for the underlying layers.
5. The session layer organizes and synchronizes information transmission between two systems. As such, it supervises each connection and validates access permission.
6. The presentation layer converts the system depending representation of data into an independent form, i.e., allows for syntactic abstraction but also for encryption or compression.

7. Last, the application layer provides access to functions of lower layers and serves as a mediator.

While the OSI reference model specifies how communication tasks are to be hierarchically decomposed within each individual unit, it remains intentionally agnostic regarding the actual mechanisms through which communication channels are accessed. In other words, the model defines what each layer is responsible for, but not how these responsibilities must be implemented in practice. Consequently, the OSI model does not specify how communication channels shall be accessed physically, logically, or timewise.
Hence, the OSI model tells us the „what“ but not the „how“.
These aspects are left to concrete communication technologies and protocols, which may interpret or implement the OSI layers in markedly different ways.

4.4. Network access

We now want to transmit signals between objects in accordance with the OSI reference model. To this end, we can draw upon the various communication structures and physical transmission media discussed in Section 4.2 and combine them with the layered communication functions introduced in Section 4.3. Our objective is to derive rules for how data is sent over a transmission medium, specifically addressing whether, when, and in what order multiple objects may access that medium. In other words, we are concerned not only with how information is processed inside each node (as described by the OSI layers) but also with how multiple nodes share or compete for use of the same physical channel.
Here, we talk about network access in the following sense:

Definition 4.10 (Network access).
Consider a network medium $\mathcal{N}_M$. We call any set of rules to access the network medium *network access* $\mathcal{N}_A$.

As outlined before, these rules may depend on physics, logic and time, i.e. „how“ is „which“ object allowed to access the network medium „when“.
To exemplify the latter, we consider a simple two-sensor setup connected by a shared bus cable (e.g., a CAN bus). Both sensors may wish to transmit temperature readings to a controller, but the rules of the bus dictate that only one message can occupy the line at any moment. The access mechanism is a time based procedure, which ensures that collisions are avoided by predefined time slots for medium access.

If we use the same network setting together with a wireless network medium, then the objects may rely on carrier sensing to detect whether the channel is free before transmission.
Considering these two cases, we see that the first one is time triggered while the second one applied logic rules.
Generalizing the latter example, we have to distinguish two forms of network access, that is, deterministic and probabilistic methods.

Definition 4.11 (Deterministic network access).
Consider a network $(\mathcal{O}, \mathcal{E})$ together a network medium $\mathcal{N}_M$ and network access $\mathcal{N}_A$. If

- $\mathcal{N}_A$ is time-based or fixed for all $\mathcal{O}$ and
- the transmission and response time are bounded and known,

then we call $\mathcal{N}_A$ a deterministic network access method.

Definition 4.12 (Probabilistic network access).
Consider a network $(\mathcal{O}, \mathcal{E})$ together a network medium $\mathcal{N}_M$ and network access $\mathcal{N}_A$. If $\mathcal{N}_A$ is event-based, then we call it a *probabilistic network access method*.

Why are these two forms relevant: The deterministic network access is only applicable if the objects themselves and the network medium can be characterized (more or less) completely. Such an implementation makes sense if the overall process will not be changed and no modifications will be made. This is typically the case for production lines. Examples of such network access methods are given in Table 4.5.

Table 4.5.: Deterministic network access methods

Method	Advantages	Disadvantages
Master/Slave	Master periodically asks slaves whether they want to access the network medium.	
	Simple organization	Max. latency proportional to the number of objects
	Guaranteed response time	communication if master fails

Continued on next page

Table 4.5 – continued from previous page

Method	Advantages	Disadvantages
Token passing	Token is handed over to net object in a ring topology.	
	High performance capability	Long delays in case of errors Required surveillance of token
TDMA[3]	Each object has one/multiple time slots for network access in a cyclic period.	
	Short and constant cycle time Low overhead	Required synchronization of time

In contrast to deterministic methods, probabilistic methods are more suitable for changing environments, a changing number of objects, and modifications to applications which result in different response times. The idea of these methods is to constantly listen to the network medium and start network access if the medium is available. While being very flexible, such methods also show structural shortcomings. In the following Table 4.6, we characterize a few of these methods.

Table 4.6.: Probabilistic network access methods

Method	Advantages	Disadvantages
CSMA[4]	Easily extendable No coordination required	Not realtime capable Requires permanent listening
CSMA-CD[5]	Detect collisions of packages via data matching, resends after object specific waiting time.	
	Short latency in low load case	Long waiting times in high load case

Continued on next page

[3]Time Division Multiple Access
[4]Carrier Sense Multiple Access
[5]Carrier Sense Multiple Access - Collision Detection

Table 4.6 – continued from previous page

Method	Advantages	Disadvantages
CSMA-CA[6]	Avoids collisions by priority rules.	
	Allocation of time slots possible	Required fixed rules

Apart from the network access, also the format of transmission itself is of interest. As the format differs depending on the network medium and the used protocol, we will not go into detail but focus on the big picture along the OSI reference model. Figure 4.11 provides an overview of how data is split and appended by network and transmission information to be ready for transmission/-transport using the OSI layers.

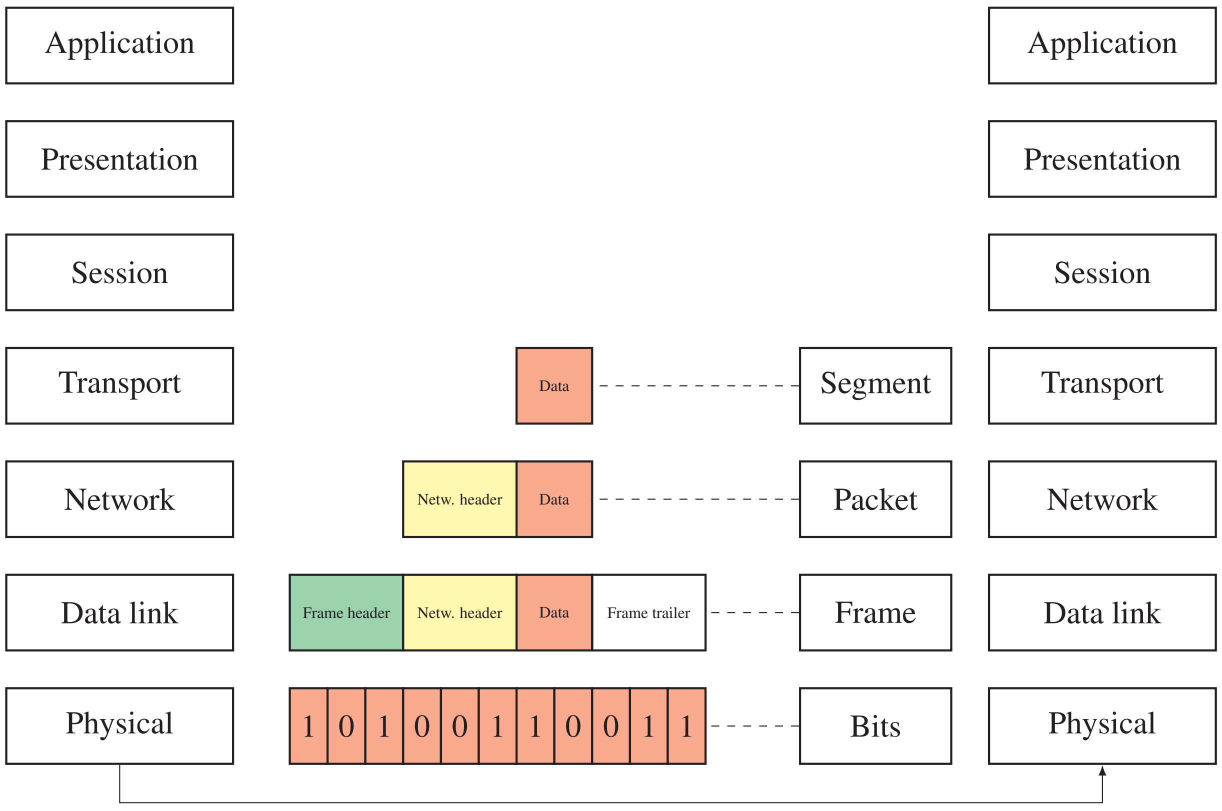


Figure 4.11.: Transforming network communication to bits

As an illustrative example, Figure 4.12 depicts the structure of a typical CAN bus frame together with its corresponding physical-bit representation using pulse-width–modulated (PWM) signal-ing.

[6]Carrier Sense Multiple Access - Collision Avoidance

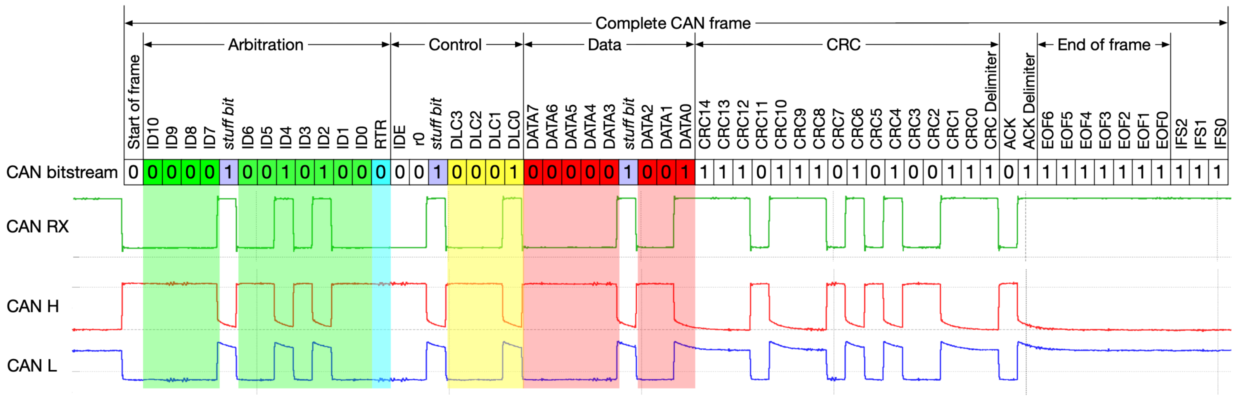


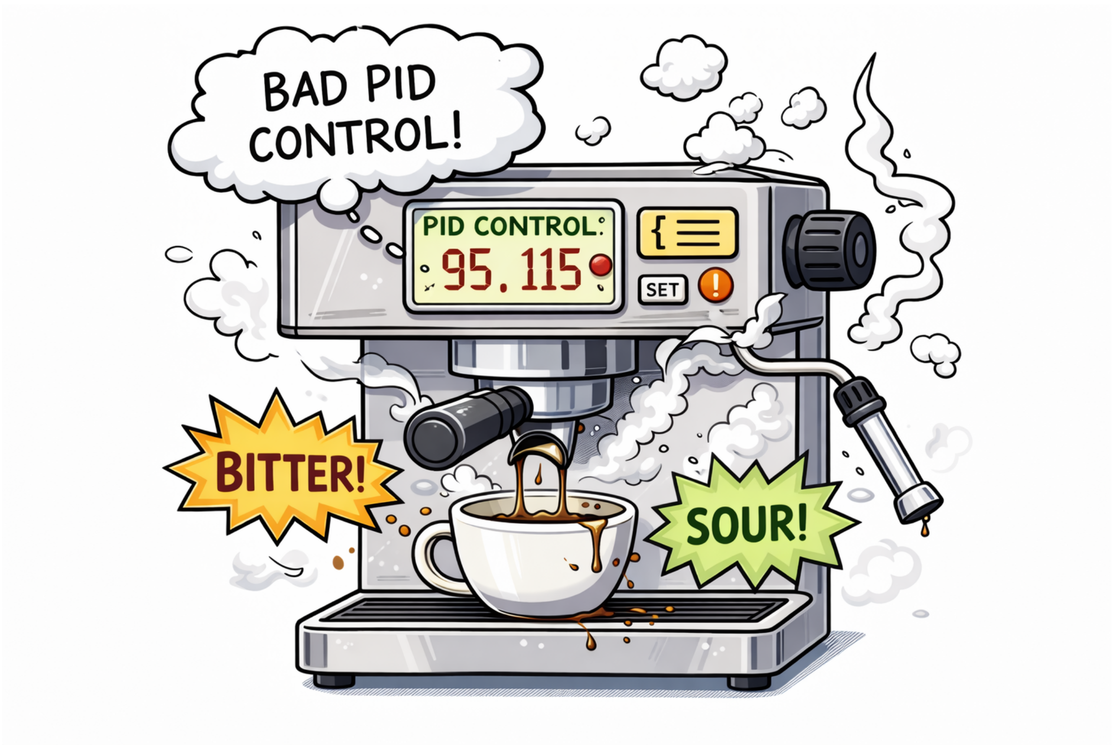
Figure 4.12.: CAN bus frame [31]

The upper part of the figure highlights the logical organization of a CAN message—such as the start-of-frame bit, arbitration field, control field, data field, CRC sequence, and end-of-frame bits—while the lower part shows how these logical bits are translated into the actual dominant and recessive signal levels transmitted on the bus. This juxtaposition makes it clear how the abstract, protocol-level frame format is encoded on the physical layer and how bit timing, synchronization, and arbitration arise directly from the underlying signal pattern.

Summing up, communication provides the structural foundation upon which we may now build control in automated systems. While the previous section focused on how information is generated, transmitted, and organized across heterogeneous devices and networks, the subsequent control part addresses how this information is transformed into decisions and actions. In other words, communication enables the availability and integrity of data, whereas control relies on this data to interpret system behavior, compute control laws, and influence the physical process. Only through the seamless interaction of both domains can an automated system function reliably, adapt to disturbances, and achieve its intended performance. The following Chapter 5 therefore builds directly on the communication structures established so far and shows how they serve as the input channels for monitoring, feedback, and decision-making.

CHAPTER 5

CONTROL



Generated with chatgpt.com

Feedback is the breakfast of champions.

Ken Blanchard

In Chapter 1, the fundamental concepts of feedforward and feedback control were introduced. Although conceptually simple, these two complementary control paradigms constitute the core building blocks of automation systems. Feedforward control enables proactive compensation of measurable disturbances and reference changes, while feedback control ensures robustness against uncertainties, unmodeled dynamics, and external perturbations by continuously correcting deviations from the desired system behavior.

Together, feedforward and feedback control establish the functional link between lower-level control loops and higher-level automation tasks, such as system integration, optimization, supervision, and planning, cf. Figure 5.1. In this sense, classical control loops form the technical foundation upon which higher abstraction layers are built ranging from coordination and scheduling up to normalization and production planning. Without this reliable and well-understood control backbone, advanced automation functionalities at the planning and optimization levels cannot be implemented in a stable and reproducible manner.

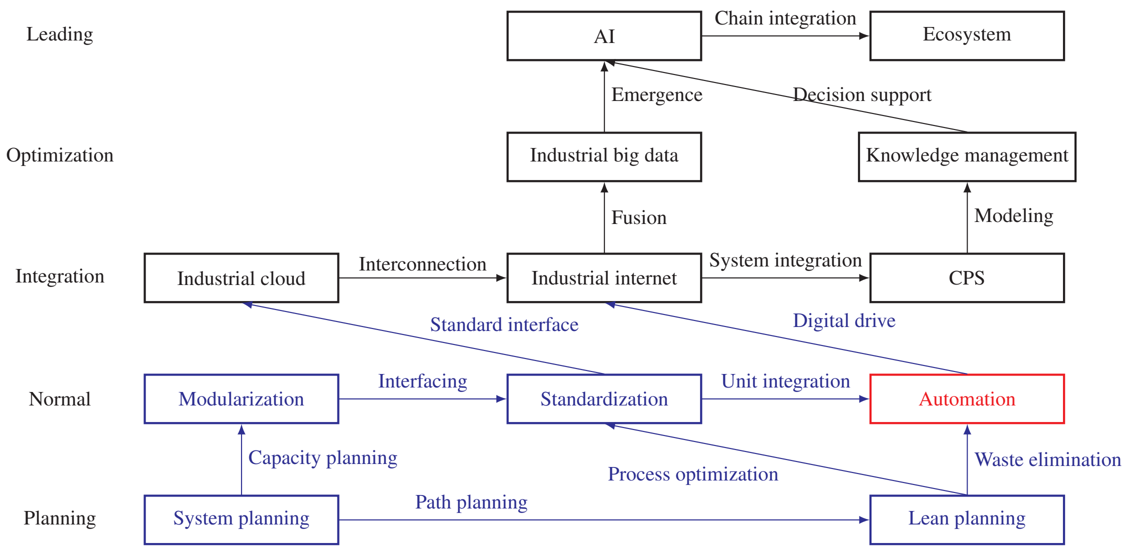


Figure 5.1.: Information in the integrated implementation path of automation

In industrial production systems, the interplay of feedforward and feedback control is a standard design principle. For example, in high-speed manufacturing systems such as printing machines, packaging lines, and CNC machining centres, feedforward control is often derived from reference trajectories or process models in order to achieve high dynamic accuracy. Feedback loops then compensate for wear, friction, modelling uncertainties, and external disturbances. This layered control structure is essential in industrial practice, as it ensures reproducibility, scalability, and maintainability across different machines and production sites.

A closely related design philosophy can be observed in automotive systems, where feedforward and feedback control form the basis of almost all mechatronic subsystems. In engine and powertrain control, feedforward maps obtained from calibration data are used to compute actuator commands—such as fuel injection, air mass, and ignition timing—based on current operating conditions. Feedback loops complement these maps by ensuring compliance with emission, torque, and drivability targets under varying environmental conditions and ageing effects.

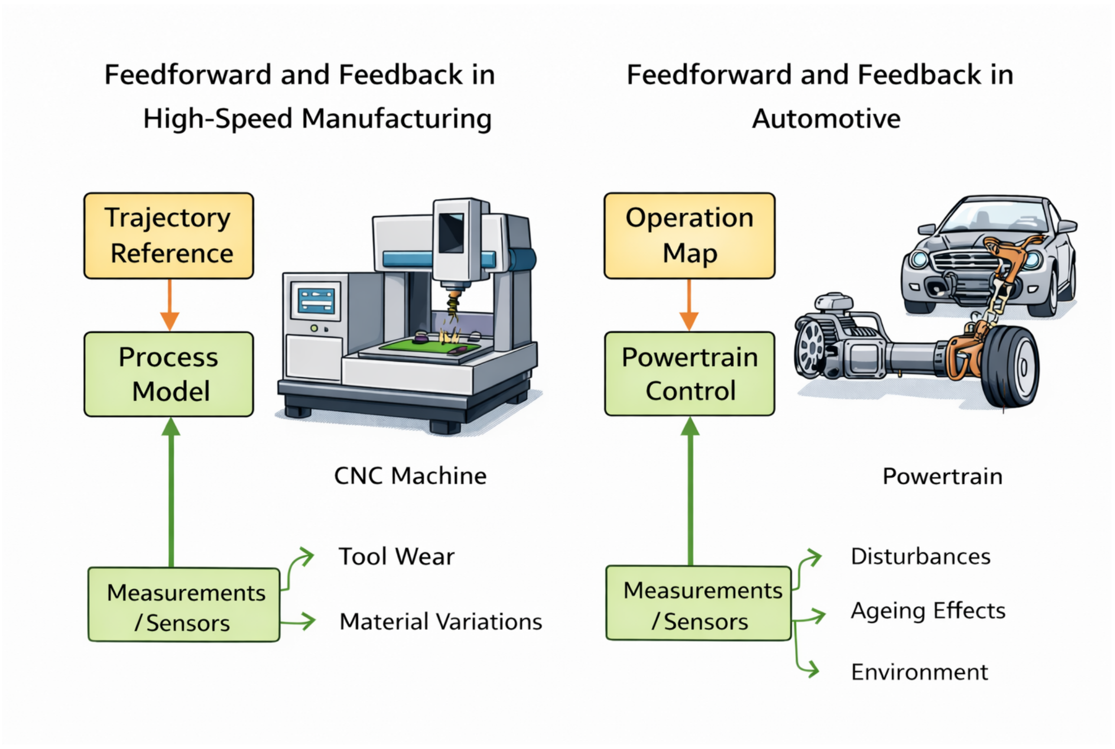


Figure 5.2.: Illustration of feedforward and feedback in production and automotive, generated with chatgpt.com

Against this background, this chapter focuses on the feedback part of such control architectures, using the widely applied PID controller as a representative example. While the PID controller is inherently limited to single-input–single-output (SISO) systems, it provides a transparent and practically relevant starting point for understanding feedback mechanisms. To overcome its structural limitations and to reflect industrial practice more accurately, the basic PID loop is subsequently extended by precontrol and prefilter structures, which allow additional feedforward information to be integrated into the control loop. However, real industrial and automotive systems rarely consist of a single controlled variable and a single actuator. Instead, multiple inputs must be computed and multiple outputs must be regulated simultaneously. Such systems are referred to as multi-input–multi-output (MIMO) systems. The transition from SISO control concepts to MIMO systems therefore forms a natural next step and serves as an entry point for linking higher-level automation tasks with machine-level control structures, which will be addressed in the subsequent chapters.

5.1. Feedforward and feedback control

In Definitions 1.10 and 1.11, two fundamental control concepts — feedforward and feedback — were introduced. Although closely related, these concepts differ fundamentally in how control actions are generated.

Roughly speaking, feedforward control prescribes control actions to be applied at specified time instants, independent of the measured system response. Consequently, a feedforward strategy can be interpreted as a mapping from time to control input. In contrast, feedback control determines control actions based on the observed behavior of the system. In this case, the control input is generated as a function of the measured output (or state), yielding a mapping from output to input. To formalize these concepts, we recall the definition of a nonlinear control system given in state-space form:

$$\begin{aligned}\dot{\mathbf{x}}(t) &= f(\mathbf{x}(t), \mathbf{u}(t), t), \quad \mathbf{x}(t_0) = \mathbf{x}_0 \\ \mathbf{y}(t) &= h(\mathbf{x}(t), \mathbf{u}(t), t).\end{aligned} \tag{1.1}$$

In order to derive a meaningful feedback law for such a system, a desired system behavior must be specified. This is achieved by introducing a reference signal, which encodes the intended evolution of the system output over time.

Definition 5.1 (Reference).
Given a system (1.1), we call $\mathbf{w} : \mathcal{T} \to \mathcal{Y}$ a *reference*.

Figures 5.3 and 5.4 illustrate the corresponding information paths for feedforward and feedback control, respectively, highlighting the fundamental difference in how control actions are generated and applied.

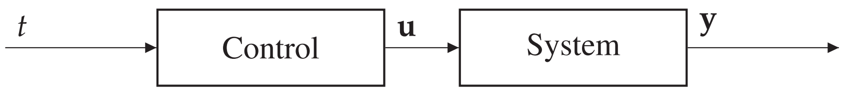


Figure 5.3.: Simple feed forward

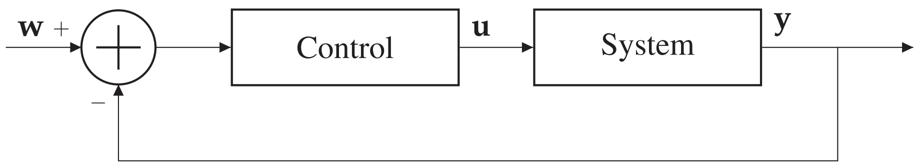


Figure 5.4.: Simple feedback

Remark 5.2
Note that both Figure 5.3 and 5.4 resemble a network consisting of objects (system, control) and edges (time input, reference input, control input/output and measurement output).

In practical control applications, feedforward and feedback are rarely used in isolation, but are typically combined in a complementary manner. While both concepts aim at influencing system behavior toward a desired objective, they exhibit fundamentally different properties with respect to robustness, anticipation, and disturbance rejection. These differences become particularly evident when considering intuitive real-world examples.

Task 5.3
Discuss the concepts of feed forward and feedback in driving situations. Highlight disadvantages.

Solution to Task 5.3:

- A feedforward strategy in driving is comparable to studying a road map or navigation plan in advance and deriving a route from a starting point to a destination. The resulting plan specifies where and when certain actions should be taken, independent of the actual driving situation encountered later.
- A feedback strategy corresponds to steering, accelerating, and braking a vehicle based on its current state, as obtained from measurements and perception of the surrounding environment (e.g., vehicle speed, lane position, other traffic participants).

The downsides for this particular example are:

- A purely feedforward approach lacks information about unforeseen disturbances and dynamic changes. For instance, a preplanned route does not account for sudden braking of a vehicle ahead, traffic jams, or road closures.
- A purely feedback-based approach reacts only to the current situation and does not exploit available preview information. As a consequence, the driver may follow a locally reasonable path but unknowingly enter an unfavorable or even dead-end route.

On a more generic view, the concepts exhibit the advantages and disadvantages given in Table 5.1.

Figure 5.5.: Illustration of shortcomings of pure feedforward and pure feedback in driving, generated with chatgpt.com

Table 5.1.: Advantages and disadvantages of feed forward and feedback

Method	**Advantages**	**Disadvantages**
Feed forward	Integrates external knowledge Plans ahead via simulation Addresses KPIs	May show infeasible solutions Requires good model
Feedback	Guarantees stable behavior Reacts to circumstances Addresses system properties	May end in unwanted operating points Requires theoretical insight

While feedback control is conceptually more demanding than feedforward control, it constitutes the fundamental enabling mechanism for higher-level automation tasks. In particular, feedback provides robustness with respect to disturbances, uncertainties, and modeling errors, which is indispensable for reliable operation in real-world systems. For this reason, we first focus on feedback control and its properties before integrating feedforward structures into such loops in Section 5.2.

5.1.1. PID control

Among feedback controllers, PID control — proportional–integral–derivative control — is by far the most widely used approach in industrial practice. Its popularity stems from its simple structure, intuitive tuning parameters, and its effectiveness across a wide range of applications. Formally, a PID controller is defined as follows.

Definition 5.4 (PID control).
Consider a system (1.1). Then we call $\mathbf{u} : \mathcal{Y} \to \mathcal{U}$ given by

$$\mathbf{u}(t) = K_P \cdot (\mathbf{w}(t) - \mathbf{y}(t)) + \int_{t_0}^{t} K_I \cdot (\mathbf{w}(\tau) - \mathbf{y}(\tau)) \; d\tau + \frac{\partial}{\partial t} (K_D \cdot (\mathbf{w}(t) - \mathbf{y}(t))) \tag{5.1}$$

PID controller. The parameters K_P, K_I and K_D are called proportional, integral and derivative gains.

Remark 5.5
Note that typically, a PID controller is designed in frequency domain using Laplacian and Inverse Laplacian transform.

Link: For further details regarding PID control, we refer to the lecture Control Engineering 1 (Regelungstechnik 1).

Like any feedback controller, a PID controller is introduced to enforce desirable dynamical properties around specific system configurations, commonly referred to as operating points. Operating points represent steady-state conditions at which a system or process is intended to operate, such as temperatures at which a 3D printer is most efficient or vehicle speeds that minimize fuel consumption.

Definition 5.6 (Operating point).
Consider system (1.1). Then the pairs $(\mathbf{x}^\star, \mathbf{u}^\star)$ satisfying

$$f(\mathbf{x}^\star, \mathbf{u}^\star) = 0 \tag{5.2}$$

are called *operating points* of the system. If (5.2) holds true for any $\mathbf{u}^\star$, then the operating point is called strong or robust operating point.

Remark 5.7
We like to point out that in industrial practice we typically linearize a system around an operating point and design the feedback for this linearization.

Link: For the overall nonlinear system, different types of controller designs are available. For details, we refer to the lecture Control Engineering 2 & 3.

5.1.2. Stability

As discussed earlier, the primary system property of interest in control design is stability. Informally, stability means that if the system is initially at an operating point, it remains there, and if it is perturbed away from this point, the system trajectories remain close to — or converge back to — the operating point.
Building upon Definition 5.6, we can introduce different notions of stability, such as stability and asymptotic stability, as well as related concepts like robustness and controllability. The precise interpretation of these properties depends on whether the control input $\mathbf{u}$ is regarded as an actively designed control signal or as an external disturbance acting on the system.

Definition 5.8 (Stability and Controllability).
For a system (1.1) we call $\mathbf{x}^\star$

- *strongly* or *robustly stable* operating point if, for each $\varepsilon > 0$, there exists a real number $\delta = \delta(\varepsilon) > 0$ such that for all $\mathbf{u}$ we have

$$\|\mathbf{x}_0 - \mathbf{x}^\star\| \leq \delta \implies \|\mathbf{x}(t) - \mathbf{x}^\star\| \leq \varepsilon \qquad \forall t \geq 0 \tag{5.3}$$

- *strongly* or *robustly asymptotically stable* operating point if it is stable and there exists a positive real constant r such that for all $\mathbf{u}$

$$\lim_{t \to \infty} \|\mathbf{x}(t) - \mathbf{x}^\star\| = 0 \tag{5.4}$$

  holds for all $\mathbf{x}_0$ satisfying $\|\mathbf{x}_0 - \mathbf{x}^\star\| \leq r$. If additionally r can be chosen arbitrary large, then $\mathbf{x}^\star$ is called *globally strongly* or *robustly asymptotically stable*.

- *weakly stable* or *controllable* operating point if, for each $\varepsilon > 0$, there exists a real number $\delta = \delta(\varepsilon) > 0$ such that for each $\mathbf{x}_0$ there exists a control $\mathbf{u}$ guaranteeing

$$\|\mathbf{x}_0 - \mathbf{x}^\star\| \leq \delta \implies \|\mathbf{x}(t) - \mathbf{x}^\star\| \leq \varepsilon \qquad \forall t \geq 0. \tag{5.5}$$

- *weakly asymptotically stable* or *asymptotically controllable* operating point if there exists a control $\mathbf{u}$ depending on $\mathbf{x}_0$ such that (5.5) holds and there exists a positive constant r such that

$$\lim_{t\to\infty} \|\mathbf{x}(t) - \mathbf{x}^\star\| = 0 \qquad \forall \|\mathbf{x}_0 - \mathbf{x}^\star\| \leq r. \tag{5.6}$$

If additionally r can be chosen arbitrary large, then $\mathbf{x}^\star$ is called *globally asymptotically stable*.

Task 5.9
Draw solutions of systems for each of the cases in Definition 5.8.

Remark 5.10 (BIBO stability)
In some books, the concept of strong/robust stability is also termed BIBO (bounded input bounded output) stability.

The standard example in control is the inverted pendulum on a cart, cf. Figure 5.6.

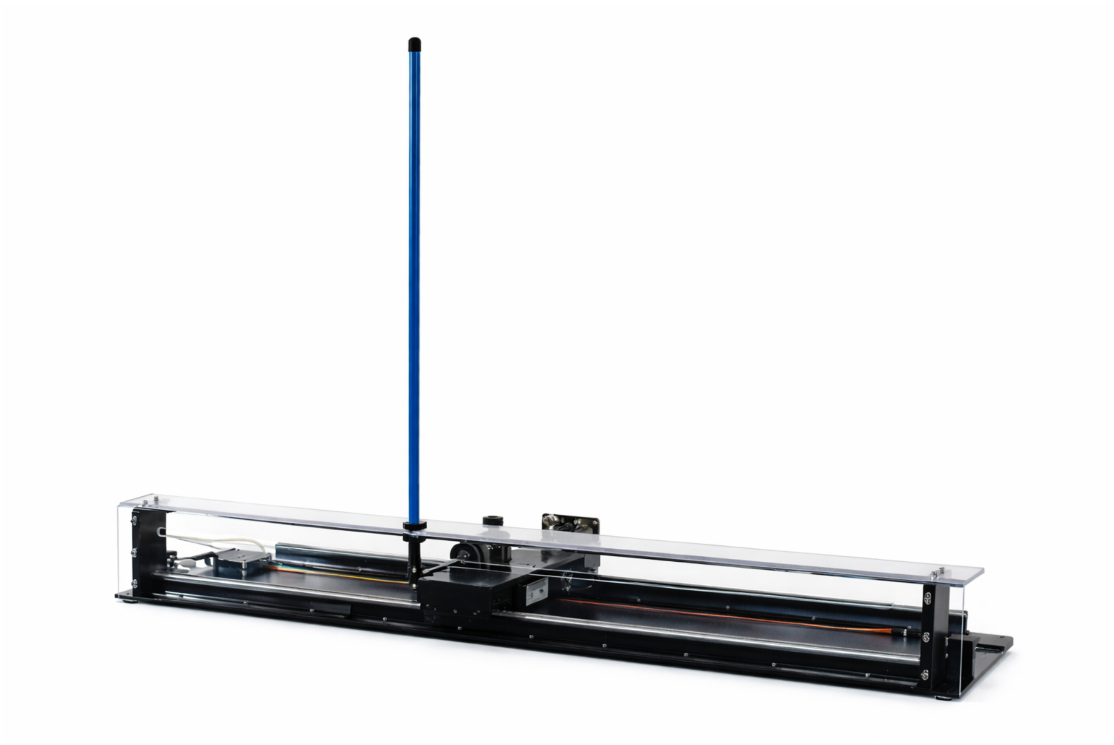


Figure 5.6.: Illustration of inverted pendulum on a cart, generated with chatgpt.com

To understand strong asymptotic stability ,we consider the downward position of the rod. Due to friction, the rod will converge into this state eventually. More generally speaking, the stability

property is inherent to the system dynamics and therefore does not depend on the specific choice of the control input. In such cases, the role of the control law is not to ensure stability itself, but rather to shape the transient behavior and steady state performance of the system, for instance with respect to convergence speed, overshoot, energy consumption, or robustness margins.
In contrast, the concept of weak stability is reflected by the upward position. Here, stability of the operating point is no longer guaranteed by the system alone, but instead must be actively enforced by a suitably designed control law. This naturally raises the question of how to construct control laws that render a given operating point weakly stable or asymptotically stable, and, beyond mere stability, how to assess and compare the quality of different control strategies in terms of performance, robustness, and implementation constraints.

> **Link:** Methods on how computing and characterizing control laws stand at the core of the lectures *Control Engineering 1 & 2*.

5.2. Prefilter and precontrol

As previously outlined, the path of automation (Figure 5.1) requires the systematic integration of higher-level planning functions with lower-level control mechanisms. Conceptually, plans generated at higher levels of the automation hierarchy naturally take the form of feedforward actions, whereas execution and regulation at the machine level rely on feedback control to cope with disturbances and uncertainties.
Below, we consider fundamental control structures that enable the integration of feedforward and feedback within a single control loop. These structures offer a systematic approach to incorporating anticipatory information from higher-level planning while maintaining the robustness and stability guarantees provided by feedback control. Two common realizations of this integration in practice are precontrol and prefilter structures, illustrated in Figures 5.7 and 5.8, respectively.

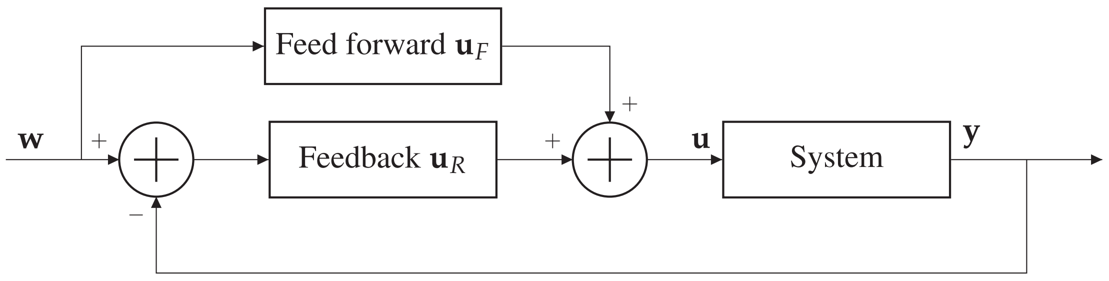


Figure 5.7.: Structure of a precontrol

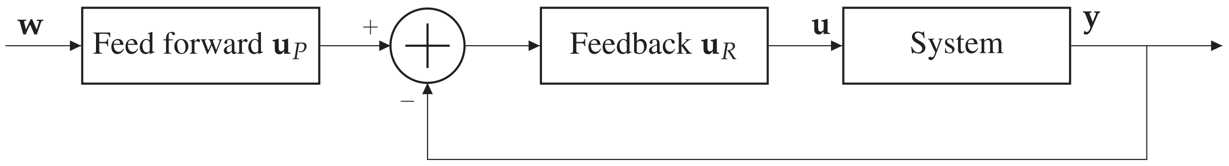


Figure 5.8.: Structure of a prefilter

Remark 5.11
It is important to emphasize that, in the considered structures, the feedforward path does not interfere with the feedback loop. As a consequence, the stability properties of the closed-loop system are preserved. The presented architectures therefore allow us to systematically combine the complementary strengths of feedforward and feedback control: anticipatory action on the one hand and robustness against disturbances and uncertainties on the other.

The precontrol structure shown in Figure 5.7 and the prefilter structure shown in Figure 5.8 both enable a simultaneous treatment of reference tracking via feedback and anticipation via feedforward. However, they differ fundamentally in where the feedforward action is introduced and how it is designed.
The design of a precontrol proceeds in two steps:

Algorithm 5.12 (Design precontrol)
Consider a control system as illustrated in Figure 5.7.

(1) Design the feed forward $\mathbf{u}_F(t)$ to be (approximately) the inverse of the open loop system (1.1).

(2) Design the feedback $\mathbf{u}_R(\mathbf{y})$ such that desired properties such as stability are guaranteed as good as possible.

In contrast, the design of a prefilter follows the reverse logic:

Algorithm 5.13 (Design prefilter)
Consider a control system as illustrated in Figure 5.8.

(1) Design the feedback $\mathbf{u}_R(\mathbf{y})$ such that desired properties such as stability are guaranteed as good as possible.

(2) Design the feed forward $\mathbf{u}_P(t)$ to be (approximately) the inverse of the closed loop system (1.1) with feedback $\mathbf{u}_R(\mathbf{y})$.

Remark 5.14
From a functional perspective, the two approaches operate in different domains: the prefilter is a mapping $\mathbf{u}_P : \mathcal{T} \to \mathcal{Y}$*, acting directly on the reference signal, whereas the precontrol defines a mapping* $\mathbf{u}_F : \mathcal{T} \to \mathcal{U}$*, acting on the system input. Consequently, prefilter and precontrol address distinct aspects of the control problem and are suited to different modeling assumptions and implementation constraints.*

It is worth noting that precontrol, in contrast to prefiltering, is designed independently of the closed-loop dynamics. Consequently, once a precontrol has been constructed, it does not need to be modified if the feedback controller is subsequently retuned or replaced. In this sense, precontrol provides a higher degree of modularity with respect to feedback redesign.
Despite this structural difference, the two approaches are fundamentally equivalent in terms of their achievable closed-loop behavior, as stated in the following result.

Theorem 5.15 (Equivalency precontrol and prefilter).
Suppose a system (1.1) *to be given. Then*

- *for every precontrol defined via Algorithm 5.12 there exists a prefilter, which exhibits an identical closed loop, and*
- *for every prefilter defined using Algorithm 5.13 there exists a precontrol, which exhibits an identical closed loop.*

The integration of feedforward and feedback discussed above directly reflects the connection between the integration, optimization, and leading levels and the normalization and planning levels in the path of automation. At this stage, however, both precontrol and prefilter structures are restricted to single-input–single-output (SISO) systems.
Recalling the general structure of an automated system or process introduced in Chapter 1 (cf. Figure 5.9), we have now established all essential concepts required to analyze and design control architectures for systems with a single input and a single output. While this setting provides a transparent and instructive foundation, it is inherently insufficient for most real-world automation applications.
In manufacturing systems, such as CNC machining centers or printing and packaging lines, multiple actuators (e.g., drives, spindles, and feeders) simultaneously influence multiple quality-relevant outputs (e.g., position accuracy, surface quality, and throughput). Strong coupling effects, timing constraints, and shared resources make it impossible to treat each input–output pair independently without sacrificing performance or stability.

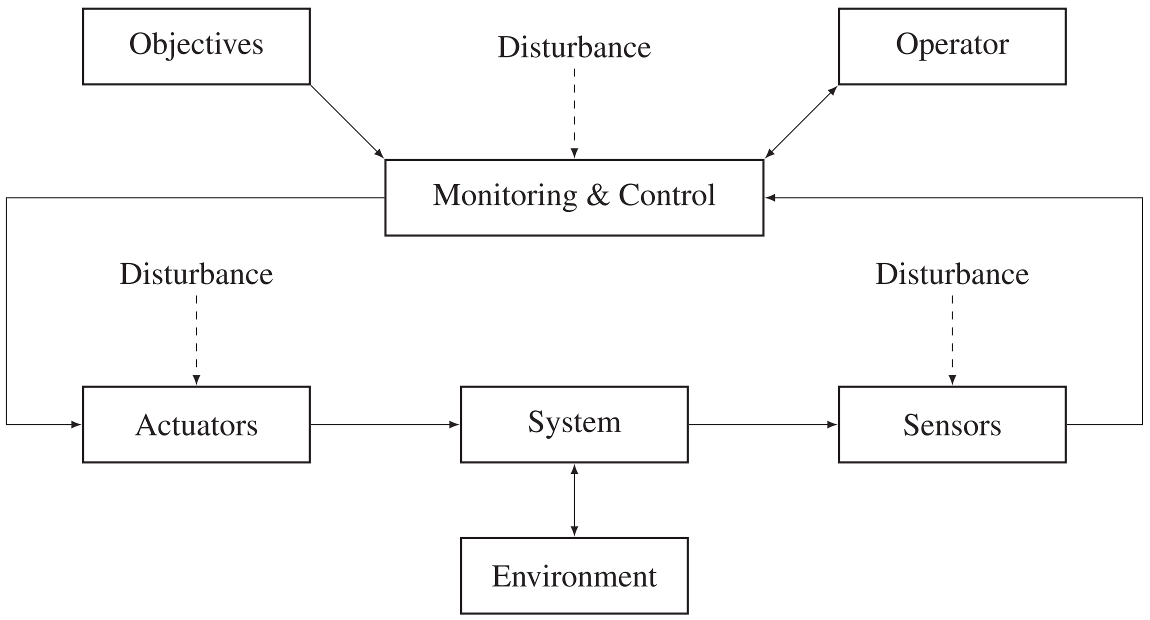


Figure 5.9.: Structure and process of automated systems

Similarly, in automotive systems, even seemingly simple functions such as longitudinal vehicle control involve multiple interacting inputs and outputs, i.e. engine torque, braking force, and transmission state jointly affect vehicle speed, fuel consumption, and emissions. In such settings, a purely SISO based control perspective neglects essential cross-couplings and interaction effects. These examples illustrate that modern automation systems are intrinsically multi-input–multi-output (MIMO) in nature. Consequently, extending the concepts developed so far to multi-variable systems is not merely a theoretical generalization, but a practical necessity for achieving robust, high-performance control in real-world applications.

At the same time, the overarching objective of automation engineering — and in particular of paradigms such as Industrie 4.0 and the Industrial Internet of Things (IIoT) — is to enable operation at a potentially global and interconnected scale. This shift from isolated machines to networked systems introduces additional challenges related to communication, coordination, and scalability. To address these aspects, the following Chapter 6 investigates how large scale automation systems can be structured and interconnected. Building on this architectural foundation, we subsequently turn to centralized, decentralized, and distributed control and optimization formulations in Chapter 7, where coordination, optimization, and leading tasks across interconnected subsystems are systematically addressed.

Part II.

Integration, optimization and leading

CHAPTER 6 NETWORKING

Generated with chatgpt.com

Everything that can be automated will be automated.

Robert Cannon

In the preceding chapters, control concepts were developed primarily in the context of single-input–single-output (SISO) systems. This setting allowed us to introduce feedback, feedforward and their integration in a transparent manner. In particular, PID control exploits a clear separation between time-domain error dynamics and a single control channel, enabling straightforward interpretation and tuning.
However, most real-world automation systems cannot be adequately described by a single manipulated variable and a single measured output. Instead, multiple actuators simultaneously influence multiple outputs, often in a coupled and time-varying manner. Such systems are referred to as multi-input–multi-output (MIMO) systems and form the dominant class of systems in practical applications. While we technically already considered such a setting in our planning Chapter 2 using networks, we left out the components communication and information as well as control and coordination. Considering our path of automation, Figure 6.1 shows the components we consider in this chapter.

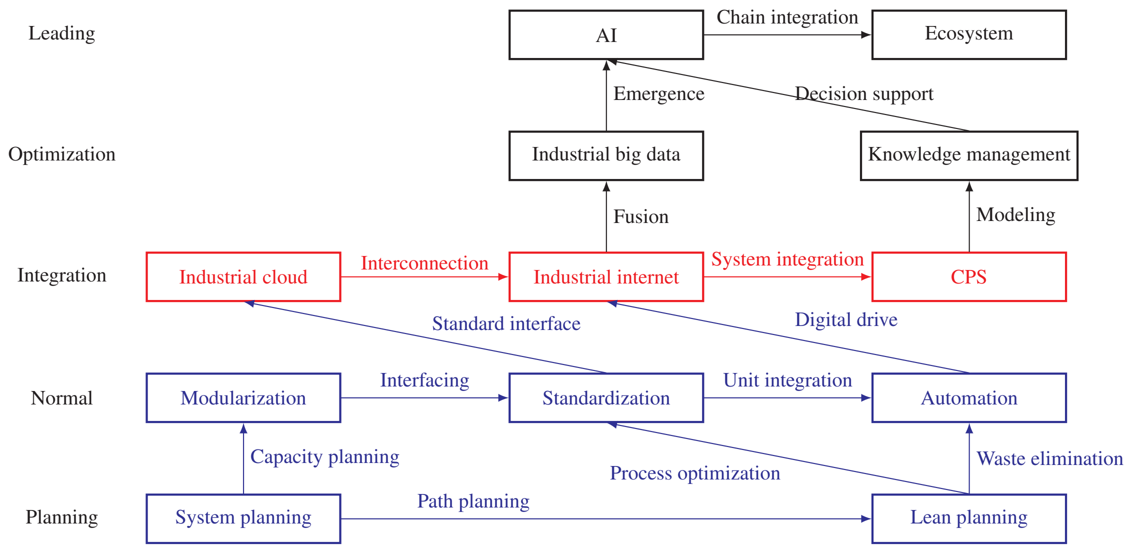


Figure 6.1.: Information in the integrated implementation path of automation

A classical approach to coping with complexity in interconnected systems is decoupling. Analogous to PID control in the SISO case—where proportional, integral, and derivative actions separate different temporal effects—MIMO systems are often simplified by attempting to decouple them along different dimensions:

- state-based decoupling, separating subsystems by buffers or intermediate storage,
- time-based decoupling, separating processes through scheduling, batching, or waiting times,
- control-based decoupling, separating control loops by treating interactions as disturbances.

While these approaches outlined in the following Section 6.1 are simple to apply and have been widely used in industrial practice, they come at a cost. In manufacturing systems, such as CNC machining centers, printing presses, or packaging lines, we observe strong coupling between drives, spindles, thermal effects, and quality metrics. Treating such systems as collections of loosely coupled SISO loops neglects essential interaction effects and quickly reaches its limits. Similarly, in automotive systems, longitudinal and lateral dynamics, powertrain, thermal management, and energy storage are tightly interconnected, making decoupling-based simplifications increasingly inadequate.
MIMO control seeks to explicitly account for interactions, coupling effects, and constraints, rather than masking them through decoupling. This enables coordinated control actions, improved performance, and more efficient use of resources. While the standard PID idea can be pursued in the case of two inputs and two outputs, such as procedure falls short in terms of scalability as we will see in Section 6.1.3.
To overcome this deficiency, we first introduce the concept of digital twins in Section 6.2, i.e., digital representations of physical systems that combine first-principles models with measurement data and operational knowledge. In the context of MIMO systems, digital twins allow interaction effects to be analyzed, monitored, and exploited rather than suppressed. The digital twin concept is a key enabler of cyber-physical systems (CPS), in which computation, communication, and physical processes are tightly integrated. We capture CPS architectures in Section 6.3 to allow continuous interaction between the physical system and its digital counterpart, enabling real-time monitoring, prediction, and control without being bound by the physical limitations of the plant or vehicle, particularly with respect to computing power and information availability.
In the extension of cyber-physical systems, we discuss the ideas of Industrie 4.0 and the Industrial Internet of Things (IIoT) in Section 6.4. These move beyond individual machines or vehicles toward large-scale, networked systems. Industrial cloud infrastructures provide services such as Infrastructure as a Service (IaaS) and Software as a Service (SaaS), enabling scalable computation, centralized data access, fleet-level analytics, and coordinated optimization across distributed assets. This leads us to the service of an Industrial Internet forming the conceptual and technological basis for addressing large-scale optimization and leading problems.

6.1. Decoupling

To understand why emerging technologies such as the industrial cloud, the industrial internet, and cyber-physical systems can have a profound impact on automation engineering, it is first necessary to examine how interconnection and system integration are traditionally addressed. A fundamental principle in engineering is to keep systems as simple as possible. When applied

to interconnection and integration, this principle typically leads to an attempt to separate and decompose systems or processes such that couplings between them are minimized or ideally eliminated.
While this approach has proven effective for many classical automation tasks, it increasingly reaches its limits as systems grow in complexity, speed, and scale. To formalize this discussion, we introduce the following definition.

Definition 6.1 (MIMO system).
Consider a system

$$\begin{aligned} \dot{\mathbf{x}}(t) &= f(\mathbf{x}(t), \mathbf{u}(t), t), \quad \mathbf{x}(t_0) = \mathbf{x}_0 \\ \mathbf{y}(t) &= h(\mathbf{x}(t), \mathbf{u}(t), t). \end{aligned} \tag{1.1}$$

Then we call the system to be *multi input multi output* (MIMO) if

$$\left\| \frac{\partial^2 \mathbf{y}}{\partial a_j \partial a_k} \right\| \geq \theta \tag{6.1}$$

for some $a_j, a_k \in \{y_1, \ldots, y_{n_y}, u_1, \ldots, u_{n_u}, t\}$ with $\theta \in \mathbb{R}^+$.

The threshold parameter θ indicates the degree of coupling of the inputs and outputs. Hence, if there exists no pair a_j, a_k for which (6.1) holds true, then all components can be treated independently. Still one has to keep in mind that the threshold allows for certain small impacts, hence the control of the independently designed systems should be capable to suppress disturbances emanating from other systems.
Before coming to approaches for decoupling, we want to highlight that coupling occur on various levels and variables. Examples for strongly coupled outputs are

- pressure and temperature for steamers,
- position, velocity and force for robot arms, or
- produced parts in serial production lines.

Coupled inputs may be

- position control of multiple drives for robots,
- roll and yaw angle control for flying curves with an aircraft, or
- temperature, pH measurement and biomass distribution for bio reactors.

In the remainder of this section, we briefly discuss classical approaches that aim to decouple systems and processes by exploiting state variables, time separation, or control structure design, and we highlight the fundamental limitations associated with each of these approaches.

6.1.1. Decoupling of states and outputs

The most fundamental and widely used decoupling approach is state-based decoupling. The underlying idea is to partition the system into subsystems by assigning distinct state variables to each part. If two states are dynamically coupled, this coupling becomes observable only through their influence on the corresponding outputs y_j and y_k. Consequently, state-based decoupling aims to isolate state dynamics such that interactions between subsystems are minimized or treated as disturbances at the output level.

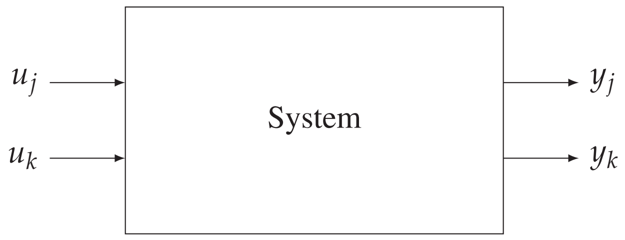


Figure 6.2.: MIMO system with two inputs and two outputs

From (6.1), we directly obtain

$$\left\| \frac{\partial^2 \mathbf{y}}{\partial y_j \partial y_k} \right\| \geq \theta. \tag{6.2}$$

The idea to decouple these two outputs is to add a new state

$$x_{jk}(t) := y_j(t) - y_k(t)$$

to the system eliminating the coupling. While introducing the new state solves the pure coupling problem at least mathematically, the downside is that the new state sums up all coupling problems. In industrial production systems, this principle is commonly applied by introducing buffers or intermediate storage. For example, in serial manufacturing lines, individual machines are often decoupled by workpiece buffers. The internal states of one machine — such as tool wear, spindle dynamics, or local thermal conditions — do not directly affect the next machine, but only become visible through output variables such as part availability or dimensional deviations. While this approach simplifies local control design, it introduces inventories and reduces overall system responsiveness.

A similar strategy can be found in automotive systems, where subsystems are often decoupled at the state level to simplify control design. For instance, in vehicle powertrain architectures, engine, transmission, and drivetrain dynamics are frequently treated as separate subsystems, with their interaction limited to torque or speed signals. Internal states such as combustion dynam-

ics or clutch temperature are hidden from neighboring controllers and only manifest themselves through measurable outputs. While this facilitates modular development, it can lead to suboptimal coordination and delayed reactions under highly dynamic driving conditions.
Note that such an approach is good to compensate for fluctuations, yet not for system/process instabilities. Hence, it can only be used to a certain extend, i.e. until the capacity is reached.

Table 6.1.: Advantages and disadvantages of output decoupling

Advantage		Disadvantage	
✓	Simple separation	✗	Induces storage
✓	Small additional load	✗	Unable to treat instabilities

6.1.2. Decoupling of time

An alternative to decoupling via states or outputs is decoupling in time. Similar to the state-based case, coupling between outputs can be characterized by

$$\left\| \frac{\partial^2 \mathbf{y}}{\partial y_j \partial y_k} \right\| \geq \theta,$$

indicating a significant interaction between output components. Instead of structurally separating subsystems, time decoupling exploits temporal separation by executing coupled actions at different time instants. Formally, this is achieved by redefining the output signals via

$$\begin{aligned} y_j(t) &= \eta(t) \cdot y_j(t) \\ y_k(t) &= \eta(t-\delta) \cdot y_j(t) \end{aligned}$$

for some temporal shift $\delta > 0$. Here, $\eta(\cdot)$ is the Heaviside function

$$\eta(t) = \begin{cases} 0, & t < 0 \\ \text{undefined}, & t = 0 \\ 1, & t > 0 \end{cases}$$

representing a unit jump. While mathematically eliminating the coupling at a specific point in time t, the result of a time decoupling is that system/process steps are executed consecutively.
In practical terms, time decoupling transforms a parallel control problem into a sequence of tasks, thereby eliminating simultaneous interactions. This approach is widely used in industrial pro-

duction systems. For example, in robotic manufacturing cells, operations such as positioning, welding, and inspection are often executed sequentially rather than concurrently. By separating motion and processing phases in time, complex multi-variable interactions are avoided, simplifying control design at the expense of cycle time.
A similar strategy appears in automotive systems, particularly in vehicle testing, calibration, and operation management. For instance, certain engine diagnostics or actuator calibrations are performed only during specific operating phases (e.g., idle or steady cruising), deliberately avoiding periods of high dynamic coupling such as acceleration or gear shifts. Likewise, advanced driver assistance systems may temporarily deactivate or sequence control functions to prevent interference between longitudinal and lateral control tasks.
While time decoupling is effective in reducing control complexity, it introduces inherent limitations. Most notably, it leads to time delays and reduced throughput, as coupled actions cannot be executed in parallel. Moreover, changes in process timing often require program-level adaptations, making such approaches less flexible in highly dynamic or reconfigurable systems. These characteristics are summarized in Table 6.2.

Table 6.2.: Advantages and disadvantages of time decoupling

Advantage	**Disadvantage**
✓ Simple separation	✗ Induces time delays
✓ No additional load	✗ Requires program adaptation

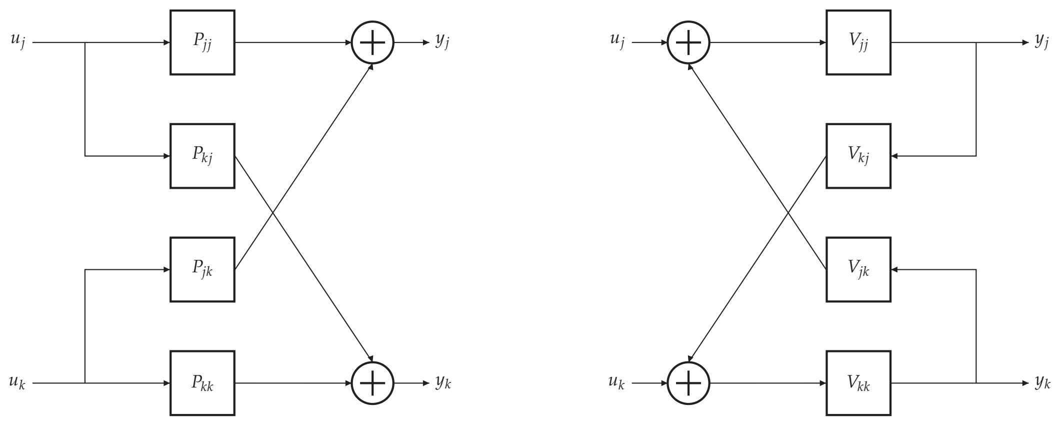
6.1.3. Decoupling of control

In contrast to state- or time-based decoupling, decoupling at the control level is significantly more involved. Rather than separating subsystems structurally or temporally, control-based decoupling explicitly compensates for interactions between inputs and outputs through appropriately designed control laws.
Focusing on the setting illustrated in Figure 6.2, we observe that coupling between inputs and outputs can occur in two fundamentally different ways. These two interaction patterns correspond to feedforward-like and feedback-like structures and are formalized by the following canonical representations.

Definition 6.2 (P canonical structure).
Consider a system with two inputs and two outputs. If the coupling of inputs to outputs exhibits

a feed forward structure as shown in Figure 6.3a where P_{jk} is the transfer function of input u_k to output y_j for all j and k, then we call it *P canonically structured*.

Definition 6.3 (V canonical structure).
Consider a system with two inputs and two outputs. If the coupling of inputs to outputs exhibits a feedback structure as shown in Figure 6.3b where V_{jk} are the respective transfer functions for all j and k, then we call it *V canonically structured*.



(a) P canonical structure (b) V canonical structure

Figure 6.3.: Canonical structures of MIMO systems with two inputs and two outputs

Both structures are often found in practice showing the properties given in Table 6.3.

Table 6.3.: Properties of P and V canonical structure

P canonical structure		**V canonical structure**	
✓	Direct correspondence to transfer matrix	✗	Required transformation of transfer matrix
✗	Typically no connection to modeling	✓	Direct derivation via modeling
✓	Easy to treat	✗	Difficult to treat
✗	Typically no equivalent of P_{jk}, P_{kj} in real system	✓	Equivalent of V_{jk}, V_{kj} in real system
✗	Physical interpretation questionable	✓	Physical interpretation given

P canonical structures typically arise from input–output transfer descriptions, for instance in frequency-domain models of machine tools or drive systems. In contrast, V canonical structures often reflect the physical interconnection of subsystems, such as force–motion feedback in robot arms or torque–speed feedback loops in automotive powertrains. Since the P canonical structure is more amenable to classical control design techniques, V canonical systems are often transformed into an equivalent P canonical representation. To this end, we have

Theorem 6.4 (Equivalence P and V canonical structure).
Consider two systems with two inputs and two outputs to be given. Suppose one system is in P canonical structure and one in V canonical structure. If

$$\begin{bmatrix} P_{jj} & P_{jk} \\ P_{kj} & P_{kk} \end{bmatrix} = \begin{bmatrix} 1 & -V_{jj} \cdot V_{jk} \\ -V_{kk} \cdot V_{kj} & 1 \end{bmatrix}^{-1} \cdot \begin{bmatrix} V_{jj} & 0 \\ 0 & V_{kk} \end{bmatrix} \tag{6.3}$$

holds, then both systems are equivalent.

In production systems, such transformations are commonly applied when converting physically motivated models — such as coupled thermal–mechanical dynamics in CNC machines — into representations suitable for controller synthesis.
Similarly, in automotive systems, V canonical feedback structures naturally arise from drivetrain and vehicle dynamics, while P canonical structures are preferred for controller implementation and calibration.
Regarding decoupling, we define the following:

Definition 6.5 (Decoupling control).
Consider a system with two inputs and two outputs in P canonical structure. If the control exhibits the structure given in Figure 6.4 where S_{jk} is the transfer function of input u_k to output y_j for all j and k, then we call it *decoupling control*.

The fundamental idea of decoupling control is to interpret cross-couplings as disturbances and to actively cancel or attenuate them through additional control paths. This enables the application of standard SISO design techniques to the resulting decoupled subsystems.
In industrial production, this approach is widely used in multi-axis motion control, for example in high-speed CNC machining. Cross-couplings between axes — caused by structural flexibility or cutting forces — are compensated through decoupling controllers, allowing each axis to be controlled independently while maintaining high precision.
In automotive applications, decoupling control appears in chassis control systems, where interactions between longitudinal and lateral dynamics are compensated to allow separate design of

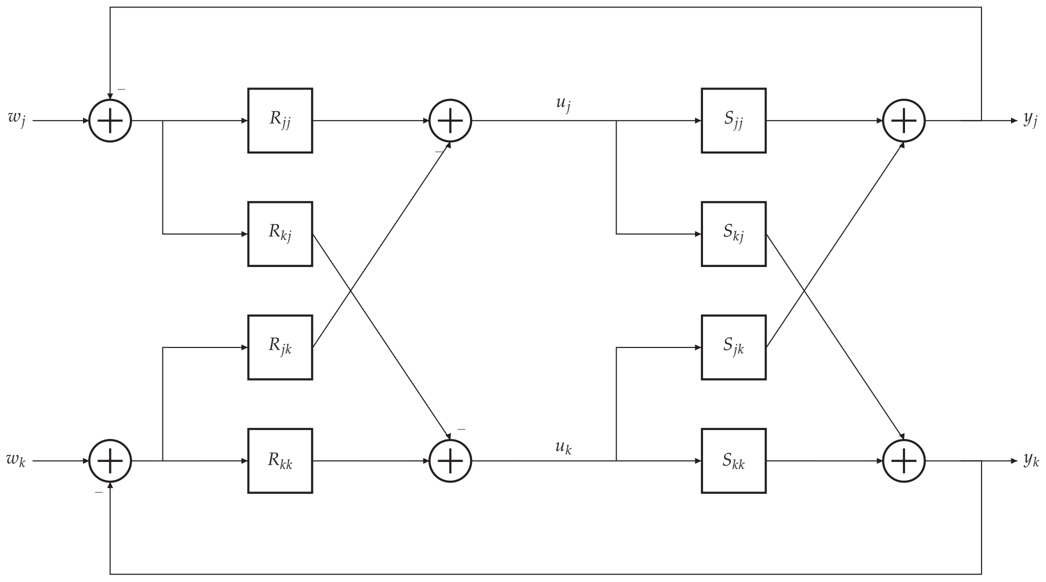


Figure 6.4.: Decoupling structure of MIMO system with P canonical structure

traction and steering controllers.
Within Figure 6.4, there are four controllers which need to be designed. While designing, the intention is that

- R_{jj} shall control y_j using u_j (main system S_{jj}),
- R_{jk} shall eliminate the impact of u_k on y_j (coupling system S_{jk}),
- R_{kj} shall eliminate the impact of u_j on y_k (coupling system S_{kj}), and
- R_{kk} shall control y_k using u_k (main system S_{kk}).

We now focus on eliminating the impact of the second system on the first, cf. Figure 6.5.
In order to eliminate one another, the blue and red paths in Figure 6.5 need to be identical. Hence, we directly obtain

Theorem 6.6 (Decoupling condition).
Consider a MIMO system with two inputs and two outputs in P canonical structure subject to a decoupling control. If the conditions

$$R_{jk} = R_{kk} \cdot \frac{S_{jk}}{S_{jj}} \tag{6.4}$$

$$R_{kj} = R_{jj} \cdot \frac{S_{kj}}{S_{kk}} \tag{6.5}$$

hold, then the system is decoupled.

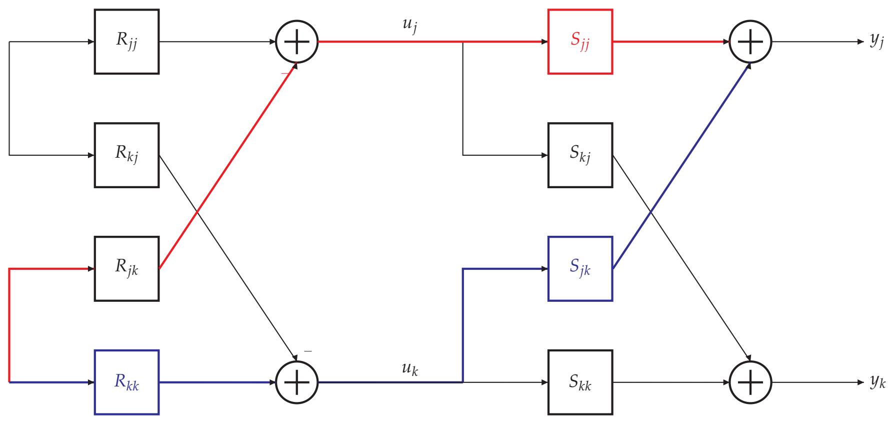


Figure 6.5.: Elimination of coupling

A crucial observation is that even under ideal decoupling, each control channel depends on both the main and the decoupling controllers. Consequently, any modification of a main controller requires a corresponding redesign of the decoupling controllers, which significantly increases development and maintenance effort.

Table 6.4.: Advantages and disadvantages of control decoupling

Advantage	**Disadvantage**
✓ Keeps for MIMO structure	✗ Computationally involved
✓ Standardized P and V structures	✗ Structure limited in usage or derivation
✓ Allows for independent design	✗ Require additional controllers
✓ Allows for basic methods	✗ Requires specific decoupling structure

The elegance of state-, time-, and control-based decoupling lies in their ability to reduce complex systems to simpler subsystems. However, this simplification inevitably introduces artificial inventories, time delays, and disturbance paths that are technically unnecessary.
In the following section, we therefore move beyond decoupling and introduce concepts that explicitly embrace interaction and coupling, laying the foundation for digital twins and cyber-physical systems to avoid these shortcomings.

6.2. Digital twin

In the context of networking, an alternative approach to decoupling is to avoid decoupling altogether by reformulating the system or process at a software-based integration layer, where the

overall system behavior can be addressed holistically. In contrast to classical decoupling strategies, this approach allows all relevant aspects of a system or process—such as structure, dynamics, constraints, and interactions—to be handled explicitly and consistently at the software level.
A prerequisite for this approach is the existence of a digital representation of the system or process. In the following, we adopt the terminology and concepts proposed by the Industrial Internet Consortium (IIC) [19] and the National Institute of Standards and Technology (NIST) [26]. The fundamental building blocks of such representations are so-called data elements.

Definition 6.7 (Data element).
A *data element* is a basic unit of information built on standard structures haveing a unique meaning and distinct units or values.

Based on data elements, more expressive digital abstractions of systems and processes can be constructed.

Definition 6.8 (Digital representation).
Consider a system/process to be given. A *digital representation* is a data element representing a set of properties of the system/process.

While digital representations are most commonly used to describe physical entities, such as machines, vehicles, or production lines, the above definition is deliberately more general. It also applies to non-physical entities, for instance to represent software behavior, control logic, or communication protocols at an abstract level.
Historically, such representations have predominantly been document-centric, relying on textual specifications, diagrams, and informal descriptions. In contrast, this lecture adopts a model-centric perspective, which is well established in systems engineering but – at least in current industrial practice – is still not consistently applied across all domains of automation.
A central modeling language supporting this paradigm is the Systems Modeling Language (SysML), which utilizes the following:

Definition 6.9 (Systems modeling language (SysML)).
Consider a system/process. Then we call a set of

- requirements,
- behaviors given by
    - activity diagram

  - sequence diagram,
  - state machine diagram, and
  - use case diagram
- structures given by
  - block definition diagram,
  - internal block diagram,
  - parametric diagram, and
  - package diagram

a model according to the *systems modeling language*.

The diagrams provided by SysML follow specific syntax and semantics and are closely related to the Unified Modeling Language (UML). Figure 6.6 illustrates the relationships between the different diagram types and highlights their role within a coherent system model.

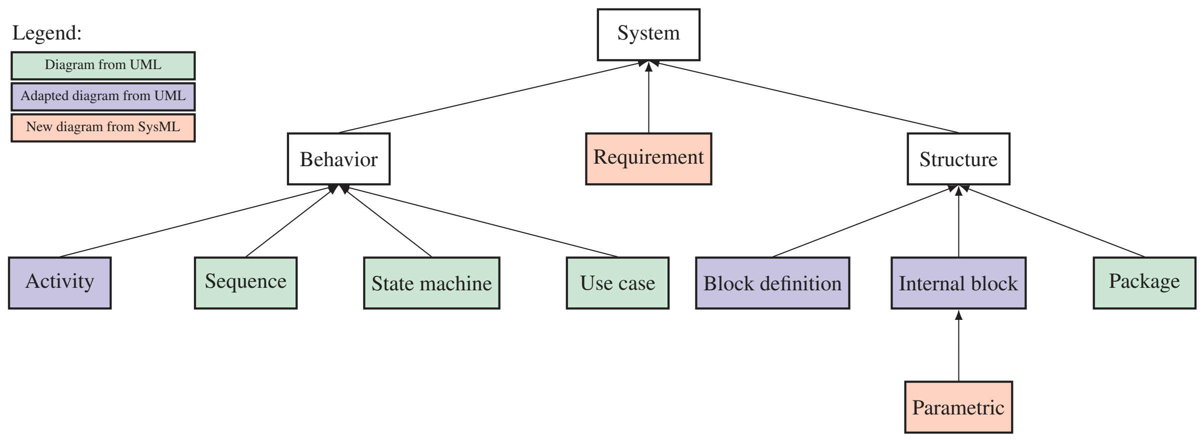


Figure 6.6.: Diagram taxonomy for systems (according to SysML)

Using this digital representation, we can define a digital twin.

Definition 6.10 (Digital model/shadow/twin).
Suppose a system/process with inputs and outputs, a digital representation of the same system/process and communication possibility between both to be given.

- If there exists at least a manual data flow from the system/process to the digital representation, then we call the digital respresentation a *digital model*.

- If there exists at least an automated data flow from the system/process to the digital representation, then we call the digital respresentation a *digital shadow*.
- If there exists a bidirectional automated data flow between the system/process and the digital representation, then we call the digital representation a *digital twin*.

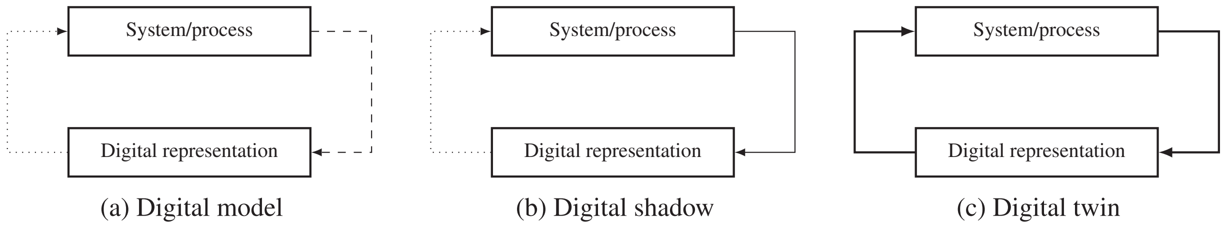


Figure 6.7.: Comprehend the difference between digital model/shadow/twin

Using Definition 6.10, we see that the difference between the three forms exists in the interaction structure. Depending on the state of digitalization within a company, we may find all of the latter to be applied on machine level, cf. Figure 6.8 for an illustration.

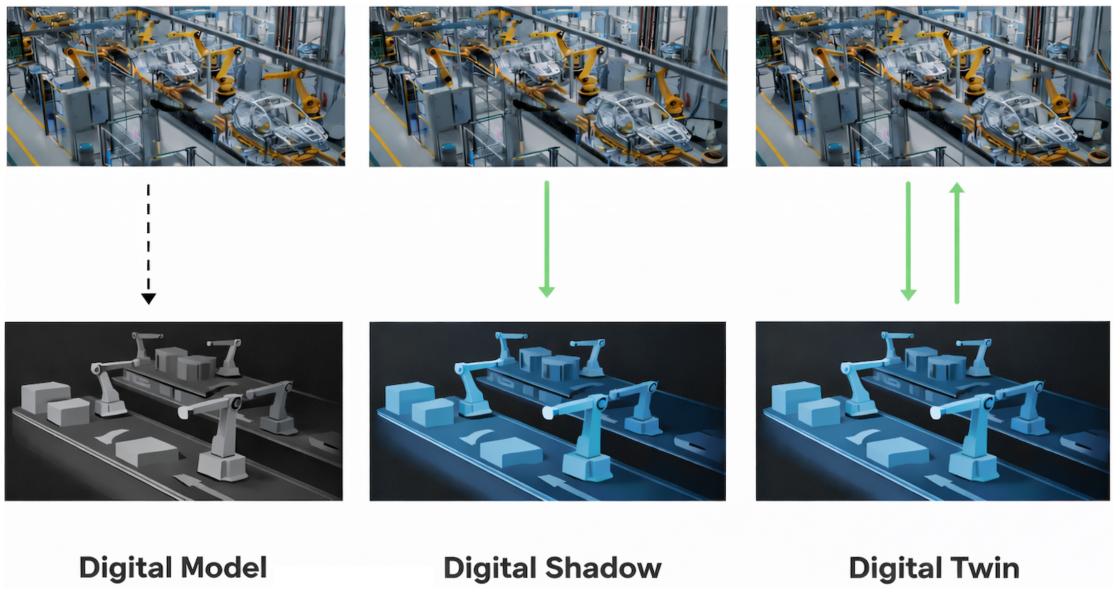


Figure 6.8.: Illustration for digital representations in production, generated with chatgpt.com

Yet, the original (or final) intention of these digital representations is different and illustrated in Figure 6.9:

- In the case of a digital model, data are transferred manually from the physical system or process to the digital representation. The primary intention of this setup is to gain qualitative and quantitative insights into system behavior, performance limits, and design alternatives. Consequently, digital models are predominantly used on the strategic layer, for example during system design, commissioning preparation, or what-if analyses.

- If an automated, potentially real-time capable data flow from the physical system to the digital representation exists, the digital representation is referred to as a digital shadow. In this case, the digital representation continuously reflects the current state of the system or process. Digital shadows are therefore particularly suited for monitoring and reporting purposes and naturally operate on the tactical layer, supporting planning, diagnostics, and predictive maintenance.
- Finally, if a bidirectional, automated data flow between the physical system or process and its digital representation is established, the digital representation becomes a digital twin. This bidirectional coupling allows the digital twin not only to observe but also to actively influence the physical system. As a result, digital twins can be deployed on the operational layer, where they support closed-loop optimization, adaptive control, and autonomous decision making.

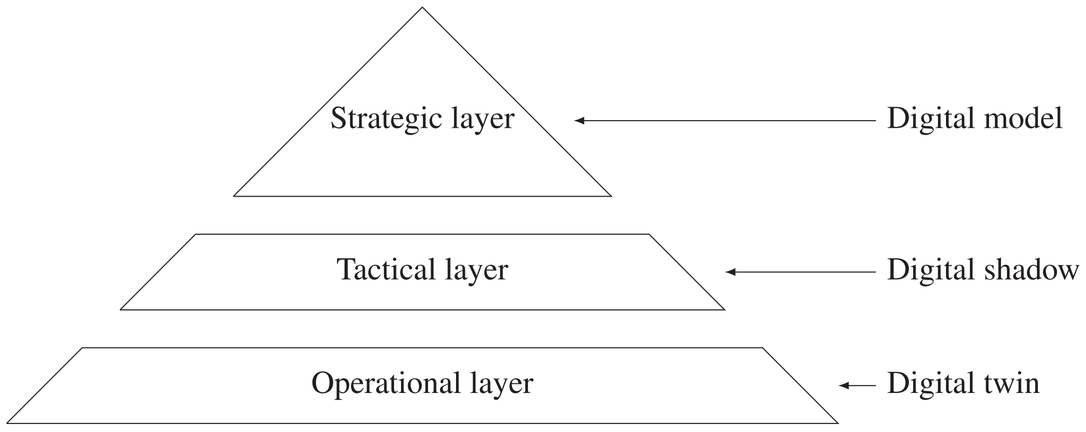


Figure 6.9.: Working layers for digital representations

In industrial practice, digital models are used to simulate production lines or CNC machining processes offline. This allows engineers to evaluate different layouts, cycle times, and tool paths before implementing any physical changes. Similarly, digital vehicle or powertrain models are used during the development of concepts to evaluate energy consumption, emissions, and vehicle dynamics under different design assumptions.
Regarding digital shadows, machine data streams are used to visualize utilization, energy consumption, and wear indicators on dashboards for operators and maintenance personnel. In the automotive industry, digital shadows aggregate sensor data, such as temperatures, fault codes,

and usage profiles. This enables condition monitoring and maintenance planning without directly influencing vehicle behavior.
Lastly, digital twins adapt process parameters, scheduling decisions, and energy usage in manufacturing systems in real time, thereby optimizing throughput and quality. Similarly, an automotive digital twin can enable functions such as adaptive powertrain control, thermal management optimization, and fleet-level software updates. In this case, digital models interact directly with physical vehicles.

Remark 6.11
Note that a representation can be given in many forms such as models or networks, which we discussed in the lecture, but may also consist in lookup tables, databases or other digital types, but may also take a physical form like analog computers.

Hence, not all digital forms can or should be applied uniformly across all management tasks; rather, each form is most effective when aligned with a specific management layer. These layers traditionally operate on different levels of abstraction and aggregation of the physical system or process.
A machine operator interacts with the concrete behavior of individual machines and actuators, focusing on parameters such as setpoints, alarms, and local disturbances. A shift supervisor aggregates information across machines or workstations, addressing throughput, quality deviations, and short-term scheduling within a production segment. At a higher level, a plant manager evaluates overall plant performance, including, e.g., output, energy consumption, and availability, while a supply chain manager operates on an even more abstract level, coordinating material flows, logistics, and intercompany interfaces across sites or organizations.

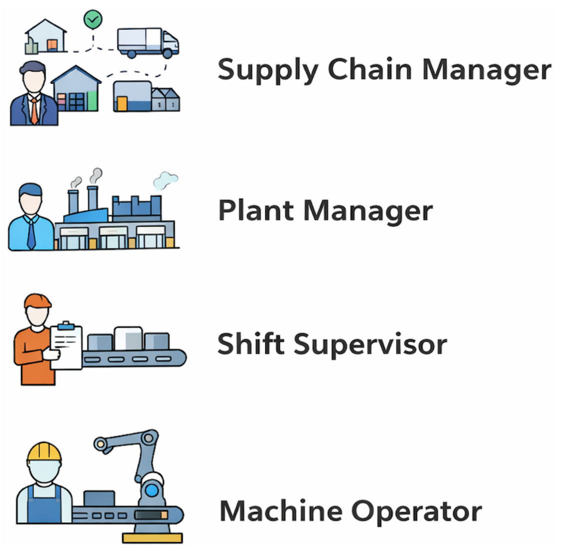


Figure 6.10.: Abstraction levels for digital representations, generated with chatgpt.com

A similar stratification can be observed in automotive systems. At the vehicle level, a digital twin of a powertrain or battery system may be used by control engineers or onboard software to optimize efficiency, thermal behavior, or aging during operation. At the fleet level, digital shadows aggregate data from thousands of vehicles to support maintenance planning, software updates, and usage pattern analysis. At the highest level, digital models of vehicle fleets and logistics networks are employed to support production planning, supply chain optimization, and lifecycle management across manufacturers, suppliers, and service providers.

Remark 6.12
As the last example indicates, the abstraction level is not directly proportional to the possibility of interaction with the system at hand.

Apart from being able to consider the overall system/process, a digital version shows two additional advantages: For one, a digital version is easier to manipulate and assess in a virtual environment that the physical system in the real world. This allows for cost-effective exploration of the behavior of the system under testing conditions. Secondly, the data from these experiments can be used to improve the system/process itself, e.g., maintenance, design, robustness etc. Hence, the application fields for digital representatives span (but are not limited to)

- Model validation with real data
- Decision support
- Identification of waste
- Optimization of overall performance
- Prediction of changes within the system/process
- Exploration of new application and revenue streams

Taking a more generic look at Figure 6.9, the idea of layered systems is to generate levels of abstraction of the system/process, which allows to zoom in on specific tasks at machine level, but also to see the big picture of requirements, interfaces, system design, analysis/tradeoff and tests put to the overall system. Here, we like to note that the model centric view provides a structure, which can be followed to properly derive a digital representation, which is suitable for the task it is created for. Additionally, such an approach allows to integrate generalizations in both directions, that is to generate evaluations and model abstractions for higher layers or to integrate realtime communication and control down to machine level.

Remark 6.13
The big advantage of the latter approach is its integration property. It doesn't matter where the starting point of a digital representative is set, the approach allows full and complete integration of properties as well as traceability of properties through the diagrams.

Table 6.5.: Advantages and disadvantages of SysML based digital representations

Advantage	Disadvantage
✓ Represent overall system/process	✗ Requires full taxonomy of SysML
✓ Allows full integration	✗ Rework document centric view
✓ Allows aggregation and specification	
✓ Allows assessment in virtual space	

Next, we concretize how digital representations are integrated into real systems by combining control and communication mechanisms.

6.3. Cyber physical systems

Our point of departure for a digital twin architecture is the concept of a cyber-physical system (CPS). In the literature [26], this term is associated with time-sensitive functions and varying degrees of interaction between physical and digital components—an idea that is already captured by our definition of a digital twin.

Definition 6.14 (Cyber physical system).
Consider a physical system/process together with KPIs and requirements to be given. Then we call a digital twin a *cyber physical system*, if the bidirectional flow is realtime capable to enforce the requirements and address the key performance indicators.

By definition, every cyber-physical system is a digital twin. However, the converse does not hold. In contrast to a general digital twin, a cyber-physical system must (i) represent a concrete physical system or process, (ii) include a control unit with real-time access to sensors and actuators, and (iii) actively influence the physical system to improve KPIs while respecting operational constraints and requirements.
To integrate the latter into one particular problem, we first need to specify a key performance indicator in the setting of our system dynamics

$$\begin{aligned} \dot{\mathbf{x}}(t) &= f(\mathbf{x}(t), \mathbf{u}(t), t), \quad \mathbf{x}(t_0) = \mathbf{x}_0 \\ \mathbf{y}(t) &= h(\mathbf{x}(t), \mathbf{u}(t), t). \end{aligned} \tag{1.1}$$

Focusing on the state space, we typically consider requirements in terms of constraints (cf. Definition 1.14) and key performance indicators (cf.Definition 1.13) in terms of cost functions. These combined information on state and input of the system to quantify performance of the control.

Definition 6.15 (Cost function).
We call a key performance criterion given by a function $\ell : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ a *cost function*.

Since a cost function evaluates performance at a single time instant $t \in \mathcal{T}$, overall performance must be assessed over an operating horizon. This leads to the notion of a cost functional.

Definition 6.16 (Cost functional).
Consider a key performance criterion $\ell : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$. Then we call

$$J(\mathbf{x}_0, \mathbf{u}) := \int\limits_0^\infty \ell(\mathbf{x}(t, \mathbf{x}_0, \mathbf{u}), \mathbf{u}(t))dt \tag{6.6}$$

cost functional.

Remark 6.17
Note that the latter definition can be used across operational, tactical and strategic level whereas Definition 2.20 is applicable for planning, i.e. the strategic level only.

Integrating these components allows us to quantify not only operating points (Definition 5.6), but also the transients from the current state of the system to such an operating point.

Definition 6.18 (Optimal control problem).
Consider a system (1.1) and a cost functional (6.6). Then we call

$$\begin{aligned} \min \; J(\mathbf{x}_0, \mathbf{u}) \quad & \text{over all } \mathbf{u} \in \mathcal{U} \\ \text{subject to } \dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t), t), \quad & \mathbf{x}(t_0) = \mathbf{x}_0 \\ \mathbf{x}(t) \in \mathbb{X}, \quad & \mathbf{u}(t) \in \mathbb{U} \end{aligned} \tag{6.7}$$

an *optimal control problem*.

The idea of the latter problem in automation engineering is to generate automated solutions without the need of human interaction. To this end, technology is used to plan and change existing devices and monitor the performance of the resulting device.

Link: For solution methods regarding optimal control problems we refer to Control Engineering 2 & 3.

In order to solve problem (6.7), we require not only a control logic, but also computing, storage, communication and planning/modeling components, cf. Figure 6.11 for a generic sketch.

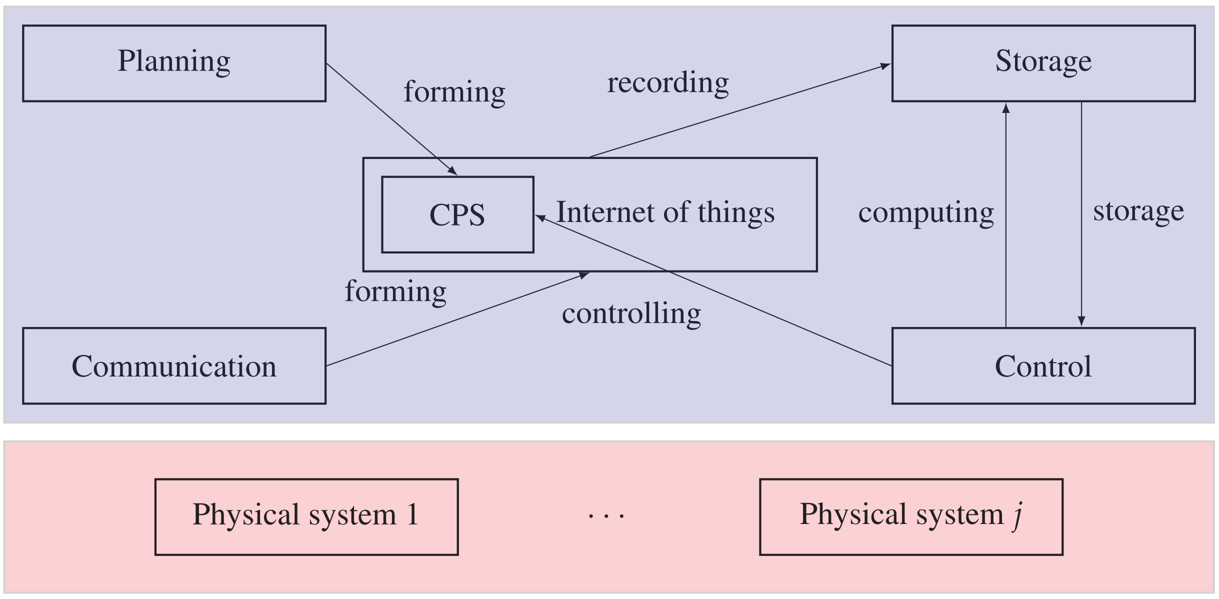


Figure 6.11.: Generic sketch of a CPS structure

The terms in Figure 6.11 specify generic components only. In practice, there are several possibilities and methods, which can be chosen for each of these components, cf. Figure 6.12 for some of the current buzzwords.
Figure 6.12 illustrates that cyber-physical systems naturally give rise to large volumes of heterogeneous data and thus enable the application of optimization and AI-based methods. Nevertheless, their primary purpose is not data generation per se, but the tight coupling of physical dynamics with a digital representative in order to continuously capture the system or process state and to act upon it in a time-sensitive manner. In this sense, cyber-physical systems serve as the operational backbone that connects sensing, computation, communication, and control. Analogous to the application layers of digital representations, cyber-physical systems can be structured into five functional levels, as depicted in Figure 6.13. These levels range from the intelligent connection layer, which provides sensor integration and communication, to the cognitive and configuration layers, where higher-level reasoning, optimization, and adaptation take place.
Considering production, sensors at the intelligent connection layer acquire data on positions, forces, and temperatures of machines and workpieces. These data may be fused and analyzed to detect deviations or wear at the data transformation and monitoring layers. At the cognitive

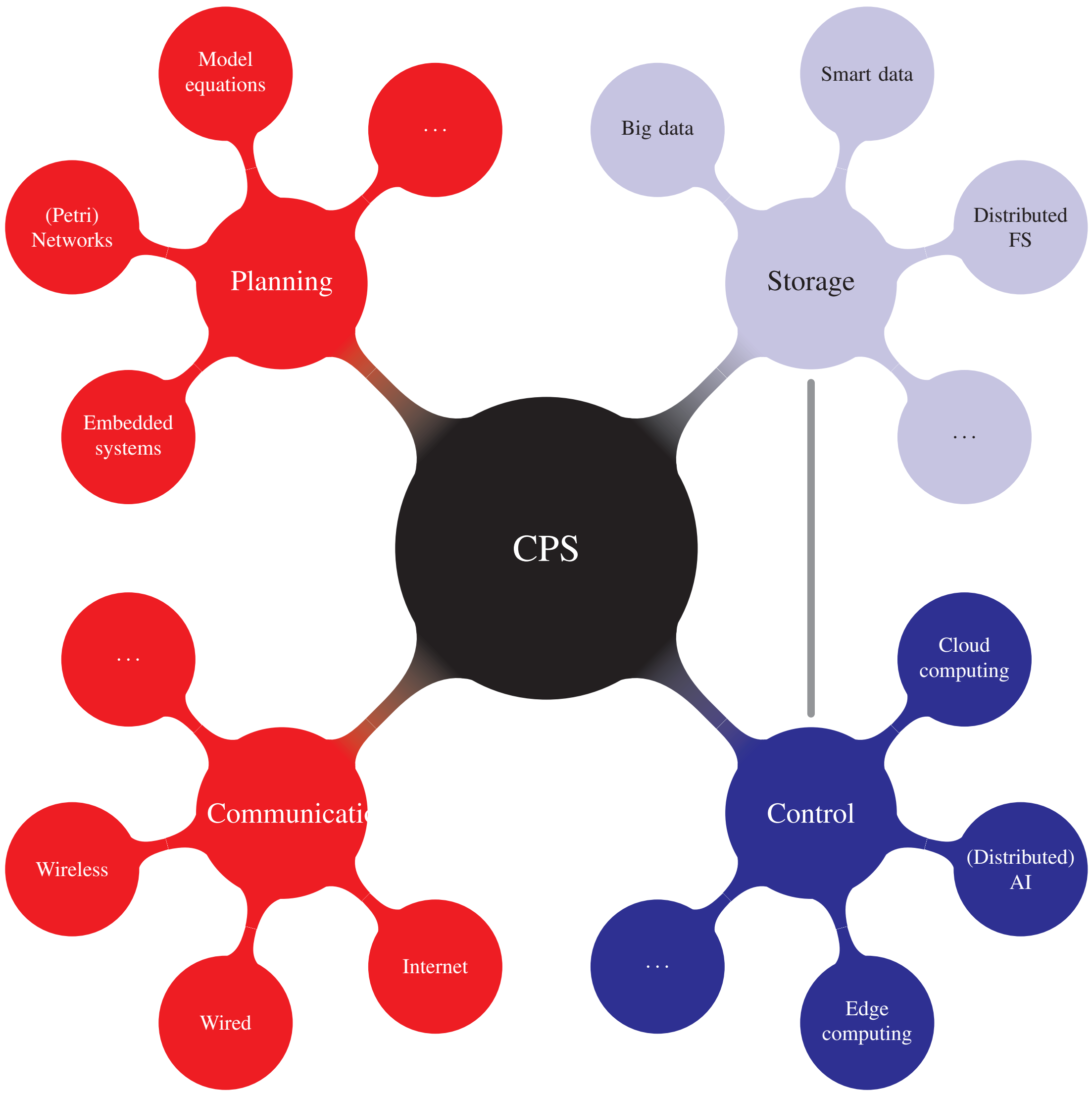


Figure 6.12.: Possibilities for cyber physical system components

and configuration layers, the system may automatically adapt process parameters or reconfigure machine settings to maintain product quality and throughput in the presence of disturbances or equipment degradation.

Similar to prodcution, cyber physical architectures are employed in the automotive range, e.g., as advanced driver assistance systems or powertrain management. Here, sensor networks collect data from cameras, radar, and vehicle states, which are fused and monitored to assess the driving situation and vehicle condition. Higher cyber physical system layers then may optimize control actions, such as torque distribution or energy management, and may autonomously adjust system behavior to changing environments, traffic conditions, or component aging.

At the most fundamental level, cyber physical systems aim to make tacit knowledge of humans, machines, methods, and design intent explicit. This knowledge is successively embedded into

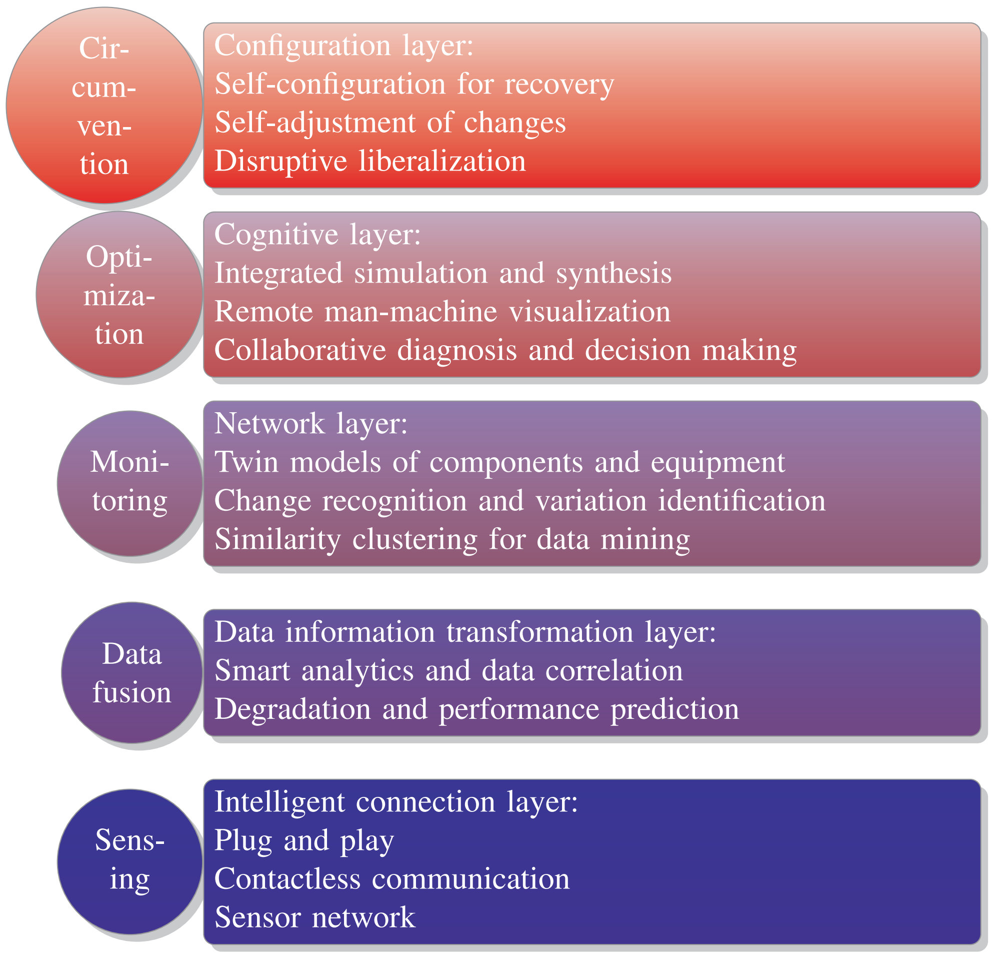

Cir-cum-ven-tion
Configuration layer:
Self-configuration for recovery
Self-adjustment of changes
Disruptive liberalization
Opti-miza-tion
Cognitive layer:
Integrated simulation and synthesis
Remote man-machine visualization
Collaborative diagnosis and decision making
Moni-toring
Network layer:
Twin models of components and equipment
Change recognition and variation identification
Similarity clustering for data mining
Data fusion
Data information transformation layer:
Smart analytics and data correlation
Degradation and performance prediction
Sens-ing
Intelligent connection layer:
Plug and play
Contactless communication
Sensor network

Figure 6.13.: Structure levels of cyber physical systems

software, software into hardware platforms, and hardware into physical devices, thereby enabling intelligent, adaptive, and scalable automation systems.

Table 6.6.: Advantages and disadvantages of cyber physical systems

Advantage	Disadvantage
✓ Represent overall system/process	✗ Requires realtime computation
✓ Allows for layers of methods	✗ Requires realtime communication
✓ Maps digital representation physically	✗ May require big data analytics

As outlined, cyber physical systems collect a large amount data upon which they act in realtime. In a greater picture, this data may be used for other purposes which leads us to the idea of services.

6.4. Industrial cloud platform

The core idea of a cloud platform is to provide customers with on-demand access to services such as computing resources, storage, software, or data that are available anytime and anywhere. To characterize this concept rigorously, we first introduce its fundamental building blocks. As before, we follow the terminology and definitions established by the Industrial Internet Consortium [19] and the National Institute of Standards and Technology [26].
Since a cloud platform provides access to resources and functionality, some form of interaction mechanism is required. More generally, this is captured by the notion of an interface.

Definition 6.19 (Interface).
Consider a system/process and a respective digital representative to be given. An *interface* is a named set of operations to read and set data of the system/process or its digital representative.

An interface therefore allows interaction of both the system and its cyber version with the outside, but also allows interaction between both internally. Based on such interfaces, access to functionality can be structured and exploited, for instance to evaluate KPIs or enforce constraints.

Definition 6.20 (Service).
A *service* is a distinct part of the functionality that is provided through interfaces. A service is called *metered* if it is payed by use and depend on the *quality* of the functionality.

This definition allows services to be applied not only to cyber-physical systems relying on digital twins, but also to digital shadows and digital models. Consequently, the applicability of services is not restricted by real-time requirements or the presence of a physical system.
One common way to provide such services is via cloud-based infrastructures.

Definition 6.21 (Industrial cloud platform).
Suppose a system/process and a digital representative to be given. Then an *industrial cloud platform* is a metered service for the digital representative.

Remark 6.22
Here, we utilized the European formulation [7], which explicitly integrates quality of service, metering and in its original also stakeholders. In contrast to that, the US version addresses IT configurability regarding resources.

As illustrated in Figure 6.14, industrial cloud platforms support different classes of services that build on the essential characteristics of cloud computing. These services can be deployed using public, private, hybrid, or community infrastructures. The choice of deployment model depends on application-specific requirements such as data sovereignty, latency, scalability, availability, and regulatory constraints. In industrial contexts, this decision is particularly critical, as production continuity, safety, and intellectual property must be ensured alongside economic efficiency.

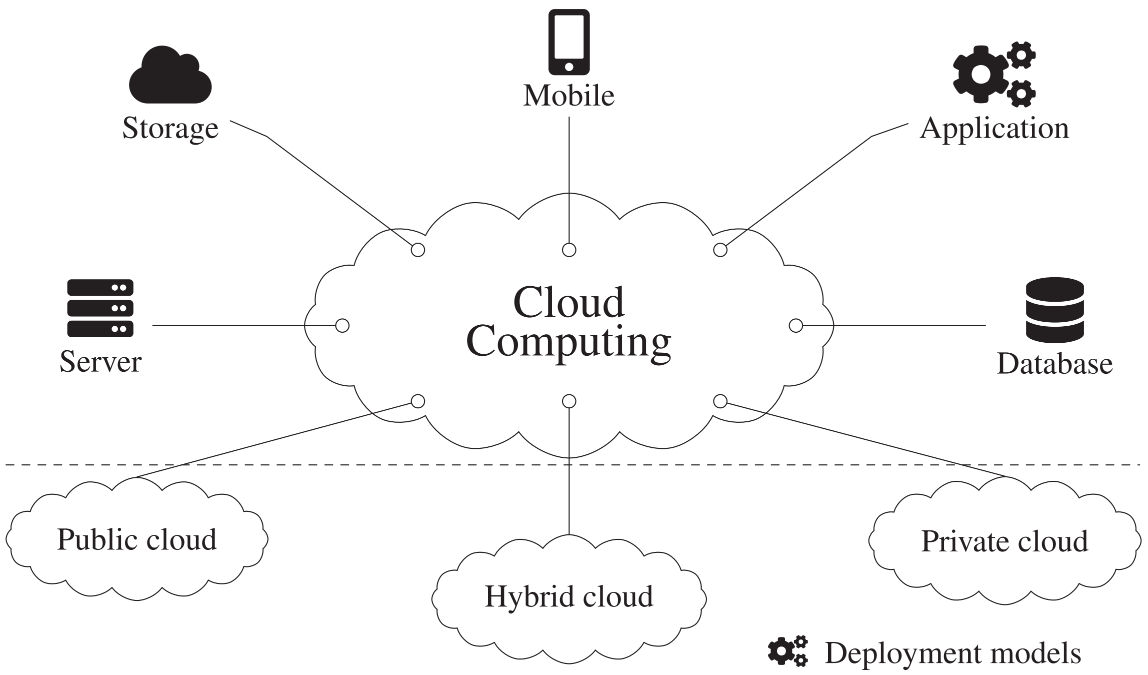


Figure 6.14.: Visual model of industrial cloud

To structure the service offerings of industrial cloud platforms more precisely, we subdivide them according to the degree to which software and hardware resources are provided to the user.

Corollary 6.23 (SaaS, IaaS, PaaS)
Consider a system/process, its digital representative and an industrial cloud platform to be given. Then providing

- *access to delivery or licensing of software is a service called Software-as-a-Service,*
- *utilization of hardware such as compute, storage and networking resources is a service called Infrastructure-as-a-Service, and*
- *access to maintained soft- and hardware is a service called Platform-as-a-Service.*

Remark 6.24
The above service models are not independent of the digital representative and the underlying system or process; rather, they directly build upon the availability and quality of digital models, shadows, or twins.

In all three cases, users of industrial cloud services can focus on their core tasks while outsourcing setup, maintenance, and scaling of IT infrastructure and software components.
In manufacturing, SaaS solutions are commonly used for condition monitoring, quality analytics, or production dashboards. These solutions visualized and analyzed machine data without requiring local installation. PaaS offerings allow engineers to deploy custom optimization or scheduling algorithms on top of plant-wide digital shadows. IaaS is often used for large-scale simulations or virtual commissioning of production lines using digital twins.
In the automotive industry, SaaS platforms support fleet monitoring, predictive maintenance, and over-the-air update management. PaaS environments are used to develop and test vehicle software functions and data-driven driver assistance algorithms. IaaS is used for high-performance computing tasks such as virtual vehicle testing, large-scale simulation of traffic scenarios, and battery aging analysis.
The advantages and disadvantages are summarized in Table 6.7.

Table 6.7.: Advantages and disadvantages of industrial cloud

Advantage	Disadvantage
✓ Separates core from auxiliary tasks	✗ Depends on use case
✓ Utilizes shared resources	✗ Depends on security

6.5. Industrial internet

Industrial cloud platforms constitute components of modern systems and processes that are modeled, monitored, or controlled as integrated wholes. Consequently, any concrete realization of an industrial cloud is typically driven by a specific application or use case.
At the same time, Corollary 6.23 has shown that services provided via industrial clouds are not inherently bound to a single use case. The concept of the industrial internet reutilizes software, data, and infrastructure components. Its objective is to abstract away from individual use cases and to facilitate the systematic modeling, sharing, and reuse of industrial technologies, methods, and

domain knowledge. Ultimately, the industrial internet aims to form an ecosystem characterized by resource enrichment, interoperability, and collaborative participation among heterogeneous stakeholders. Formally, we have the following:

Definition 6.25 (Industrial internet).
Suppose the set of all industrial clouds to be given and any interfaces to the underlying systems/processes to be removed. Then any element of the powerset is called *industrial internet*.

Note that by definition, the industrial internet is not unique but depends on the choice of combined industrial clouds. The combination of clouds additionally stresses the necessity of standardization of systems/processes, their digital representations and all of their components.
The downside of such an idea is that the industrial internet is completely decoupled from all systems/processesfrom which the underlying data and digital representations originate. While this detachment enables scalability and reuse, it also weakens the direct link to operational reality.
Despite its apparent simplicity, the following observation forms the foundation of a large number of companies operating in the field of industrial digitalization:

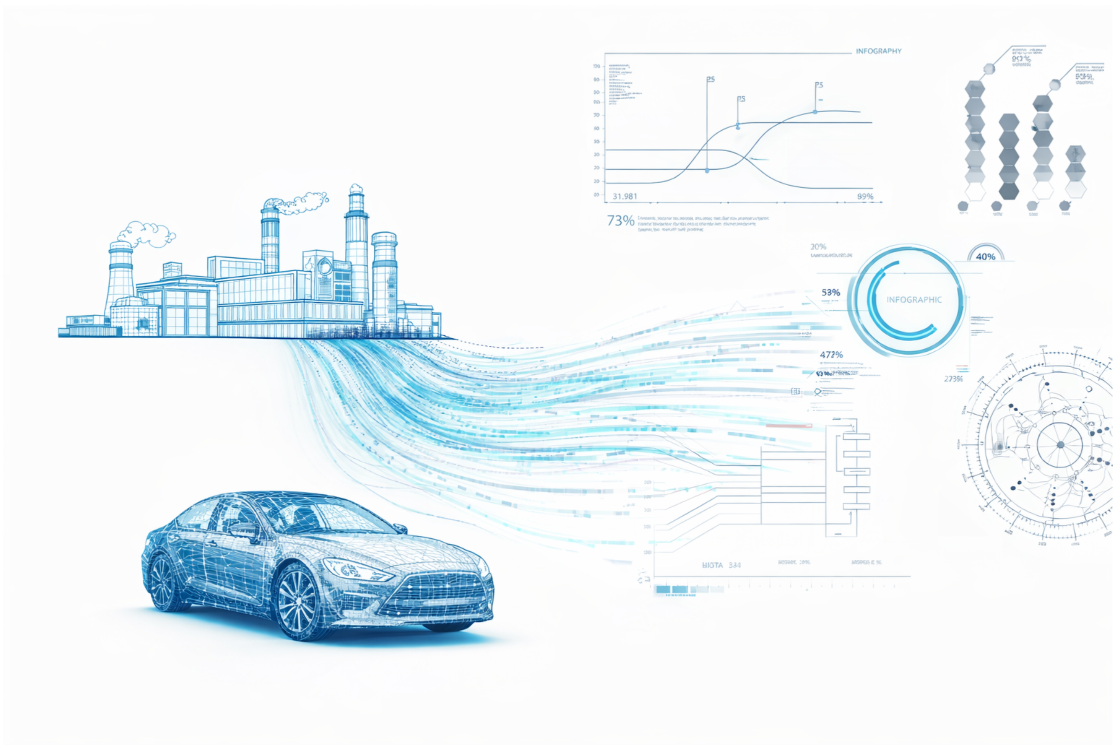
Corollary 6.26
Every industrial internet is a service.

The advantages and disadvantages are summarized in Table 6.8.

Table 6.8.: Advantages and disadvantages of industrial internet

Advantage	**Disadvantage**
✓ Allows aggregation of use cases	✗ Looses connection to reality
✓ Reutilizes methods and data	✗ Misses uniqueness of service

CHAPTER 7 OPTIMIZATION AND LEADING


Generated with chatgpt.com

The original question „Can machines think?“ I believe to be too meaningless to deserve discussion.

Alan Turing

In the preceding chapters we focused on structuring and connecting components using modules, interfaces and networks. Next, the level of optimization and leading is primarily decision-oriented. In particular, we will not introduce new physical building blocks, but convert the results of integration of services into improved system behavior and better decisions along the chain of automation captured in Figure 7.1.

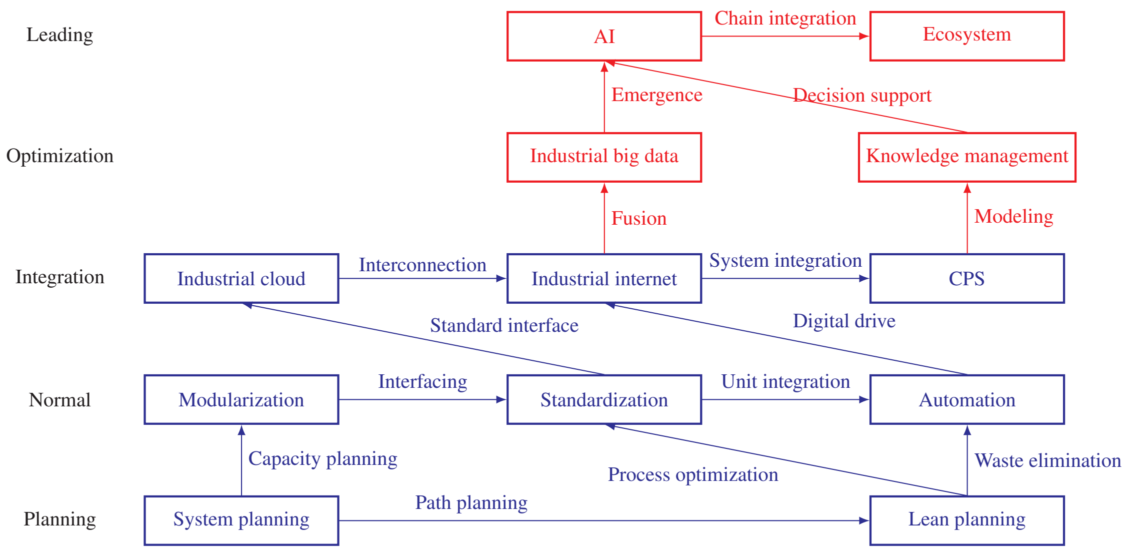


Figure 7.1.: Information in the integrated implementation path of automation

To this end, we discuss industrial big data, i.e., the systematic collection, fusion, and analysis of heterogeneous data streams from machines, products, and IT systems in Section 7.1. The aim of industrial big data is to detect patterns, quantify variability, and derive actionable insights for process and system improvement. Closely related is knowledge management, which we consider next in Section 7.2. This concept aims to capture, structure, and disseminate experience-based (often tacit) know-how, which is typically inherent to persons, e.g. troubleshooting routines, quality rules, or best practices, so that it becomes reusable beyond individuals, shifts, and sites. Within both, we apply artificial intelligence as a class of methods that can learn the latter. Together, these concepts provide the methodological foundation to move from connected automation to scalable improvement, coordinated decision-making, and, ultimately, ecosystem-level integration.

7.1. Industrial big data

In the last decade, the concept of big data has attracted significant attention in both academia and industry. The idea is to utilize this data to retrieve answers for questions of stakeholders. To this end, methods such as Large Language Models (LLMs) such as ChatGPT or others may be applied due to their capabilities in natural language understanding and generation. Here, we use LLMs but do not go into details on how they are defined or operate.

7.1.1. Retrieval-Augmented Generation

While being simple in their usage, LLMs are parametric, rely on static training data, and lack explicit guarantees regarding factual correctness, consistency, and traceability. In industrial, scientific, and engineering contexts these limitations are unacceptable.
The concept of *Retrieval-Augmented Generations* (RAGs) can be employed to cope with these issues by coupling a generative model with a domain based data source, i.e. industrial big data. The concept of this approach was introduce in [23]. Figure 7.2 sketches the workflow of such an approach.

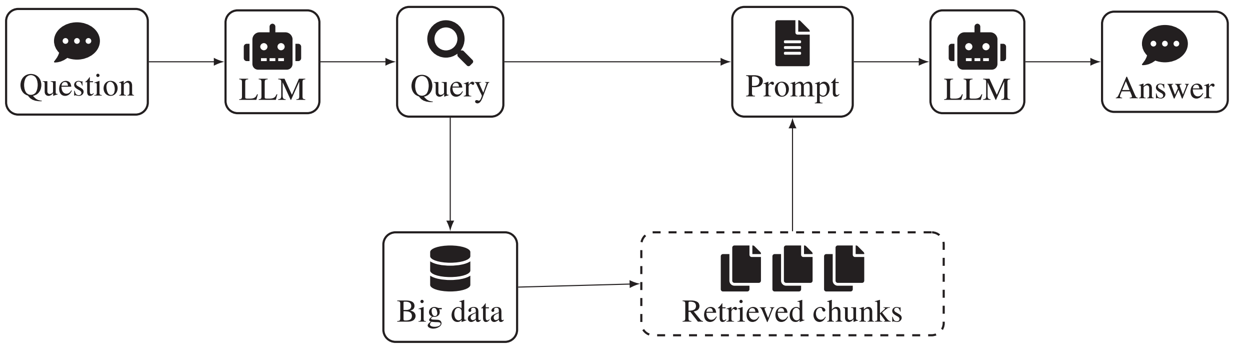


Figure 7.2.: Query for RAG

The separation of the question–answer plane to the data plane allows for a worksplit. In the background, experts work on the big data to keep its properties, e.g. privacy in the context of industrial data. On application layer, users can access the data plane by using common language and don't require programming knowledge to generate queries and interpret retrieved chunks.

Remark 7.1
Note that the shortcomings of LLMs (or AI based methods in general) still apply, but are limited to the question–answer plane, e.g. by generating incorrect queries or insufficient interpretation of results.

Technically, any data source may be used within this workflow to retrieve an answer. In the context of automation engineering, we face the difficulty of handling data, which is not only complex due to its size, but also due to additional properties such as heterogeneity, dynamics, and uncertainty. To properly distinguish big data from conventional data processing as well as from concepts such as the industrial internet, it is therefore necessary to clarify these properties explicitly.

7.1.2. Data processing

In Section 4.1, we discussed the transformation of raw data into information by means of data processing. Note that at that time, we did not specify on how/where such methods are executed and/or data is stored. In the context of on-site automation, we can specify so called traditional data processing approaches:

Definition 7.2 (Traditional data processing methods).
Consider a processing method $m : \mathcal{D} \to \mathcal{I}$ with given data set $\mathcal{D}$ and information set $\mathcal{I}$. Then we call $m : \mathcal{D} \to \mathcal{I}$ *traditional data processing method* if both data set $\mathcal{D}$, information set $\mathcal{I}$ and method $m(\cdot)$ can be stored and executed on a single machine.

Note that depending on the implementation, any service from industrial clouds may also be a traditional processing method. While in Section 4.1 our aim was to retrieve information from data on a sensor level, for a generic service the purpose of a method $m(\cdot)$ may include

- validation (ensuring correctness and relevance of data)
- sorting (putting data in sequence for a given ordering)
- classification (assigning data to categories)
- summation (combining data additively)
- aggregation (combining data by meaning)
- reporting (presenting data for given KPIs)
- analysis (interpreting data given KPIs)

Depending on the chosen purpose, the resulting information set $\mathcal{I}$ may take different forms. A common prerequisite of traditional methods is that the input data is structured, consistently formatted, and semantically well-defined.
In contrast, big data typically comprises a mixture of structured, semi-structured, and unstructured data originating from multiple, often heterogeneous sources, and evolving over time. The challenge is no longer limited to storage, but extends to acquisition, integration, processing, and interpretation.

Definition 7.3 (Big data).
We call a data set $\mathcal{D}$ or method $m : \mathcal{D} \to \mathcal{I}$ big data if at least one of its properties

- volume,
- variety,
- valence,
- velocity,
- veracity,
- value

exceeds the ability of traditional methods.

These properties are often summarized as the „V dimensions“ of big data and highlight that big data is defined by processing limitations, not by absolute size alone.
Note that industrial cloud, industrial internet and big data are intertwined. In particular, we have that:

- The industrial cloud provides the computing infrastructure and services such as storage, compute, and platforms, which are required to handle large industrial datasets. It answers the question where and how data is processed.
- The industrial internet generalizes this idea by enabling the reuse and interconnection of data, models, and services across multiple industrial clouds and organizations. It addresses how data and capabilities are shared and reused.
- Big data focuses on what is done with the data: the analytical methods, statistical models, and learning techniques used to extract patterns, correlations, and trends from industrial datasets.

In short, the industrial cloud is an enabler, the industrial internet is an ecosystem, and industrial big data is an analytical discipline operating on top of both.

7.1.3. Data query

For analytics, i.e. the queries in Figure 7.2, we typically distinguish four different levels, that is

- descriptive analytics (what happened),
- diagnostic analytics (why it happened),
- predictive analytics (what is likely to happen), and
- prescriptive analytics (what should be done).

To formally introduce these, we define the data set for a generic system under consideration

$$\begin{aligned} \dot{\mathbf{x}}(t) &= f(\mathbf{x}(t), \mathbf{u}(t), t), \quad \mathbf{x}(t_0) = \mathbf{x}_0 \\ \mathbf{y}(t) &= h(\mathbf{x}(t), \mathbf{u}(t), t). \end{aligned} \tag{1.1}$$

as follows:

Definition 7.4 (Generic big data set).
Given a system/process (1.1) we call

$$\mathcal{D} = \{(t_j, \mathbf{x}_j, \mathbf{u}_j, \mathbf{y}_j)\}_{j=1}^{N} \tag{7.1}$$

generic big data set including time $t_j \in \mathcal{T}$, state $\mathbf{x}_j \in \mathcal{X}$, input $\mathbf{u}_j \in \mathcal{U}$ and output $\mathbf{y}_j \in \mathcal{Y}$ with number of instances N.

Since we want to analyze this data, we introduce the follow set of measures:

Definition 7.5 (Statistics set).
For a given system/process (1.1) we call $\mathcal{S}$ *set of descriptive statistics* such as means, variances, distributions, correlations, or aggregates.

Furthermore, we are interested in connections between data points. Since typically the original system is not at hand, unknown or lost in aggregation, we have to re-find it. To this end, we introduce the following:

Definition 7.6 (Set of admissible models).
For a given big data set $\mathcal{D}$ we let $\mathcal{M}$ denote a *set of admissible models* of systems.

Utilizing data and statistic sets, we can now introduce the so called descriptive level:

Definition 7.7 (Descriptive analytics).
Given a big data set $\mathcal{D}$ and a statistics set $\mathcal{S}$, we call the problem of computing a mapping

$$A_{\text{descriptive}} : \mathcal{D} \to \mathcal{S}, \tag{7.2}$$

descriptive analytics.

Note that $A_{\text{descriptive}}$ extracts observable properties of historical data without modeling causal relationships or future evolution. Hence, descriptive analytics answers what happened, but does not infer why or what will happen.
In diagnostic analytics, we are interested in why certain data was measured. This corresponds to fitting a suitable model. More formally:

Definition 7.8 (Diagnostic analytics).
Given a big data set $\mathcal{D}$, a statistics set $\mathcal{S}$ and a set of admissible models $\mathcal{M}$, we call the computation of a minimizer

$$f = \underset{f \in \mathcal{M}}{argmin} \{J(\mathcal{D}) \mid J \in \mathcal{S}\} =: A_{\text{descriptive}}(\mathcal{M}, \mathcal{D}, \mathcal{S}) \tag{7.3}$$

diagnostic analytics.

Here, we point out that the statistics set $\mathcal{S}$ is multivalued, hence the minimizing argument in (7.3) is subject to a multiobjective optimization. By identifying a suitable candidate model, diagnostic analytics aims to explain why something happened by uncovering dependencies, sensitivities, or root causes.

Remark 7.9
While all the computations in diagnostic analytics are correct, the choice of statistical measures and admissible models may be wrong. Hence, any result should be verified and validated before it is used in further applications.

In contrast to considering the past, both predictive and prescriptive analytics consider the future.

Definition 7.10 (Predictive analytics).
Given a big data set $\mathcal{D}$, a statistics set $\mathcal{S}$ and a minimizer $f \in \mathcal{M}$ of (7.3), then a map $A_{\text{predictive}} : \mathcal{T} \times \mathcal{X} \times \mathcal{U} \to \mathcal{Y}$ defined by the system/process (1.1) where the dynamics is given by the diagnosed model f is called *predictive analytics*.

Hence, predictive analytics answers what is likely to happen, assuming current trends, inputs, and uncertainties. This allows us to identify strategies, which are optimal for future actions. This basically resembles the optimal control problem we introduced in Definition 6.18 for big data.

Definition 7.11 (Prescriptive analytics).
Given a big data set $\mathcal{D}$, a statistics set $\mathcal{S}$, a minimizer $f \in \mathcal{M}$ of (7.3) and a prediction method $A_{\text{predictive}} : \mathcal{T} \times \mathcal{X} \times \mathcal{U} \to \mathcal{Y}$. Then we call a minimizer

$$\mathbf{u} = \underset{\mathbf{u} \in \mathcal{U}}{argmin} \left\{J(A_{\text{predictive}}) \mid J \in \mathcal{S}\right\} \tag{7.4}$$

prescriptive analytics.

As a result, prescriptive analytics answers what should be done by explicitly linking prediction with optimization and decision-making.

Remark 7.12
Similar to diagnostics, the chosen model may be wrong so the outcome of prediction and prescription should always be verified and validated.

Table 7.1 summarized advantages and disadvantages of big data.

Table 7.1.: Advantages and disadvantages of big data

Advantage	**Disadvantage**
✓ Integrates unstructured data	✗ No new knowledge
✓ Identifies links and predicts behavior	✗ May be arbitrarily wrong

Combined, big data constitutes the analytical backbone to scale computations. While it relies on industrial clouds for scalable computation and benefits from the connectivity of the industrial internet, it does not gain any knowledge such as relations, constraints, causality, and multi-hop dependencies from it.

7.2. Knowledge Graphs and Knowledge Management

So far we have seen how big data can be used to work on data. Yet, data alone is insufficient to manage any type of system/process effectively. What is required instead are structured and semantically grounded representations of *knowledge* that allow interpretation, reasoning, and decision-making beyond numerical correlations.
Here, we particularly focus on *knowledge graphs*, which provide a formal, graph based representation in an explicit and machine-interpretable way. Complementarily, *knowledge management* encompasses the organizational, technical, and methodological processes required to create, maintain, govern, exploit and revise knowledge over its lifecycle. Hence, knowledge graphs are the system whereas knowledge management refers to the process.

Remark 7.13
There exist alternatives to knowledge graphs in the literature such as, e.g.,

- *Relational databases, which offer data consistency and integrity but typically don't cover heterogeneous data schemes and offer no reasoning. Typical applications can be found in ERP systems (enterprise resource planning), production or inventory management.*
- *Rule based systems (also called expert systems), which show explicit reasoning and are easily accessible by humans. Yet, knowledge is typically fragmented, unstructured and the handle changing domains poorly. In applications, these systems are used in diagnostics, fault handling and safety applications.*
- *Document stores such as NoSQL, which provide flexible application and scalable storate for heterogenous data. However, explicit relationships and reasoning are not supported. Such systems are often used for data loggers, sensor aggregation or in MES (manufacturing execution systems).*

7.2.1. Graph Retrieval-Augmented Generation

Similar to RAG, here we focus on an LLM based approach called *Graph Retrieval-Augmented Generation* (Graph RAG), which was introduced in [4]. The latter extends RAG by putting a query to a knowledge graph instead of the data source itself as sketched in Figure 7.3.

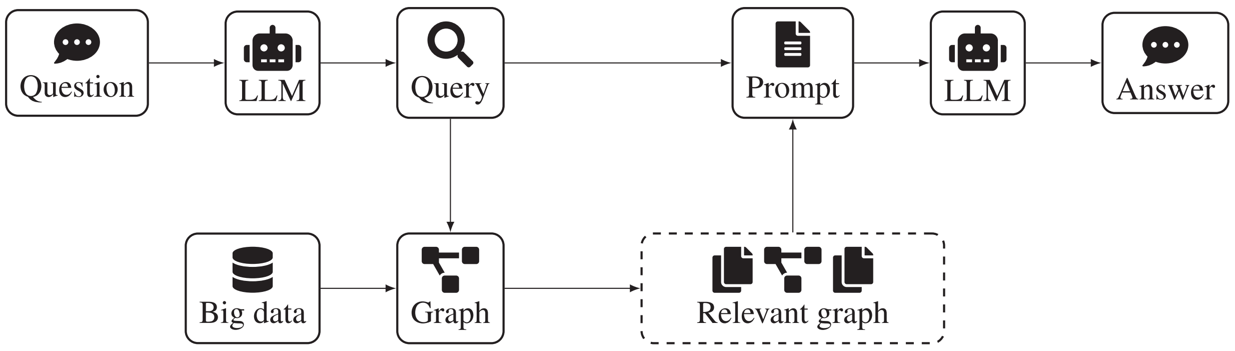


Figure 7.3.: Query for Graph RAG

The big advantage of Graph RAGs over RAGs is that the data plane now operates on graphs, which can be checked for properties similar to Petri Networks, see Section 2.4. We will outline these properties in Section 7.2.3 after properly defining Graph RAGs.
Here, we introduce knowledge graphs similar to networks from Chapter 2 from a mathematical and systems-theoretic perspective and embed them into the broader framework of knowledge management.
Formally, we define a knowledge graph as follows.

Definition 7.14 (Knowledge graph).
A *knowledge graph* is a labeled, directed, attributed network $\mathcal{N} = (\mathcal{V}, \mathcal{E}, \mathcal{R}, \mathcal{L})$, where where $\mathcal{V}$ is a finite set of vertexes, $\mathcal{R}$ is a finite set of relation types, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{R} \times \mathcal{V}$ is a set of directed, typed edges, and $\mathcal{L} : \mathcal{V} \cup \mathcal{E} \to \mathcal{X}$ assigns labels to vertexes and edges.

Remark 7.15
Note that the edges in a knowledge graph extend the notion of a graph by including the relation type. Additionally, labels may be assigned to both vertexes and edges.

From the latter definition, we obtain that a knowledge graph makes explicit

- what exists (objects/vertexes such as physical, logical or conceptual components),
- how things are related (relations, e.g., dependencies, subsets, requirements, conclusions),
- what properties they have (labels such as values, texts, timestamps),
- what is allowed or meaningful (semantics and constraints such as transitivity, causality, safety).

To illustrate such a graph, we utilize an automotive example with two vehicle types featuring different sets of driving functions. The detailed data tables are included in Appendix A. Visualizing the knowledge graph, we obtain Figure 7.4.

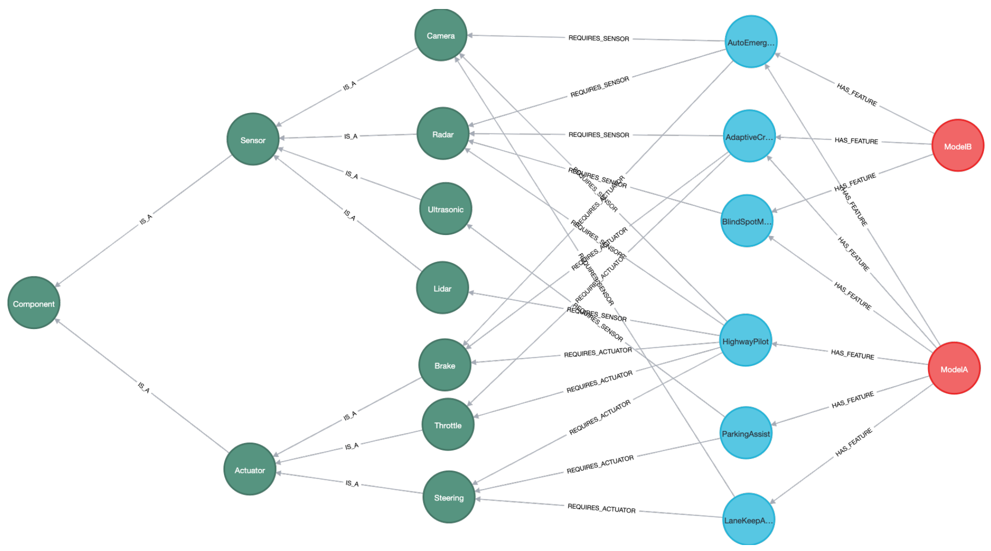


Figure 7.4.: Knowledge graph of an automotive example

Unlike databases, knowledge graphs are not limited to fixed schemes and can naturally represent heterogeneous, interconnected, and evolving knowledge.

7.2.2. Semantics and Ontology

Included within such a graph structure, we find so called *semantics*. Semantic gives a the possibility to introduce meaning and relation in form of sentences subject–relation–object into a graph. Formally, we define the following:

Definition 7.16 (Semantics).
Given a knowledge graph $\mathcal{N} = (\mathcal{V}, \mathcal{E}, \mathcal{R}, \mathcal{L})$ we call a triple

$$(v, r, v) \in \mathcal{V} \times \mathcal{R} \times \mathcal{V}, \tag{7.5}$$

a *semantic*.

Note that semantics can be defined either implicitly through application specific interpretation or explicitly by means of formal logic.

Utilizing our automotive example, we can now ask, e.g., which sensor is used in which driving function, cf. Figure 7.5.

This provides us with the means to define the so called *ontology*, which provides the semantic foundation of a knowledge graph. It defines what kinds of entities exist, how they may relate, and what statements are meaningful or admissible. In other words, it specifies the intended interpretation of the graph structure. More formally, we use the following:

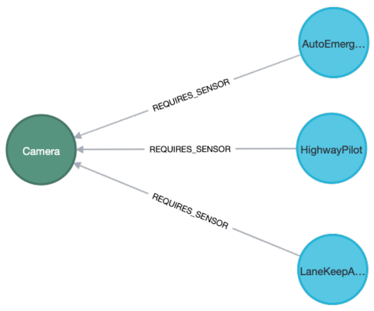


Figure 7.5.: Use of semantics in knowledge graph

Definition 7.17 (Ontology).
An *ontology* is a tuple $\mathcal{O} := (\mathcal{C}, \mathcal{R}, \mathcal{H}, \mathcal{D}, \mathcal{A})$ where

- $\mathcal{C}$ is a finite set called *concepts*,
- $\mathcal{R}$ is a finite set termed *relation types*,
- $\mathcal{H} \subseteq \mathcal{C} \times \mathcal{C}$ is a set defining *hierarchy relations*,
- $\mathcal{D} : \mathcal{R} \to \mathcal{C} \times \mathcal{C}$ is a map assigning a *domain* and *range* to each relation,
- $\mathcal{A}$ is a set of *axioms* defining semantic constraints and logical properties.

The idea of ontologies is to create overviews and see how vertexes and semantics are connected on higher levels. In particular, we can summarize similar vertexes, e.g. all lidar sensors, in one concept „Sensor". On higher level, we can summarize all sensors in a concept „Component", which contains the concept „Sensor" as shown in Figure 7.6. If a relation exists for all lidars, then we can abstract it as a semantic on the concept. The hierarchy allows us to specify, which vertex/concept is contained in which concept, that is

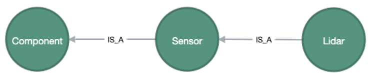


Figure 7.6.: Use of ontology for tracing in knowledge graph

$$(c_{\text{Lidar}}, c_{\text{Sensor}}) \in \mathcal{H} \quad \Longrightarrow \quad c_{\text{Lidar}} \subseteq c_{\text{Sensor}}.$$

The map $\mathcal{D}$ extends the latter to clarify in semantics (7.5), which concept c_j can be a subject and which c_k can be an object. Using the example above, a concept „camera" can be subject in a semantic (camera, measures, picture). Last, the axioms $\mathcal{A}$ describe rules, which are defined a priori and can be used to derive new insights or detect errors. As an example, if a gyro sensor is contained in a concept „camera", this can detect this as an error.

Remark 7.18
Note that an ontology is again a graph consisting of vertexes and edges. These, however, exist on semantic level instead of object level. In order to be well-defined, we need an explicit relation between these two.

Having defined the ontology, we can now introduce the central part of our question–answer scheme, the so called *query*. As for a network in general, we can put a query to a knowledge graph by projecting onto a subgraph.

Definition 7.19 (Graph Query).
Consider a knowledge graph $\mathcal{N}$ and a corresponding ontology $\mathcal{O}$. Then a *graph query* is a mapping

$$q : \mathcal{N} \to \Pi_{\mathcal{O}}(\mathcal{V}, \mathcal{E}), \tag{7.6}$$

selecting subgraphs satisfying elements of the ontology $\mathcal{O}$.

Such queries may either produce tabular output or visualize it. Programs 7.1 and 7.2 showcase feature extraction and tracing, respectively.

```
// Which features use a Camera sensor?
MATCH (s:Component {name: "Camera"})<-[:REQUIRES_SENSOR]-(f:Feature)
RETURN f.name AS FeatureUsingCamera;
```

Program 7.1: Query features per camera

```
// Find all ancestor categories of "Lidar"
MATCH (c:Component {name:"Lidar"})-[:IS_A*]->(ancestor)
RETURN DISTINCT ancestor.name AS AncestorCategory;
```

Program 7.2: Query tracing in ontology for lidar

Remark 7.20
The projection always satisfies $\Pi(\mathcal{V},\mathcal{E}) \subseteq \mathcal{N}$. *However, as the projection is always subject to the ontology, e.g. relations or constraints, it is not equivalent to a configuration, which can be any subset.*

7.2.3. Reasoning

Apart from converting data into a graph, we can apply semantics and ontology to a large data set to determine additional information or make corrections to the graph itself. This is the big difference to RAG, where the data plane was managed by experts but it truly up to data itself. The graph allows us to introduce domain knowledge into the data. By that, we can add new connections and identify/delete incorrect ones.
First, we must walk through the raw observations and supplement them with concepts, relations, relation types, hierarchies, and axioms. Based on these additions, we obtain an initial knowledge graph. However, this graph only contains information that was readily available in our data. Therefore, querying this graph according to Figure 7.3 would yield the same result as querying the data in Figure 7.2.
To improve on the latter, we apply the concept of *reasoning*. To this end, we use *inference rules*, or relations known to be correct. Reasoning involves deriving implicit knowledge from explicit facts.

Definition 7.21 (Graph-Based Inference).
Given a knowledge graph $\mathcal{N} = (\mathcal{V},\mathcal{E},\mathcal{R},\mathcal{L})$ and a set of inference rules $\mathcal{I}$, then we call the map $\mathcal{I}_{\mathcal{N}} : \mathcal{N} \times \mathcal{I} \to \mathcal{N}^\star$ *graph-based inference* where $\mathcal{N}^\star = \text{closure}(\mathcal{N},\mathcal{I})$ is a modified graph.

The fundamental distinction between inference and axioms lies in their ability to express meaning. Axioms define the meaning, while rules implement meaning.
Applying the identification and creation Programs 7.3 and 7.4 within our automotive example, we can identify which vehicles require which sensors.

```
// Infer sensors required by each vehicle based on its features
MATCH (v:Vehicle)-[:HAS_FEATURE]->(f:Feature)-[:REQUIRES_SENSOR]->(s:
    Component)
RETURN v.name AS Vehicle, collect(DISTINCT s.name) AS RequiredSensors;
```

Program 7.3: Infer sensors required per feature

```
// Materialize the inference: link vehicles to needed sensors
MATCH (v:Vehicle)-[:HAS_FEATURE]->(f:Feature)-[:REQUIRES_SENSOR]->(s:
    Component)
MERGE (v)-[:NEEDS_SENSOR]->(s);
```

Program 7.4: Create inference links for sensors required per feature

This is illustrated in Figure 7.7. Note that „NEEDS_SENSOR“ is a newly introduced relation type and the respective relations enrich the graph.

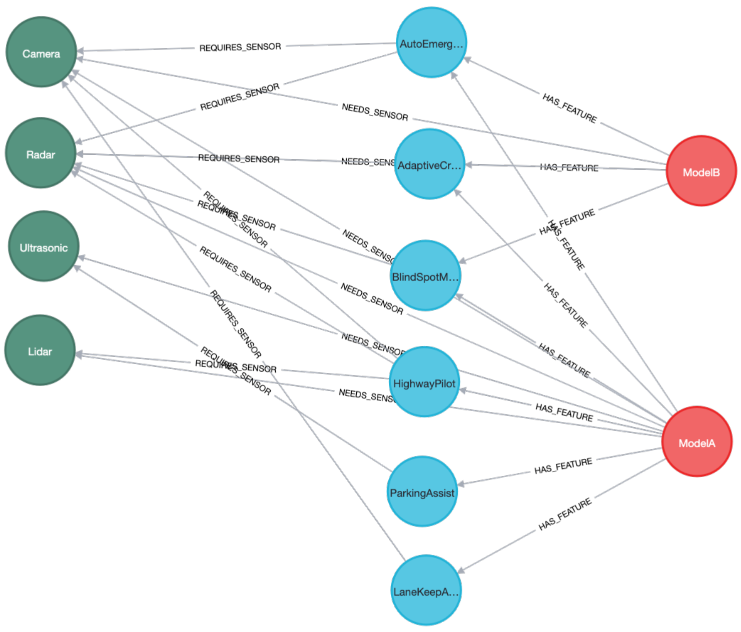


Figure 7.7.: Inference in knowledge graph to identify sensors per vehicle type

In general, applying the set of inference rules $\mathcal{I}$ allows us to identify incorrect relations, which are then deleted from the knowledge graph $\mathcal{N}$. Thus, we obtain a new, consistent knowledge graph $\mathcal{N}^\star$ given the asserted inference rules. Furthermore, we can derive new connections via

- transitive closure, e.g.

$$(a \to b) \wedge (b \to c) \implies (a \to c),$$

- subsumption reasoning, e.g.

$$\text{if } c_{\text{robot}} \subseteq c_{\text{machine}} \text{ then we can conclude robot}(\mathbf{x}, \mathbf{u}) \implies \text{machine}(\mathbf{x}, \mathbf{u}),$$

- constraint, e.g.

$$(a \leq b) \wedge (b \leq c) \implies (a \leq c),$$

- rule-based logic and description logic, e.g.

$$(a, \text{installedIn}, b) \wedge (b, \text{installedIn}, c) \implies (a, \text{installedIn}, c)$$

Once the knowledge graph is completed, it is consistent and offers additional insights that can be retrieved directly by our approach, as shown in Figure 7.3.

7.2.4. Graph Management

In practical applications, the big data basis changes/extends continuously. Consequently, the knowledge graph must be continuously recaptured. To this end, we can repeatedly apply inference to integrate newly available data from the big data basis or account for changes. More formally:

Definition 7.22 (Knowledge Graph Evolution).
Given a knowledge graph $\mathcal{N}$, we call a sequence $(\mathcal{N}_j)_{j=0,\ldots}$ a *knowledge graph evolution* if $\mathcal{N}_0 := \mathcal{N}$ and $\mathcal{N}_j$ being obtained from $\mathcal{N}_{j-1}$ by addition, deletion, or modification of vertexes, edges, or labels.

Due to these ongoing changes, the knowledge graph itself is subject to a management process to ensure it properties.

Definition 7.23 (Knowledge Management).
Knowledge management is the systematic process of creating, maintaining, governing, exploiting and revising a knowledge graph $\mathcal{N}$ within an organization or system.

On a timely basis, a lifecycle of a knowledge graph is a transformation sequence:

$$\mathcal{N}_0 \xrightarrow{\text{creation}} \mathcal{N}_1 \xrightarrow{\text{formalization}} \mathcal{N}_2 \xrightarrow{\text{storage}} \mathcal{N}_3 \xrightarrow{\text{usage}} \mathcal{N}_4 \xrightarrow{\text{revision}} \mathcal{N}_5,$$

where $\mathcal{K}_j$ denotes the knowledge state at stage j. Knowledge graphs themselves primarily operate in the formalization, storage, and usage phases. This highlights that the actual procedure in Graph RAGs are cycled between big data and graph and regarding the graph itself, cf. Figure 7.8.

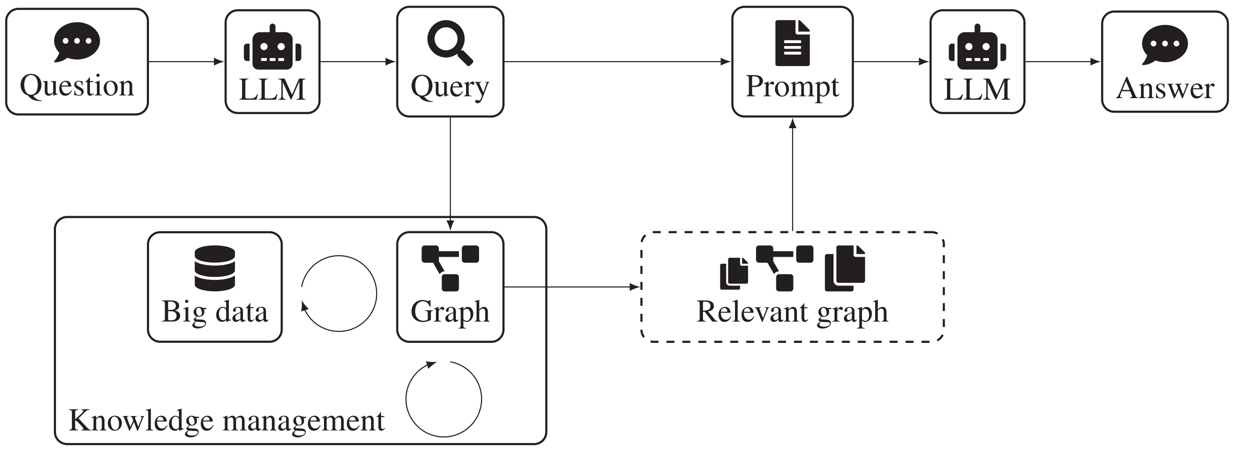


Figure 7.8.: Knowledge management and query for Graph RAG

Summarizing, Graph RAGs extend the RAG paradigm by replacing or augmenting document retrieval with graph-structured knowledge. As such, they overcome purely text-centric or data based approaches and enable reasoning over entities, relations, hierarchies, constraints, and dependencies, which are fundamental in engineered systems.

Remark 7.24
Note that while the graph as such is consistent given the inference rules, this still does not reveal an explainable AI and issues such as hallucination are still present within that approach.

Table 7.2 summarized advantages and disadvantages of Graph RAGs.

Table 7.2.: Advantages and disadvantages of graph RAGs

Advantage	Disadvantage
✓ Integrates ontologies and inference ✓ Identifies multihop links and errors	✗ May be arbitrarily wrong

Appendices

APPENDIX A GRAPHRAG EXAMPLE

The following tables represent an illustrative example, which can be used to showcase the addded value of Graph RAGs over simple data.

Table A.1.: GraphRAG example – vehicles

Name	Autonomy level
ModelA	3
ModelB	2

Table A.2.: GraphRAG example – features

Vehicle	Feature
LaneKeepAssist	"Lane Keeping Assist helps keep the vehicle centered by automatic steering adjustments."
AdaptiveCruiseControl	"Adaptive Cruise Control maintains a set speed and distance from vehicles ahead."
AutoEmergencyBraking	"Automatic Emergency Braking can autonomously apply the brakes to prevent collisions."
BlindSpotMonitor	"Blind Spot Monitoring alerts the driver to vehicles in blind spots during lane changes."

Continued on next page

Table A.2 – continued from previous page

Vehicle	**Feature**
ParkingAssist	"Parking Assist automatically steers the vehicle into parking spots using sensors."
HighwayPilot	"Highway Pilot enables semi-autonomous highway driving by combining multiple ADAS functions."

Table A.3.: GraphRAG example – sensors

Feature	**Sensor**
Camera	"Forward-facing optical camera sensor for lane and object detection."
Radar	"Radio radar sensor for detecting the distance and speed of objects."
Lidar	"Laser-based LiDAR sensor for 3D environment mapping."
Ultrasonic	"Ultrasonic proximity sensor for short-range distance measurement (e.g. parking)."

Table A.4.: GraphRAG example – actuators

Feature	**Actuator**
Steering	"Electronic power steering system for automated direction control."
Throttle	"Engine throttle control for adjusting vehicle speed."
Brake	"Electronic braking system for automated deceleration."

Table A.5.: GraphRAG example – Taxonomy

Parent	**Child**
Component	Sensor
Component	Actuator
Sensor	Camera
Sensor	Radar
Sensor	Lidar
Sensor	Ultrasonic
Actuator	Steering
Actuator	Throttle
Actuator	Brake

Table A.6.: GraphRAG example – Vehicle features

Vehicle	**Feature**
ModelA	LaneKeepAssist
ModelA	AdaptiveCruiseControl
ModelA	AutoEmergencyBraking
ModelA	BlindSpotMonitor
ModelA	ParkingAssist
ModelA	HighwayPilot
ModelB	AdaptiveCruiseControl
ModelB	AutoEmergencyBraking
ModelB	BlindSpotMonitor

Table A.7.: GraphRAG example – Feature requiring sensor

Feature	**Sensor**
LaneKeepAssist	Camera
AdaptiveCruiseControl	Radar
AutoEmergencyBraking	Camera

Continued on next page

Table A.7 – continued from previous page

Feature	Sensor
AutoEmergencyBraking	Radar
BlindSpotMonitor	Radar
ParkingAssist	Ultrasonic
HighwayPilot	Camera
HighwayPilot	Radar
HighwayPilot	Lidar

Table A.8.: GraphRAG example – Feature requiring actuator

Feature	Actuator
LaneKeepAssist	Steering
AdaptiveCruiseControl	Throttle
AdaptiveCruiseControl	Brake
AutoEmergencyBraking	Brake
ParkingAssist	Steering
HighwayPilot	Steering
HighwayPilot	Throttle
HighwayPilot	Brake

The data can be created, maintained, governed, exploited and revised in tools such as Neo4j, which we used here.

```
// Load Vehicle nodes
LOAD CSV WITH HEADERS FROM 'file:///vehicles.csv' AS row
MERGE (v:Vehicle {name: row.Name})
  ON CREATE SET v.autonomyLevel = toInteger(row.AutonomyLevel);

// Load ADAS Feature nodes
LOAD CSV WITH HEADERS FROM 'file:///features.csv' AS row
MERGE (f:Feature {name: row.Name})
  ON CREATE SET f.description = row.Description;

// Load Component (Sensor/Actuator) nodes
```

```
LOAD CSV WITH HEADERS FROM 'file:///sensors.csv' AS row
MERGE (c:Component {name: row.Name})
  ON CREATE SET c.description = row.Description;
LOAD CSV WITH HEADERS FROM 'file:///actuators.csv' AS row
MERGE (c:Component {name: row.Name})
  ON CREATE SET c.description = row.Description;

// Create taxonomy (IS_A relationships)
LOAD CSV WITH HEADERS FROM 'file:///taxonomy.csv' AS row
MERGE (parent:Component {name: row.Parent})
MERGE (child:Component {name: row.Child})
MERGE (child)-[:IS_A]->(parent);

// Create Vehicle HAS_FEATURE Feature relationships
LOAD CSV WITH HEADERS FROM 'file:///vehicle_features.csv' AS row
MATCH (v:Vehicle {name: row.Vehicle})
MATCH (f:Feature {name: row.Feature})
MERGE (v)-[:HAS_FEATURE]->(f);

// Create Feature REQUIRES_SENSOR relationships
LOAD CSV WITH HEADERS FROM 'file:///feature_requires_sensor.csv' AS row
MATCH (f:Feature {name: row.Feature})
MATCH (s:Component {name: row.Sensor})
MERGE (f)-[:REQUIRES_SENSOR]->(s);

// Create Feature REQUIRES_ACTUATOR relationships
LOAD CSV WITH HEADERS FROM 'file:///feature_requires_actuator.csv' AS row
MATCH (f:Feature {name: row.Feature})
MATCH (a:Component {name: row.Actuator})
MERGE (f)-[:REQUIRES_ACTUATOR]->(a);
```

Program A.1: Loading data

```
// Load all data and display
MATCH (n)
RETURN n;
```

Program A.2: Visualizing data

BIBLIOGRAPHY

[1] BLONDEL, V.D. ; GUILLAUME, J.-L. ; LAMBIOTTE, R. ; LEFEBVRE, E.: Fast unfolding of communities in large networks. In: *Journal of Statistical Mechanics: Theory and Experiment* (2008), No. 10. http://dx.doi.org/10.1088/1742-5468/2008/10/P10008

[2] DEUTSCHES INSTITUT FÜR NORMUNG E.V.: *DIN V 19233:1998 Control technology - Process automation - Automation with process computer systems, definitions*. Beuth, 1998

[3] DEUTSCHES INSTITUT FÜR NORMUNG E.V.: *DIN IEC 60050-351 Internationales Elektrotechnisches Wörterbuch Teil 351: Leittechnik (IEC 60050-351:2014-09)*. Beuth, 2014. http://dx.doi.org/10.31030/2159569

[4] EDGE, D. ; TRINH, H. ; CHENG, N. ; BRADLEY, J. ; CHAO, A. ; MODY, A. ; TRUITT, S. ; METROPOLITANSKY, D. ; OSAZUWA NESS, R ; LARSON, J.: *From Local to Global: A Graph RAG Approach to Query-Focused Summarization*. https://arxiv.org/abs/2404.16130. Version: 2025

[5] EMMONS, S. ; KOBOUROV, S. ; GALLANT, M. ; BÖRNER, K.: Analysis of Network Clustering Algorithms and Cluster Quality Metrics at Scale. In: *PLOS ONE* 11 (2016), pp. 1–18. http://dx.doi.org/10.1371/journal.pone.0159161

[6] ERLEBACH, T. ; BRANDES, U.: *Network analysis: Methodological foundations*. Springer, 2005. http://dx.doi.org/10.1007/b106453

[7] EXPERT GROUP ON CLOUD COMPUTING: *The Future of Cloud Computing: Opportunities for European Cloud Computing Beyond 2010*. Commission of the European Communities, 2010. http://dx.doi.org/20.500.12708/36467

[8] GIANI, M. ; FRANK, N. ; VERL, A.: Towards Industrial Control from the Edge-Cloud: A Structural Analysis of Adoption Challenges According to Industrial Experts. (2022). http://dx.doi.org/10.1145/3567445.3567450

[9] INTERNATIONAL ELECTROTECHNICAL COMMISSION: *IEC 61512-4:2009 Batch control*. IEC, 2009

[10] INTERNATIONAL ELECTROTECHNICAL COMMISSION: *IEC 61508:2010 Functional safety of electrical/electronic/programmable electronic safety-related systems*. IEC, 2010

[11] INTERNATIONAL ELECTROTECHNICAL COMMISSION: *IEC 61131:2013 Programmable controllers*. IEC, 2013

[12] INTERNATIONAL ELECTROTECHNICAL COMMISSION: *ISO/IEC 81346:2022 Industrial systems, installations and equipment and industrial products – structuring principles and reference designations*. IEC, 2022

[13] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION: *ISO 7498:1994 Information Technology - Open Systems Interconnection - Basis Reference Model*. Beuth, 1994

[14] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION: *ISO 22400:2014 utomation systems and integration — Key performance indicators (KPIs) for manufacturing operations management*. ISO, 2014

[15] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION: *ISO 9001:2015 Quality management systems - Requirements*. Beuth, 2015. http://dx.doi.org/10.31030/2325651

[16] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION: *ISO 9241:2019 Ergonomics of human-system interaction*. ISO, 2019. http://dx.doi.org/10.31030/3104744

[17] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION: *ISO/IEC Directives, Part 2*. International Organization for Standardization, 2021

[18] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION: *ISO/IEC DIS 27032:2022 Cybersecurity - Guidelines for Internet security*. ISO, 2022

[19] KARMARKAR, A. ; BUCHHEIT, M.: *The Industrial Internet of Things Volume G8: Vocabulary*. Industrial Internet Consortium, 2017

[20] LAI, C.: *Intelligent Manufacturing*. Springer, 2022. http://dx.doi.org/10.1007/978-981-19-0167-6

[21] LANGMANN, C. ; TURI, D.: *Robotic process automation – Digitalisierung und Automatisierung von Prozessen*. Springer, 2020. http://dx.doi.org/10.1007/978-3-658-34680-5

[22] LEICHT, E.A. ; NEWMAN, M.E.J.: Community structure in directed networks. In: *Physical review letters* 100 (2008), No. 11, pp. 118703. http://dx.doi.org/10.1103/PhysRevLett.100.118703

[23] LEWIS, P. ; PEREZ, E. ; PIKTUS, A. ; PETRONI, F. ; KARPUKHIN, V. ; GOYAL, N. ; KÜTTLER, H. ; LEWIS, M. ; YIH, W. ; ROCKTÄSCHEL, T. ; RIEDEL, S. ; KIELA, D.: Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In: LAROCHELLE, H. (Hrsg.) ; RANZATO, M. (Hrsg.) ; HADSELL, R. (Hrsg.) ; BALCAN, M.F. (Hrsg.) ; LIN, H. (Hrsg.): *Advances in Neural Information Processing Systems* Bd. 33, Curran Associates, Inc., 2020, 9459–9474

[24] LUNZE, J.: *Automatisierungstechnik*. 5. Auflage. DeGruyter, 2020. http://dx.doi.org/10.1515/9783110465624

[25] MURATA, T.: Petri Nets: Properties, Analysis and Applications. In: *Proceedings of the IEEE* 77 (1989), No. 4, pp. 541–580. http://dx.doi.org/10.1109/5.24143

[26] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY: *Framework for Cyber-Physical Systems*. National Institute of Standards and Technology, 2015. http://dx.doi.org/10.6028/NIST.SP.1500-201

[27] NEUMANN, K. ; MORLOCK, M.: *Operations Research*. Hanser, 2002. http://dx.doi.org/10.1002/zamm.19940740918

[28] PLENK, V.: *Grundlagen der Automatisierungstechnik kompakt*. Springer, 2019. http://dx.doi.org/10.1007/978-3-658-24469-9

[29] STJEPANDIC, J. ; SOMMER, M. ; DENKENA, B.: *DigiTwin: An approach for production process optimization in a built environment*. Springer, 2022. http://dx.doi.org/10.1007/978-3-030-77539-1

[30] UTTERBACK, J.M. ; ABERNATHY, W.J.: A dynamic model of process and product innovation. In: *Omega* 3 (1975), No. 6, pp. 639–656. http://dx.doi.org/10.1016/0305-0483(75)90068-7

[31] WIKIPEDIA: *CAN bus*. https://en.wikipedia.org/wiki/CAN_bus, 2023. – Accessed: 2025-11-01

[32] ZENG, W. ; KHALID, M.A.S. ; CHOWDHURY, S.: In-Vehicle Networks Outlook: Achievements and Challenges. In: *IEEE Communications Surveys & Tutorials* 18 (2016), No. 3, pp. 1552–1571. http://dx.doi.org/10.1109/COMST.2016.2521642

Jürgen Pannek
Institute for Intermodal Transport and Logistic Systems
Hermann-Blenck-Str. 42
38519 Braunschweig