

DISSERTATION

A Model-driven Engineering Methodology for the Development of Service-oriented Control Procedures for Automated Manufacturing Processes

Vom Fachbereich
Maschinenbau und Verfahrenstechnik der
Technischen Universität Kaiserslautern
zur Verleihung des akademischen Grades
Doktor-Ingenieur (Dr.-Ing.)
genehmigte Dissertation

Vorgelegt von
Diplom-Ingenieurin Lisa Maria Ollinger
aus Idar-Oberstein

Tag der mündlichen Prüfung: 28.03.2017

Dekan: Prof. Dr.-Ing. Jörg Seewig

Promotionskommission:

Vorsitzender: Prof. Dr.-Ing. Jörg Seewig

1. Berichterstatter: Prof. Dr.-Ing. Dr. h.c. Detlef Zühlke

2. Berichterstatter: Associate Professor Charlotta Johnsson

Kaiserslautern, 2017

D386

Zusammenfassung

Die Steuerungsentwicklung als Teil der Planung von automatisierten Produktionsanlagen nimmt einen immer größeren Stellenwert ein. Dies ist zum einen begründet durch den Anstieg der Automatisierungsgrade von Produktionsprozessen und Betriebsmitteln innerhalb der letzten Jahrzehnte [Vyat13]. Zudem werden immer mehr Innovationen durch Automation getrieben, um höhere Verfügbarkeiten, Flexibilität und Anpassungsmöglichkeiten von Produktionsprozessen bei gleichzeitiger Verringerung von Kosten und Markteinführungszeiten für neue Produkte zu ermöglichen [Häst11]. Darüber hinaus werden von strategischen Initiativen wie Industrie 4.0 grundlegende Änderungen im Produktionsumfeld hinsichtlich zunehmender Vernetzung und Autonomie von Produktionsanlagen postuliert, die durch den steigenden Einsatz von Informationstechnik generiert werden [Kage13].

Allerdings kommt die heutige Situation der gestiegenen Bedeutung der Steuerungsentwicklung nicht nach. Innerhalb der Anlagenplanung, die stark vom mechanischen Design dominiert wird, kommt der Steuerungsentwicklung eine eher untergeordnete Rolle zu [Lukm13]. Sie wird typischerweise als letzter Schritt vor der Realisierung begonnen und zieht sich oft noch bis in die Inbetriebnahme der Produktionsanlage [Li12][Mend11]. Zudem fehlt es an etablierten Entwicklungsmethoden und Werkzeugen, welche die Entwicklung der Steuerungsprogramme ideal unterstützen [Vyat13]. Als Konsequenz entstehen heute in der Regel komplexe, unstrukturierte Programme mit geringer Wiederverwendbarkeit [Zühl10].

Diese Arbeit adressiert diese Situation, indem eine neue Entwicklungsmethodik für Steuerungsprogramme vorgestellt wird. Die Grundlage dafür wird durch Konzepte aus der Informatik gelegt, um den Softwareentwicklungsprozess mit Hilfe von Abstraktion, Modularisierung und Modellierung zu verbessern. Hierbei stellt das Konzept der Serviceorientierten Architektur die Grundlage für die Strukturierung der Steuerungsprogramme und einen systematischen Softwareentwicklungsprozess dar. Zudem werden Ansätze aus der modellgetriebenen Entwicklung aufgegriffen, um den Entwicklungsprozess durch das Festhalten der Planungsergebnisse in Modellen durchgängig und anwenderfreundlich zu unterstützen. Um die Steuerungsentwicklung besser in die gesamte Anlagenplanung zu integrieren, werden zudem Ansätze zur Stärkung der interdisziplinären Planung untersucht. Insbesondere Ideen des System Engineerings werden verwendet, um die neue Entwicklungsmethodik in den Anlagenplanungsprozess zu verankern.

Das Ziel dieser Arbeit ist eine modellbasierte Methodik für die Entwicklung von serviceorientierten Steuerungsprozeduren für automatisierte Fertigungsprozesse zu

entwickeln, welche die Effizienz der Steuerungsentwicklung verbessert sowie die Anpassungsfähigkeit und Wiederverwendbarkeit der Steuerungsprogramme erhöht. Die modellbasierte Entwicklungsmethodik umfasst die funktionale Spezifikation der Steuerungsprozeduren und ist unabhängig von bestimmten Technologien. Um diese in der Praxis anwendbar zu machen und zudem ihre Praxistauglichkeit zu demonstrieren, wird ein Implementierungskonzept vorgestellt, das an einem industrienahen Anwendungsfall eingesetzt wird. Die Durchführung verschiedener Anwendungsszenarien soll zudem zur Bewertung der Ergebnisse der Arbeit dienen.

Zu den Ergebnissen dieser Arbeit zählen:

- **Referenzarchitektur für SOA-AT:** Das Paradigma der Serviceorientierten Architektur wird zunächst auf die Domäne der Produktionsautomatisierung transferiert und mit der ursprünglichen Anwendungsdomäne der IT-basierten Geschäftsprozesse verglichen. Es werden zwei Ebenen von SOA-AT Services als Betriebsmittel-Services und Steuerungs-Services definiert, die weiterhin in Kategorien von Services mit spezifische Eigenschaften und Aufgaben unterteilt werden.
- **Vorgehensweise zur Spezifikation von Services:** Für beide Service-Ebenen werden Vorgehensweisen zur Spezifikation der Services vorgestellt. Hierbei werden unterschiedliche Designaspekte bestimmt, die innerhalb von zwei Planungsschritten konkretisiert werden. Zudem werden Bibliothekskonzepte eingeführt, um eine effiziente Wiederverwendbarkeit der Planungsergebnisse zu ermöglichen.
- **PESCOP Prozess:** Für die Entwicklung von SOA-AT Services wird ein systematischer und durchgängiger Entwicklungsprozess definiert, welcher die vier Phasen Analyse, Design, technische Spezifikation und Implementierung beinhaltet. Diese neue Vorgehensweise zur Steuerungsentwicklung wird dann in den Gesamtprozess der Anlagenplanung eingebettet.
- **MDE for SOA-AT:** Das Hauptresultat dieser Arbeit ist eine modellbasierte Methodik für die Entwicklung von serviceorientierten Steuerungsprozeduren, die auf den bereits genannten Teilergebnissen basiert. Für die Darstellung der Planungsergebnisse werden verschiedene Planungsmodelle mit Hilfe von Metamodellen beschrieben. Eine durchgängige Vorgehensweise mit definierten Planungsschritten wird vorgestellt, welche die Erstellung der einzelnen Planungsmodelle beinhaltet.
- **Anwendungskonzepte und Machbarkeitsnachweis:** Die Anwendbarkeit der neuen Entwicklungsmethodik soll mit einem Konzept zur standardisierten Benennung der Services und einem Implementierungskonzept erhöht werden. Auf deren Basis wird die modellbasierte Methodik für die Entwicklung von serviceorientierten Steuerungsprozeduren an einem praxisnahen Demonstrationssystem angewendet.

Table of Contents

1	Introduction	1
2	Engineering of Manufacturing Control Systems.....	3
2.1	Automated Production – Definition, Historical Development and Today's Challenges	3
2.2	Industrial Automation Systems	6
2.3	Control Engineering within the Production Life Cycle	9
2.3.1	Factory Planning.....	9
2.3.2	Production Engineering	11
2.3.3	Control Engineering.....	13
2.3.4	Situation of Control Engineering Today	15
3	Design Concepts for Distributed Control Systems.....	19
3.1	Trends in Industrial Automation.....	19
3.2	Distributed Automation Systems	21
3.3	Concepts for Distributed Control Architectures.....	22
3.3.1	IEC 61499	23
3.3.2	Multi-agent Systems	25
3.3.3	Service-oriented Architecture.....	28
3.3.4	Comparison of the Concepts	32
3.4	Design and Implementation of SOA Systems.....	33
3.4.1	Process-oriented Design Strategy	34
3.4.2	Service Specification	35
3.4.3	Composition Principles	36
3.4.4	Reference Architecture for Services	37
3.4.5	SOA Technologies.....	38

4	Concepts for Efficient Control Engineering.....	41
4.1	General Concepts from Software Engineering	41
4.1.1	Programming Paradigms.....	42
4.1.2	Life Cycle Models.....	43
4.2	Model-driven Engineering of Control Procedures.....	45
4.2.1	Basic MDE Concepts	45
4.2.2	Modeling Languages for Control Engineering.....	47
4.2.3	Existing Approaches for MDE of Control Procedures	52
4.3	Comprehensive Production Engineering Concepts	56
4.3.1	Integrated Engineering.....	56
4.3.2	Mechatronic Systems.....	57
4.3.3	Object-oriented Engineering.....	59
4.3.4	Holonic Manufacturing Systems (HMS).....	60
4.3.5	Systems Engineering	61
4.3.6	Planning of Service-oriented Factory Control Systems.....	62
4.4	Engineering Standards and Guidelines	63
4.4.1	Reference Architectures According to ISA-95 and ISA-88.....	63
4.4.2	Standards Providing Uniform Terms for Modeling	65
4.5	Assessment of the Concepts	66
5	Problem Statement, Objective Target, and Procedural Method	69
5.1	Problem Statement.....	69
5.2	Objective Target	70
5.3	Procedural Method	72
6	Methodology for the Model-driven Development of Service-oriented Control Procedures	75
6.1	Service-oriented Automation.....	75
6.2	Reference Architecture for SOA-AT.....	77
6.2.1	Equipment Services	77
6.2.2	Control Services.....	81
6.3	Specification of Equipment Services	85
6.3.1	Service Description	85

6.3.2	Design Aspects of Equipment Services	87
6.3.3	Library Concepts	88
6.3.4	Abstract and Concrete Specification of Services	90
6.4	Development of Control Services	92
6.4.1	Specification of Control Services	92
6.4.2	Process-oriented Development of Control Procedures	93
6.5	Engineering Process	97
6.5.1	Control Engineering Process	97
6.5.2	Placement within the Production Engineering Process	99
6.6	Model-driven Engineering Methodology	101
6.6.1	MDE for SOA-AT Reference Model	102
6.6.2	Modeling Concepts.....	103
6.6.3	Process Model.....	106
6.6.4	Equipment Model.....	107
6.6.5	Service Model.....	109
6.6.6	Control Logic Model.....	110
6.6.7	Model-driven Engineering Workflow	111
7	Application Concepts	115
7.1	Standardized Naming of Planning Objects	115
7.1.1	Naming of Functions.....	115
7.1.2	Naming of Services and Service Operations.....	116
7.2	Implementation Concept	118
7.2.1	Representation of the Process Model	119
7.2.2	Representation of the Design Phase Models	120
7.2.3	Realization of Services-oriented Control Procedures	123
8	Proof of Concept.....	127
8.1	Description of the Use Case.....	127
8.1.1	Demonstration System	127
8.1.2	Assembly Unit	128
8.2	Application Scenarios.....	129
8.2.1	Execution of the Development Planning	129

8.2.2	Execution of Reconfiguration Tasks	133
8.3	Evaluation.....	137
9	Conclusions and Outlook	143
9.1	Conclusions	143
9.2	Outlook.....	145
10	Summary	149
	Acronyms and Abbreviations.....	151
	Bibliography	155
	Appendix A: Model-driven Engineering Methodology.....	171
	Appendix B: Function Library	173
	Appendix C: Process Model of Use Case	175
	Appendix D: Mappings of Use Case	179
	Appendix E: Abstract Equipment Model of Use Case	185
	Appendix F: Concrete Equipment Model of Use Case.....	187
	Appendix G: Service Model of Use Case	195
	Appendix H: Control Logic Models of Use Case.....	205
	Appendix I: Configuration Tasks.....	209
	Appendix J: SysML2JG Transformation Rules	213
	Curriculum Vitae.....	217

1 Introduction

The present situation of control engineering in the context of automated production can be described as a tension field between its desired outcome and its actual consideration. On the one hand, the share of control engineering compared to the other engineering domains has significantly increased within the last decades due to rising automation degrees of production processes and equipment [Vyat13]. This trend is even intensified since more and more enhancements and innovations are enabled by automation and controls. Manufacturing companies seek for higher performance, flexibility, and adaptability of production processes to produce more product variants in shorter time frames and in a more cost-effective way whereby automation constitutes the key enabler [Häst11]. Moreover, fundamental changes of the manufacturing environment are postulated by strategic initiatives like Industrie 4.0 which are mainly driven by the use of information technology permitting a stronger networking within production systems and more intelligent, autonomous production equipment [Kage13].

On the other hand, the control engineering domain is still underrepresented within the production engineering process, which is primarily dominated by the mechanical design [Lukm13]. This situation is expressed by the mainly sequential execution of the engineering domains where control engineering is the very last step [Li12][Mend11]. Mature application concepts for an interdisciplinary design are missing to realize faster production engineering processes and an overall shorter time-to-market for product innovations [Voge14a].

Another limiting factor is the control engineering itself. Control programs are usually characterized as rigid monolithic software architectures with high complexity and poor reusability [Zühl10]. Obviously, there's a lack of methods and tools to decrease the amount of software engineering efforts and to permit the development of innovative automation applications that ideally support the business requirements [Vyat13].

This thesis addresses this challenging situation by means of the development of a new control engineering methodology. The foundation is built by concepts from computer science to promote structuring and abstraction mechanisms for the software development. In this context, the key sources for this thesis constitute the paradigm of Service-oriented Architecture and concepts from Model-driven Engineering. To mold these concepts into an integrated engineering procedure in accordance with the overall production engineering process, existing approaches from software engineering and comprehensive production engineering are examined.

The overall objective is to develop an engineering methodology to improve the efficiency of control engineering by a higher adaptability of control software and decreased programming efforts by reuse. For making the engineering methodology generally applicable without dependencies to certain technologies, it is specified as a theoretical concept. A further goal of this thesis is to demonstrate the applicability of the methodology and its relevance regarding real automation applications. Therefore, a set of implementation concepts are needed that support the transfer from the theoretical concept into best practices for the efficient application. The proof of concept is to be carried out at an industry-related use case, which builds the basis for an evaluation regarding the initial objective targets.

2 Engineering of Manufacturing Control Systems

For the realization of automated production processes, control systems are needed that comprise the hardware and software to connect, operate, and control the production equipment. In the following, the demands on automated production systems deriving from today's challenges are examined. Furthermore, the characteristics and properties of industrial automation systems and the control system engineering as a part of the whole factory planning are presented. Thereby, the focus is placed on production plants and their control systems for discrete production processes.

2.1 Automated Production – Definition, Historical Development and Today's Challenges

According to Helbing *production* is defined as the collectivity of technology, organization, and tasks to create usable products [Helb10]. The term *manufacturing* is often used interchangeably but usually refers to production in an explicit industrial manner. Industrial production processes can be characterized as discrete, continuous, or batch production depending mainly upon the appearance of the process output [John99][Voge09a]. Continuous production processes produce in a continuous flow, like textiles or fluid chemicals, whereas discrete production processes create individual units like cars or computers. Batch processes have characteristics from both continuous and discrete production, since their output appears as lots or quantities of materials and are popular in chemical, food, and pharmaceutical industries [John99].

Today, many sectors of industrial production are characterized by a high degree of automation, particularly in high-wage countries such as Germany [VDI13]. The term *automation* can be generally interpreted as the capability of causing a machine to carry out a specific operation on command, which implies operating or acting independently without human intervention [Nof09]. The biggest automation domain constitutes the *production automation*, which includes the planning, design, and realization of automated production facilities. In terms of an engineering discipline, production automation comprises software and hardware concepts, methods, tools, products, and solutions for controlling fully or partly self-running processes [VDI13].

The history of automated production is closely linked to historical phases known as *Industrial Revolutions* (see Figure 2-1). The first one encompasses the industrialization during the 19th century based on mechanization and mechanical drives with steam as the main power source [Nof09]. Enabled by electrical power the second Industrial Revolution

took place in the beginning of the 20th century leading to Taylorism and mass production [Leit04]. The immense progress in information and communication technology (ICT) and microelectronics heavily affects automation technology (AUT) in a third revolution since the 1970s [Kage13]. By means of digital technology the realization of control logic has developed from hard-wired electronic components to a much more flexible implementation in software. During the last decades increasing automation degrees and steadily developed automation technologies led to significant improvements of the productivity, safety, feasibility, and quality of production [Nof09].

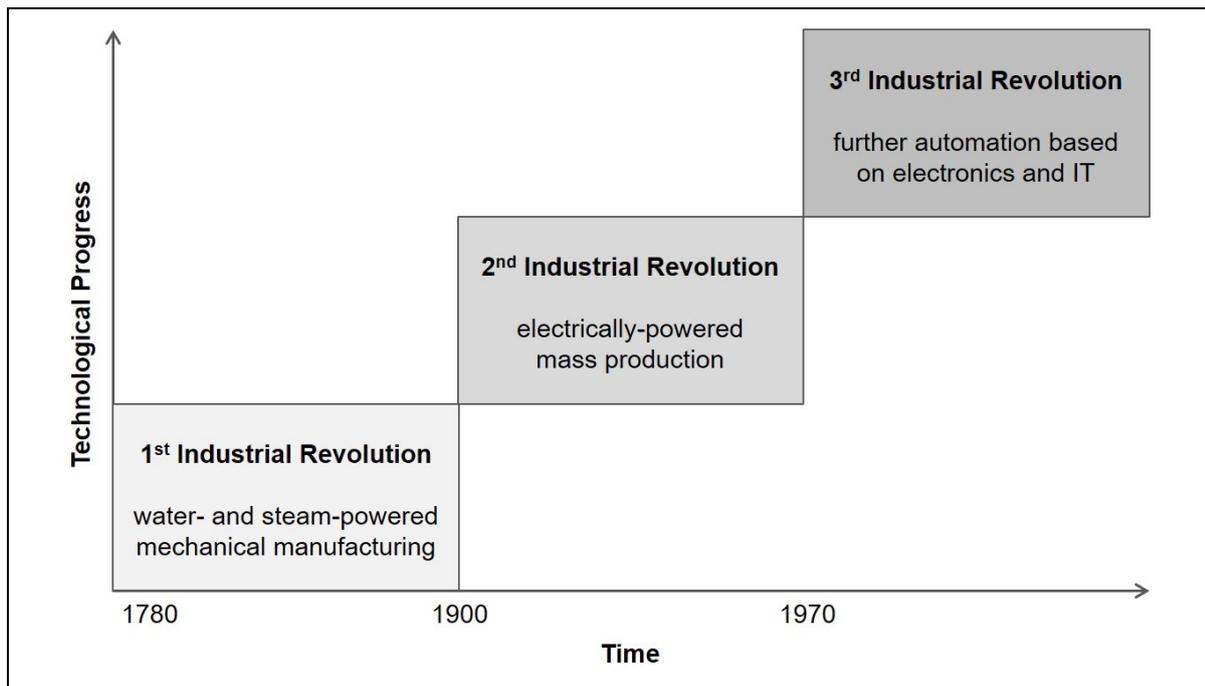


Figure 2-1: Industrial revolutions [Kage13]

In recent times, the production domain constitutes an increasingly dynamic environment for manufacturing companies, which have to deal with manifold and steadily changing demands to stay competitive. Concerning the automation of production processes, the following trends are the main drivers for future developments [Abel11]:

- **Globalization:** Most industry sectors operate internationally so that markets, business partners, and production sites over the world can emerge. However, these new opportunities also imply an intense competition. Particularly, manufacturing companies in high-wage countries like Germany are confronted with strong cost and innovation pressure.
- **Shorter product life cycles and customization:** Enabled by faster technological progress and innovation cycles, the time between product generations is decreasing. For example, the development cycles in the automotive industry have been reduced from seven to five years within the last decade [Drat09]. Simultaneously, a stronger

customization increases the amount of product variants. Manufacturing companies can gain competitive advantage by entering the market first with new products and by offering individual product features. Therefore, new or changed production processes with a higher flexibility have to be set up in shorter time.

- **Technological innovation:** The engineering of production systems, which stems traditionally from the mechanical engineering field, increasingly includes other disciplines from electronics and computer science. Besides deep knowledge in single disciplines and technologies, an increasing interdisciplinary understanding becomes important to establish new innovations. Through this technological progress the product to be manufactured as well as the production equipment become more sophisticated leading to a rising complexity of production plants and processes.
- **Shortage of resources:** The growing demand for resources and their limited quantities lead to a shortage of raw materials and consequently, to cost increase. Manufacturing companies have to react on this trend with new production strategies and technologies for the optimization of resource consumption and minimization of waste. Besides economically motivated reasons, the desire for more sustainable production affects also ethical and ecological topics.

The consequence for manufacturing companies is the need to empower their production systems to produce a high diversity of product variants, to rapidly adapt to new or optimized production processes, and to quickly react on current order or resource situations whilst handling a growing complexity of processes and equipment at the same time. Based on this, a number of requirements on automated manufacturing systems can be derived. In literature, a set of well-known terms are used to describe these. However, their exact meaning differs a lot sometimes. In the following, unambiguous definitions of the most important requirements are given for this thesis:

- **Flexibility:** Flexible manufacturing systems are able to execute different production processes or process variants in a predefined scope of action that are determined a priori by design [EIMa05][West09].
- **Adaptability:** The capability to adapt the manufacturing system concerning its hardware structure and control system due to new or changing requirements is defined as adaptability [West09]. A related term is reconfigurability defined as the ability to add, remove, and/or rearrange the components and functions of a system in a timely and cost-effective manner [Fari08].
- **Robustness and fault tolerance:** To reach a high degree of availability, manufacturing systems need to withstand the influence of disturbances or faults without essential changes in the system's behavior [Schr11][Vánc11].
- **Reusability:** The possibility to set up a new or changed manufacturing system by reusing existing engineering results, production equipment, and control procedures rises the efficiency of plant engineering tasks [Voge09a].

-
- **Agility:** The combination of the above mentioned aspects to reach effectively the target objectives both in a business and a technical dimension [Tren09].

Empowering manufacturing systems to meet these requirements in a proper way, necessitates new demands on the control technology [Sünd06]. One of the most important ones is allowing an open and transparent communication between all interacting components of the automation system. This constitutes a key concept to enable flexible, robust, and easy adaptable production processes [Pohl08]. Besides the application of automation technology itself, the planning and realization tasks play a major role to set up such automated production facilities, which constitute highly complex systems. One goal is the application of effective and productive engineering processes to lower the overall efforts for developing production systems [Walt13]. Another factor which is gaining in importance is the period of time needed to put a production system in the desired operational mode [Nof09]. By time-saving, which leads to a reduced time-to-market, an immense competitive advantage can be gained. Efficient engineering strategies lowering costs and time for realizing advanced automation degrees are required accordingly.

In summary, manufacturing companies have to handle a rising complexity concerning external demands, production technology, and engineering concepts in order to keep pace in the globalized market.

2.2 Industrial Automation Systems

Typical industrial automation systems are organized in a hierarchical structure characterized by a layered system structure containing a variety of heterogeneous devices, networks, protocols, and applications [Grob08]. The global control problem is split into hierarchically dependent sub-problems with decreasing time ranges (i.e., strategic, tactic, operational) assigned to hierarchically dependent decisional entities [Tren09]. There are significant differences between the automation of continuous and discrete production processes due to their physical conditions and the traditional separation of the domains [Mers11a]. Consequently, particular engineering tools, engineering strategies, controllers, and partly different field devices are used according to the kind of production process. Furthermore, there exist different terms describing the control tasks to automatically execute a production process. The term *manufacturing control* typically refers to the control of discrete production processes, whereas *process control* is used for continuous production processes and *batch control* correspondingly for batch processes.

A well-known representation of this hierarchical structure is the *Pyramid of Automation*, also known as CIM (Computer Integrated Manufacturing) pyramid, which comprises four layers that realize particular automation tasks (see Figure 2-2). The lowest layer (i.e., Layer 1) contains the field devices that constitute the interface between the production equipment on the shop floor and the automation system. Therefore, electrical signals are transformed

into physical values (e.g., mechanical motion, air pressure) and measured quantities are translated back to electrical signals. Thus, field devices are separated in sensors that receive information about the production process and actuators that manipulate the technical process again [Dels12]. The scope of complexity of different types of field devices can considerably differ ranging from simple devices with one simple output like a proximity switch to servo drives with thousands of inputs and outputs (I/Os).

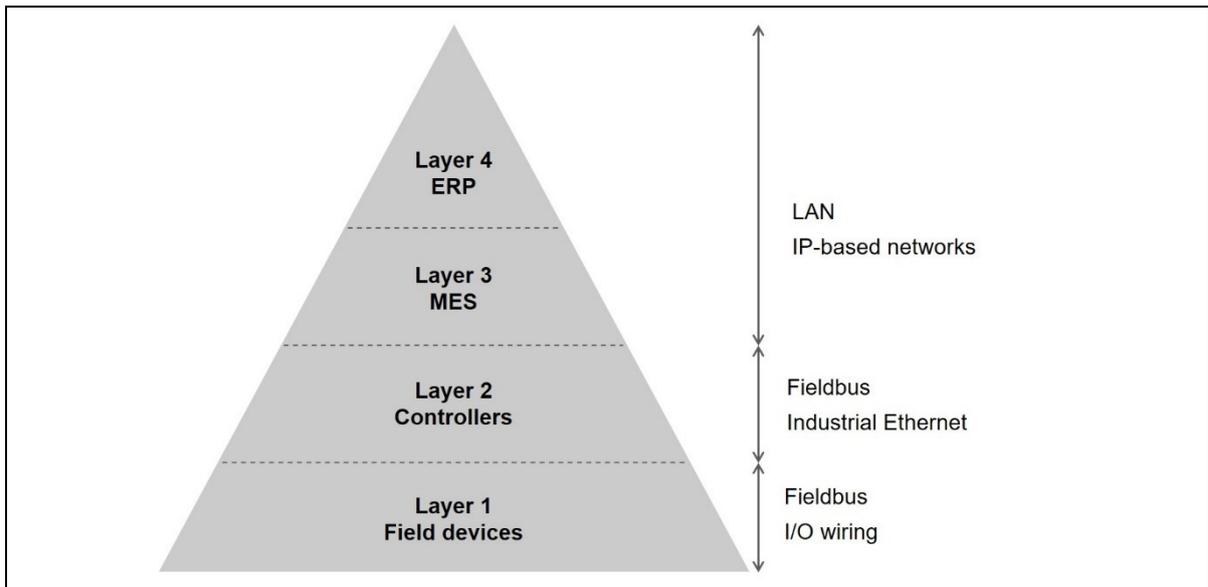


Figure 2-2: Pyramid of Automation [IEC08]

Layer 2 comprises different types of controllers that control and monitor the actions of the field devices to execute a certain technical production process (see Figure 2-3). The most widely used technology used here are Programmable Logic Controllers (PLCs) that are particularly applied for manufacturing control [Sünd06]. They execute a control procedure implementing the dedicated process sequence and the commands to communicate with the actuators and sensors on the shop floor of the production plant. The production process is usually subdivided into several disjunctive manufacturing steps, each executed by a production cell which is controlled by a PLC [Math09b]. For complex machine functionality, other controllers are used subordinated to the PLC; particularly for Motion Control (MC), Computerized Numerical Control (CNC), or robot control. For process control applications Distributed Control Systems (DCS) are common instead of PLCs. DCS organize a set of distributed controllers, which are typically feedback controllers, and often provide additional visualization options. Since their technology evolved very similar as PLCs over the years, the differences to PLCs in their functionality have become less straightforward [Nof09]. Besides the well-established PLCs and DCS, more and more Industrial Personal Computers (IPCs) gain in popularity [Voge09a]. They often run a Soft PLC which acts as a conventional PLC running on a PC platform. Often other systems are also included in this layer to supervise and monitor the production process like HMIs (Human Machine

Interaction) and SCADA (Supervisory Control and Data Acquisition) systems. Typical are also superior PLC levels where cascaded PLCs exercise local area control of various production cells and interact with the controllers beneath [Bolt09].

The connection between the controllers among each other and the production equipment is realized by a direct wiring to their I/Os or by field buses using a wide variety of communication standards [Seit08]. The integration of controllers to higher layers is typically realized with the communication standards OPC (OLE for Process Control), OPC UA (OPC Unified Architecture), or conventional Ethernet and TCP/IP protocols [Seit08].

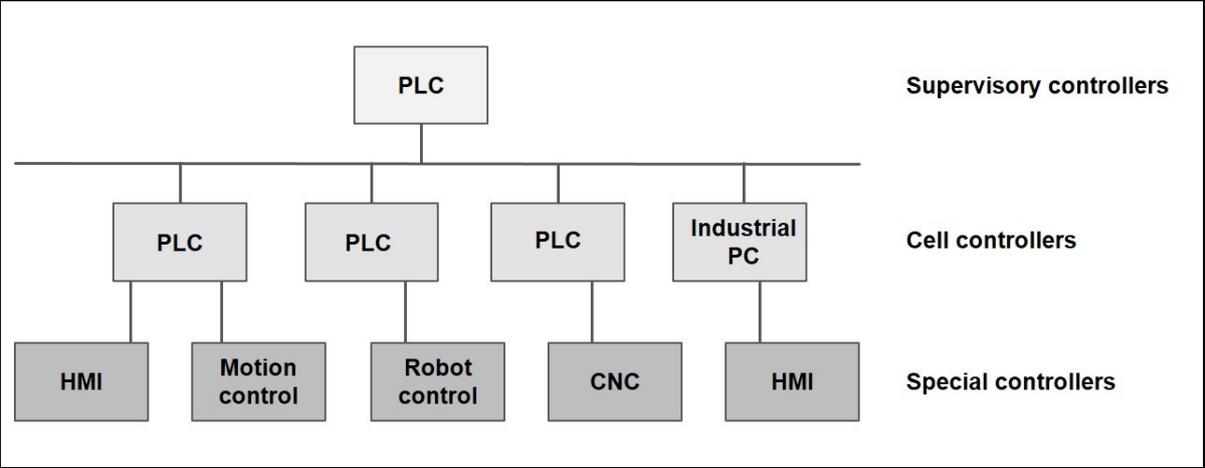


Figure 2-3: Hierarchy of controllers (layer 2)

Manufacturing Execution Systems (MES) can be found on the third layer and represent the link between enterprise systems and the controllers on production level [Souz08]. Their major task constitutes manufacturing scheduling, which contains translating production orders into concrete control commands for the controllers on layer 2 and assigning manufacturing resources [Shen06]. Additionally, a MES supplies and processes information from the automation systems to superior business management systems [Math09b]. These are located on the highest layer, i.e., layer 4, which is particularly characterized by Enterprise Resource Planning (ERP) systems that organize the production planning among other things. They link all aspects from the bill of material required from suppliers to incoming custom orders and utilize different algorithms in order to efficiently schedule the production programs that are passed to the MES [Nof09].

According to the IT technologies that are used in the respective layer, two main areas can be identified with fundamentally different requirements. The business and production planning tasks on layer 3 and layer 4 are realized with wide-spread and cost-effective office IT systems and standard IT applications like PCs, Ethernet, and TCP/IP [Bang08]. On the two lower layers specific IT technologies are used because of the special characteristics of the environment they control [Marc08]. Thus, automation technologies are realized with specific equipment, like PLCs, field buses, and I/O devices, that are optimized for the high

requirements regarding reliability, availability, response times, safety, etc. [Bang08]. This leads to highly vendor-dependent properties of the equipment so that control equipment vendors usually offer engineering tools for programming and configuration based on proprietary software architectures and programming techniques [Esté12].

2.3 Control Engineering within the Production Life Cycle

The development of industrial control systems is one of numerous tasks that are necessary for building an automated factory system. *Control engineering* is one essential task where the control applications are developed and the automation system is configured. To comprehend the typical characteristics of control engineering, its position in the overall factory planning process and its main tasks are described. Afterwards, today's situation of control engineering is discussed and its deficits are assessed.

2.3.1 Factory Planning

Factory planning is the systematic, objective-oriented process for planning a factory. It is structured into a sequence of phases, extending from the setting of objectives to the start of production, also including supervision of the realization [VDI11]. It constitutes a part of the whole corporate planning so that dependencies to other planning processes exist, such as the product planning [Berg06].

Generally, factory planning processes are structured according to two principles: the temporary dimension in terms of planning phases and the functional dimension in terms of functional systems of the factory. The factory planning process is subdivided into seven sequential phases (see Figure 2-4) [VDI11]:

- **Setting of objectives:** Tasks relating to factory planning are clarified including the analysis of the corporate and factory objectives and they are structured into work packages.
- **Establishment of the project basis:** The data and information required for the planning work are gathered and generated.
- **Concept planning:** The factory is planned as a totality with the objective to specify a feasible factory concept which best meets the factory objectives. The result is a rough layout of the factory concept and the factory building.
- **Detailed planning:** The selected factory concept is planned out comprising a detailed description of the individual elements and the specification of services, which are required by suppliers.
- **Preparation for realization:** The award of contracts to suppliers and the planning of the implementation are organized based on the specifications of the factory elements.
- **Monitoring realization:** Securing and documenting the correct and proper construction of the building, its outdoor facilities, the factory equipment, and the expansion of personnel specified in the plan.

- **Ramp-up support:** The factory is put into operation including the ramp-up to the level of its intended performance and the evaluation with reference to the factory objectives.

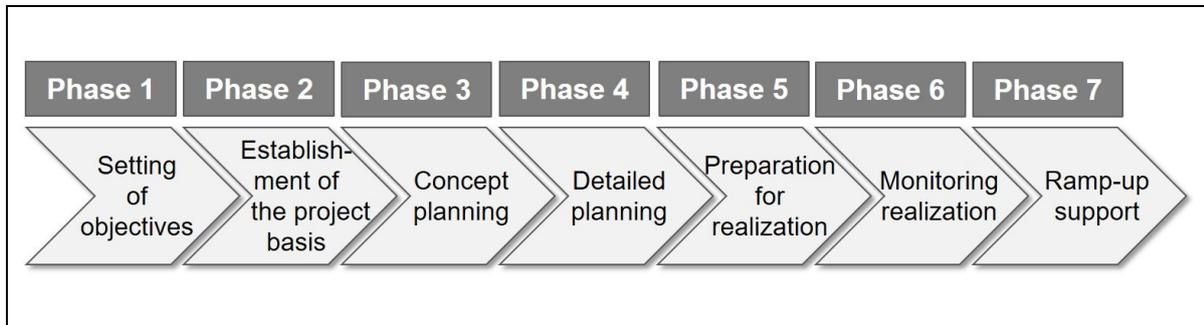


Figure 2-4: Phases of the factory planning process

The factory planning process comprises the design of numerous different planning details that can be grouped to a number of functional systems [Pawe08]: procurement, production, logistics, technical support areas, internal organizational areas, distribution, and disposal. Each functional system depends on the following planning aspects characterizing the overall production process: product, technology, equipment, organization staff, and finance. The functional system and the planning aspects can be arranged in a matrix so that the exact design area of a certain planning task can be visualized (see Figure 2-5) [Pawe08].

Different types of factory planning can be derived from the life cycle of a factory [VDI11]. *Development planning* is the planning of a completely new factory on a so-called greenfield site. Modifications or extensions of existing factories are realized during *replanning* or *reconfigurations*. The shutdown of a factory is performed during *demolition* and *revitalization* applies when an industrial wasteland site is made available again. Since the production domain is an increasingly dynamic environment—with changing products, product variants, order margins, etc.—reconfiguration tasks are increasingly becoming important [Nof09].

For an efficient planning of an agile factory, methodologies are required that define the procedure steps and instructions to develop a suitable solution in the form of a plan of the factory and the implementation of the plan to reality [Berg06]. Therefore, three building blocks are necessary [Schn92]:

- **Modeling and structural concepts:** Rules that define how information is represented as models (i.e., meta-models).
- **Design concepts and architectures:** Fundamental design blueprints and alternative solutions (i.e., reference models/architectures).
- **Procedures, methods, and tools:** Specification of the sequence and content of planning steps and support of single planning tasks with libraries, standards, mapping rules, etc.

2.3.2 Production Engineering

Production engineering or *production planning* is a subset of the whole factory planning comprising all activities to specify, design, realize, modify, and start-up of the technical production plant [VDI08]. Since the details of the technological realization of the plant depending on the factory concepts are designed here, production engineering usually starts during the detailed factory planning phase. According to the functional classification of planning tasks, it comprises the functional system “production” (see Figure 2-5). Since the design of the plant depends heavily on the product itself and the product quantity, the planning aspect “product” has to be considered. Furthermore, the aspects “technology” and “equipment” are determined by designing the production equipment and the technologies used for realizing the desired operation mode like automation systems.

As input of plant engineering mainly act two sources that can be regarded as results of a preceding analysis phase before the production system is planned: The product design and the results of the factory planning phases beforehand including descriptions of required production workflow and a rough layout of the required production lines [VDI08]. Production engineering requires the cooperation of different disciplines, particularly those involving processing, mechanical engineering, electrics, and automation engineering [VDI10].

		Planning aspects					
		Product	Technology	Equipment	Organization	Staff	Finance
Functional systems	Procurement						
	Production						
	Logistics						
	Technical support areas						
	Internal organizational areas						
	Distribution						
	Disposal						

Figure 2-5: Functional classification of production engineering within factory planning

The production engineering process is also structured in several sequential phases (see Figure 2-6) [Drat09][Schm05][VDI08]:

- **Rough planning:** The rough workflow logic comprising a sequence of the production steps is described in process plans that are usually depicted as Gantt charts. Based

on this, a detailed system concept considering the required production equipment is developed.

- **Detailed planning:** The specifics of each individual workstation are worked out in respect of several engineering disciplines, which are usually performed sequentially. Today's typical sequence starts with the design of the mechanical structure including pneumatic and hydraulic systems, followed by a planning of the circuit diagrams for the electrical components. It is concluded by the development of the control systems during control engineering.
- **Realization:** The individual components of the production facility are assembled and built up to form the real production plant based on the planning results.
- **Commissioning:** The correct wiring and functional capability of actors and sensors is checked. After that the interaction between mechanics, electrics, and the control software is tested. Usually, adjustments and tests are iteratively executed to establish and ensure the required behavior of the plant. Finally, the completely assembled and mechanically reviewed production system is put into operation.
- **Start-up of production:** The approved plant is set into a stable operational mode according to the desired quantities and cycle times.

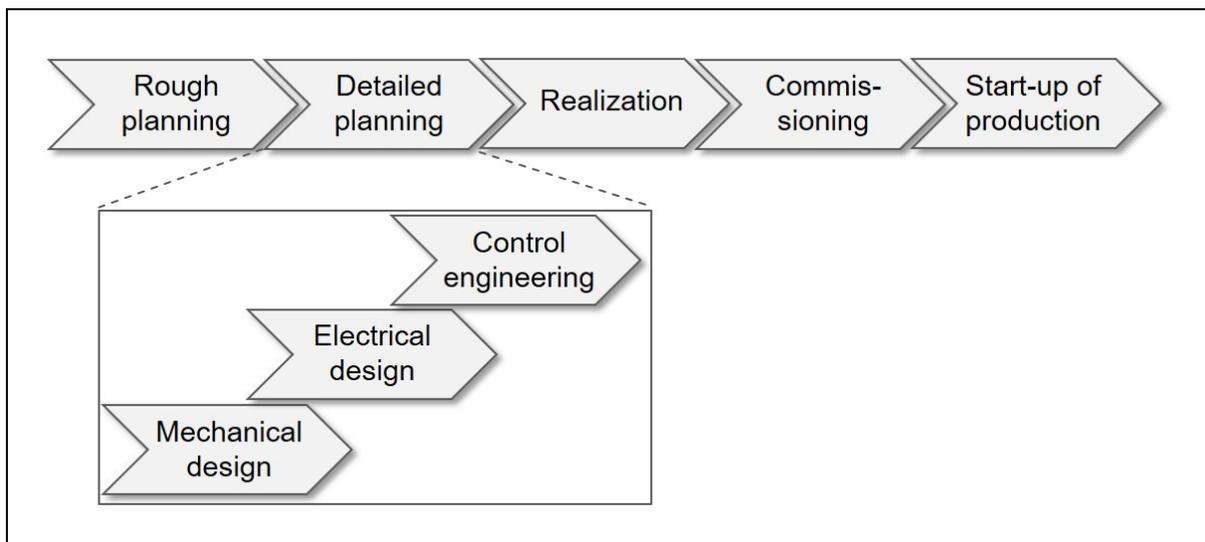


Figure 2-6: Phases of production engineering

Usually, several stakeholders are involved in a production engineering process [Schm05]. The most obvious one is the manufacturing company requiring a suitable production plant to produce a designated product and operating the plant. The realization and often already the detailed planning are executed by general contractors that are specialized in production facility development. Besides this first-tier supplier, other second-tier and third-tier suppliers are commissioned to provide special services. Both manufacturing companies and suppliers experience production engineering as a crucial task which is under a growing competition pressure regarding cost and time [Fay09][Schl08a].

A successful execution of a production engineering process depends more and more on its efficiency. This is mainly driven by the following three demands [Frag09]:

- **Quality assurance:** Meeting the quality specification is fundamental to successfully realize engineering projects.
- **Shortening of development time:** Shorter development times enable an earlier time-to-market for new or changed products and are essential to gain a competitive advantage over competitors. Furthermore, the time needed for reconfiguration and the preceding restart tasks have to be minimized to keep downtimes low.
- **Reduction of efforts:** The reduction of development time accompanied by reduced efforts already leads to a lowering of costs. Besides this, cost reductions are applied to all cost drivers in terms of production equipment, development systems, and personnel.

Two measures supporting these demands during production engineering are parallelization and lowering efforts for the execution of planning tasks. Today, the individual engineering disciplines within the detailed planning are normally executed sequentially. However, slight temporal overlappings are already found in practice and further parallelization is desired in order to save overall development time. The efforts that are needed for executing the individual planning tasks are heavily influenced by planning methods and tools supporting their application. Altogether, challenges to realize these measures have an organizational dimension concerning the optimization of the engineering methodology and a technical dimension asking for a flexible and reusable system architecture [Frag09].

2.3.3 Control Engineering

The last task of the detailed planning is the control engineering after the development of the workflow logic and the mechanical and electrical subsystem [Thra05]. It comprises the software development for controlling and monitoring the production process executed by PLCs, HMIs, SCADA systems, etc. and the configuration of the communication systems like field bus systems [Epl10]. The major part constitutes the development of the control software implementing the correct manufacturing control strategy, normally realized as PLC programs [Marc08]. The engineering of these control procedures is a key factor during the development of automated production systems. It is the connecting link for a correct cooperation of all disciplines and the last step before commissioning [VDI10].

A PLC is a special form of microprocessor-based controller that uses programmable memory to store instructions and to implement control functions [Bolt09]. Therefore, various logic, sequencing, timing, counting, and arithmetic operations are used to realize programs for controlling processes. Through its I/O system a PLC is able to connect to a certain number of machines and to control them in hard real-time [Paol05]. The well-established standard IEC 61131-3 defines a model and a set of five programming languages for the development of industrial automation software [IEC13][Thram11a]. It defines that a PLC

program is organized in several modules called program organization units (POUs) that can call each other with or without parameters. There exist three types of POUs [Tieg09]: Function (FUN), Function Block (FB), and Program (PROG). A POU may consist of other POUs and multiple expressions in one of the defined programming languages [Math09a]. Furthermore, the standard defines data types and standardized functions like timer, counter, and functions for type conversion. The IEC 61131-3 defines five PLC programming languages to program the POUs: Instruction List (IL), Ladder Diagram (LD), Structured Text (ST), Function Block Diagram (FBD), Sequential Function Chart (SFC).

PLC programs for manufacturing control applications comprise typically two main parts: the implementation of the process logic and the integration of the field devices. The process logic implements the required sequences of operations in order to execute the desired process sequence. Usually, several routines implement different modes of the production equipment. Besides the logic for the normal operational mode, routines for start-up, shut-down, safety functions, and fault handling are necessary [Gütt08]. Within the process logic the functions of the production equipment have to be accessed in the respective process steps. Therefore, functions of the field devices are used that are typically implemented in individual FBs. Inside a FB the functionality of a field device has to be implemented, which is normally described in their instruction manuals. Therefore, the programmer has to be well versed with the functionality of the field device and the respective functional roles of its individual I/Os in order to obtain the intended functionality [Yu11].

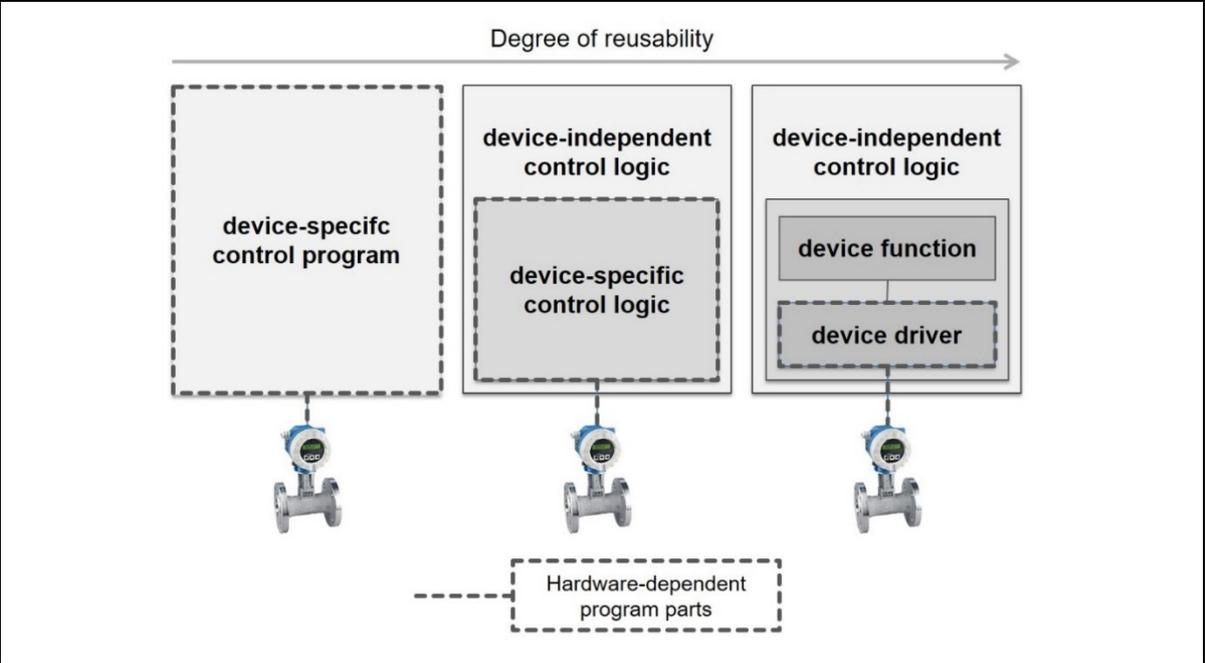


Figure 2-7: Reusability concept for PLC programming

Usually, manufacturing companies use field devices of the same type multiple times in a production plant and deploy similar production processes or sub-processes for related

products or product variants. This makes it desirable to develop control programs with a high degree of reusability. A widely-used concept to promote reusability is the separation of device-specific program parts from device-independent control logics (see Figure 2-7). Therefore, the control logic for each field device is implemented in an own FB. In case of an exchange of devices, the high-level control logic remains unaffected and hence, modification efforts are kept low. To further increase the reusability of device FBs being suitable for similar field devices of the same device category, the device logics can be split up again in a common functional part and a hardware-dependent device driver. The device driver connects the generally implemented device functions with the specific I/Os of the respective field device type. Initially, the programming effort might be higher to apply these reusability concepts for developing modularized PLC programs. Despite this fact, the more often a program module is reused, the lower is the total programming effort.

Besides the programming concept, the degree of reusability of PLC programs heavily depends on the chosen target platform. By means of the wide distribution of the IEC 61131 standard, the transfer of PLC programs from one PLC system to another one is at least partially possible. An additional task of PLC engineering is the bus and hardware configuration and the assignment of the variables of the I/O image table to internal variables of the PLC program. In contrast to the programming languages, these configuration tasks are not standardized and heavily dependent on the respective PLC and its configuration tools.

2.3.4 Situation of Control Engineering Today

Today, the control engineering discipline is now more than ever a crucial phase of the whole plant engineering resulting from a growing amount of software. A survey of the Mechanical Engineering Industry Association VDMA from 2012 indicated that 30% of the production costs for engineering products already accounts for IT and AUT with a current growth rate of 11% [VDMA12]. One reason for this situation is the strongly risen amount of machine functionality implemented in software because of the higher flexibility and adaptability of software compared to pure electromechanical solutions [Rein07]. An example for the replacement of mechanics through software is the use of coordinated servo axes instead of mechanical cam disks for the realization of automated motion sequences [Walt13]. Additionally, the amount of software-based automation functions has significantly increased like monitoring, HMI, or complex control functions. These basically cannot be realized by mechanics to fulfill rising demands on system flexibility, performance, and cost reduction [Fant11][Voge09a]. Since these trends will probably continue during the next years, the importance of an efficient control engineering providing software with a high quality will further increase [Voge09a].

Despite its great significance, control engineering is located as the last planning task depending on all previous planning disciplines. Due to the historical development of

production technology, control engineering is rather underrepresented in the production life cycle, which still is dominated by the mechanical design [Zäh05]. Thus, an early consideration of control tasks and the linking of control engineering with the other planning disciplines are missing [Wüns07]. Additionally, control engineering is usually executed under a great time pressure. The delays that occurred during the previous planning disciplines have to be compensated during the control engineering in order to still meet the time schedule. Thus, the control software has often a low degree of maturity when commission starts with the consequence of further “development on site” [Drat09]. An investigation for the German Association of Machine Tool Builders (VDW) showed that 90% of the commissioning time is used for delays and activities related to electric and control devices and that 70% of this time delay was associated with errors in control software (see Figure 2-8) [Rein07][VDW97]. The high impact of erroneous software can be related to an inadequate consideration of software engineering within the plant engineering process regarding its increased importance [Kief08]. Due to shortening development cycles and thus, decreasing time periods for control engineering, the situation will be even aggravated in future. These deficits are mainly due to organizational issues that consider the role of control engineering within the production engineering process.

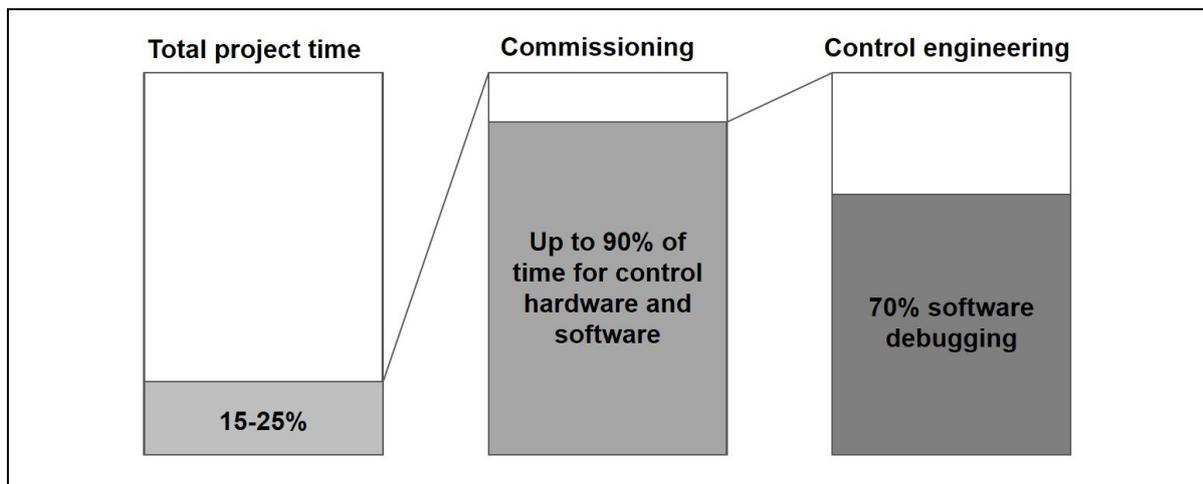


Figure 2-8: Contribution of control software to project delay [Rein07]

Besides this, the applied methods and technologies for the development of control software are responsible for its quality and particularly for the degree of its reusability. Today, the development of well-structured control programs providing clarity is often hindered by a lack of time and missing development guidelines, like the application of the presented reusability concepts (see Chapter 2.4.3). There exist no standards for the design and realization of industrial production processes so that control engineers usually rely on best practice solutions developed over the years [Math09a].

Another crucial point constitutes the PLC programming techniques according to the IEC 61131 standard. Since control engineering was originally realized with hard-wired electrical

circuits, it is still associated to the domain of electrical engineering and mainly executed by engineers and technicians. Thus, the programming languages are very low-level and the programming takes place on signal level with poor abstraction mechanisms [Zühl10]. The low programming level requires a high degree of expert knowledge about the functionality of the field devices and how the provided functions can be accessed via the I/Os of the device [Yu10]. Additionally, profound knowledge about the hardware, the communication technologies, and the development environment of the PLC is necessary so that many hardware-specific details have to be considered. However, the FB concept already enables the encapsulation of application parts in FBs to modularize control applications and foster the reuse of application parts. Despite the support of some high-level programming features, decisive features of object-orientation such as inheritance are still rarely used [Zoitl09b].

Although the IEC 61131 standard is well-established, a reuse of PLC code between different PLC systems is severely restricted by the use of vendor-specific extensions or only partial support of the IEC 61131-3 and proprietary engineering tools [Zoitl09b][Marc08]. Even if PLC manufacturers declare their systems compliant with the IEC 61131-3 standard, usually far-reaching differences exist between the implementation of PLC code in PLCs of different vendors [Fant11]. This is due to vendor-specific programming functions (e.g., object-oriented concepts, pointers) and hardware-specific implementation details like the declaration of variables and I/O configurations which are not part of the standard. Besides the PLC programming, the configuration and parameterization of field devices is typically done via their own special engineering software which leads to increasing engineering costs [Mers10].

Moreover, the low-level programming induces a significant gap between process planning in previous planning phases and process implementation [Thra11a]. Input for the control engineering phase are usually very rough process plans defining the sequence of the control logic as simple graphical representations [Leit06]. These planning results cannot be used directly due to their insufficient detailing degree and the divergent representation of information. Hence, the process model is manually detailed and transformed to the corresponding implementation. Thereby, code is usually written from scratch or existing FBs from previous projects are individually adapted according to the “Copy and Modify” principle [Voge09a]. Today’s typical situation is the direct design of control procedures in the IEC 61131 standard, specifically for the respective target platform leaving most of the work to the control code developers [Frey11].

In sum, the mentioned circumstances describe an inadequate situation of control engineering today, which is characterized by the following drawbacks [Favr06][Sünd06]:

- Rigid monolithic architectures of control programs and variable structures

-
- High complexity of control programs due to insufficient clarity and abstraction mechanism inducing error-prone code
 - Limited reusability and adaptability of poorly structured and hardware-dependent code
 - Lack of an integrated information flow leading to an inadequate use of results from previous planning phases
 - High efforts for the initial programming, maintenance, and adaptations to existing programs

Thus, control engineering has a huge potential for improvement to increase its efficiency leading to less efforts and shorter commissioning and overall project durations. This action field comprises two main aspects:

- Early organizational integration of control engineering with other planning disciplines enabling integrated information flows and a stronger consideration of control aspects.
- Methods and technologies for the development of control programs supporting modularization and abstraction to promote reusability, comprehensibility, and adaptability.

3 Design Concepts for Distributed Control Systems

Production systems are increasingly expected to be more agile due to frequently changing market demands and shorter product life cycles [Feld09]. A promising approach to fulfill rising demands towards flexibility, adaptability, and reusability is the idea of modular and collaborative production systems. These would consist of distributed, autonomous, and reusable units, which operate as a set of collaborating entities [Mend08a]. Besides mechanical aspects, particularly the properties of the automation system are affected by the realization of such modular production systems. Therefore, distributed and modular control components are required that interact in order to accomplish control activities [Mend08a]. In this chapter, the trend towards a higher degree of distribution in the area of industrial automation is investigated and design concepts for the realization are reviewed.

3.1 Trends in Industrial Automation

Current requirements for industrial automation systems will entail far-reaching changes to the traditional properties of industrial automation systems portrayed as the automation pyramid (see Chapter 2.2). The main factors impelling this change are a rising amount of automation tasks and a heavily increasing networking through interconnecting different types of automation devices [Kage13]. The growing automation degree leads to a rising number of IT applications supporting the production process in different ways (maintenance, optimization, HMI, positioning systems, etc.). Additionally, on the lower automation layers the functions of field devices and, consequently, control procedures are getting more complex and increasingly diverse. Since automation devices exchange information and thus, run dependently on each other, the ability of a seamless connection of different automation devices is required. The term *horizontal integration* describes the interconnection of automation devices on the same automation level, whereas *vertical integration* means to link automation systems on different automation levels [Gerb14].

Since the 90's it becomes more and more important to link PLCs with various field devices on lower automation levels and with production planning on higher automation levels [Seit08]. At the same time, current manufacturing control systems are often fragmented and isolated from higher level business systems [Jamm05a]. Usually, connections spanning several levels are realized via inflexible proprietary communication technologies, which results in an integration gap between field- and business-level [Feld09][Karn07]. Although standardized IT technologies are already common on business level, connectivity is mainly restricted through the use of specialized hardware and software on the manufacturing layer [Nguy08]. The mixture of different technologies leads to numerous breaks in the

communication paradigm due to a lack of standardized interfaces between these two worlds [Grob08][Math09a].

To overcome these drawbacks the interaction of various automation devices and applications needs to be managed by bridging the integration gaps to leverage flexibility and interoperability [Grob08]. The vision of future automation systems can be described as a distributed system capable of integrating a variety of heterogeneous devices into an interoperable and easily configurable network [Jamm05a]. Within this network automation devices act as intelligent, autonomous, and collaborative entities to provide flexibility and automatic reconfigurability [Mend11][Mend08b]. Functionality that is traditionally programmed within the PLC or DCS will be carried out by these devices directly [Bang09]. In the research context, they are often referred to as *smart*, *intelligent*, or *mechatronic devices* (see Chapter 4.11).

The biggest driver for transferring this vision into reality is a strong IT-driven change in automation technology by making use of modern ICT [Mend11]. Increasingly, automation platforms, which have been fully proprietary systems in the past, use common IT technology today [Nof09]. Thereby, two major developments in information technology are decisive: the progress in semiconductor technology and the propagation of internet technologies. The rapid development of semiconductors led to low cost, high-performance computational components paving the way for the dissemination of embedded systems. The trend is still continuing towards significantly enhanced functionality, complexity, scalability, and connectivity enabling wired and wireless networks of large-scale distributed real-time embedded systems [Pere07]. The success story of the internet during the last decades provides software technologies dealing with highly distributed information [Paol05]. Adapted by automation technology, common internet technology like TCP/IP and Ethernet are becoming widely accepted among automation systems.

Today, several research initiatives try to formulate frameworks and solutions to raise automation technology to the next level by adapting these IT trends. One of the most popular initiatives originating from Germany is the high-tech strategy Industrie 4.0. Its name declares the fourth industrial revolution by introducing the Internet of Things and Services into the manufacturing environment to create networks of physical devices incorporating the entire manufacturing process [Kage13]. The propagated achievements are profitable production of highly individualized products, dynamic business and engineering, optimized decision-making, and creation of novel business models [Kage13]. Similar concepts outside the German-speaking world exist like the U.S. initiative Advanced Manufacturing Partnership (AMP) announced in the USA in early 2012, which is followed by Nationwide Network for Manufacturing Innovation (NNMI) program, the Industrial Internet consortium (IIC) founded by General Electric, Intel, Schneider Electric, IBM, and SAP, and the EU

funding program Horizon 2020 initiated 2014 under the topic Factories of the Future [Maju11][Herm15].

One general vision of these initiatives is the conversion of the traditional automation pyramid to an automation network of interacting and independently acting devices, which are often referred to as *smart devices*. The mentioned IT trends drive this change since they constitute the foundation to equip more and more production equipment with some computational power enabling the realization and dissemination of such next-level automation devices [Când09b]. As soon as hardware devices have their own embedded computers, each device can implement various programmable control functions and also provide some network interfaces and memory capacity [Vyat03].

In the scope of Industrie 4.0 these ideas are covered under the term *Cyber-Physical Systems* (CPS) describing a network of physically distributed embedded sensors and actuators equipped with computing and communication capabilities [Lee15]. The individual entities are capable of autonomously exchanging information, triggering actions, and controlling each other independently [Tabu06]. The scientific basis for describing the interaction of various independent computing devices within a CPS stems from the research field distributed systems.

3.2 Distributed Automation Systems

In computer science, systems with various cooperating software components are referred to as *distributed systems*. Such computing systems are subject of the research field of the same name, which constitutes the scientific basis for CPS. According to [Tane06] a distributed system is defined as a collection of independent components (i.e., computers) that appears to its users as a single coherent system. This definition implies the interconnection of autonomous and interoperable components and their collaboration to achieve a common objective. The main goals of distributed systems are making resources easily accessible, enabling distribution transparency, openness, and scalability [Tane06].

The application of the principles of distributed systems to automation is considered as a promising approach for handling the increasing complexity and dynamics of automated manufacturing systems [Vall09]. A distributed automation system can be defined as an assembly of logical and, where necessary, spatially divided modules which cooperate to achieve automatic control functions and communicate over a network [Fran11]. Since the 1990s, various research activities took place in this topic. Most of them propagate an increased agility and reactivity by means of bringing the intelligence and autonomy closer to the production equipment [Thom12].

To achieve the expected benefits, the adequate organization of the system as a collection of relatively small and easily replaceable or adaptable components is necessary [Tane06]. Therefore, a distinction is made between the *software architecture* and the *system*

architecture of a distributed system. The software architecture defines the logical organization of distributed systems into software components. The realization of a distributed system requires a system architecture to instantiate and place the software components on computing machines. As a result, the design of distributed systems is mostly about the layout of software components and the way they interact with each other. Their architectural style can be hierarchical, fully heterarchical, or semi-heterarchical (also called hybrid) [Tren09]. A hierarchical system is multi-tiered in several control levels distributing the decision-making to superior levels. In contrast, a fully heterarchical architecture promotes control by distributing each decision capacity to autonomous entities, without any centralized decisional control [Thom12]. The hybrid or semi-heterarchical architectures comprise both characteristics of hierarchical and heterarchical relationships by adding the interaction between modules at the same hierarchical level [Leit09].

Today, control systems are typically hierarchical systems containing several automation levels (see Chapter 2.2). However, the individual automation functions are realized as centralized solutions characterized by a single decision node [Leit09]. For manufacturing control, the PLC concentrates all processing functions for executing the technical manufacturing process leading to complex control programs (see Chapter 2.3.4). Applying the idea of a distributed control system, the control software can be split up in several software components so that a complex control problem can be divided into several smaller ones. By dividing the overall automation task in several partial problems the handling of the rising complexity of the automation environment is supported, where single components can be developed more independently of each other [Mers10]. Additionally, software components enable an application of the Black Box principle where complex or specific code can be hidden to improve scalability and clarity. Instead of re-programming large monolithic control procedures, the individual software components can be easily rearranged and accessed via their interfaces [Grob08]. This saves efforts, when a system needs to be reconfigured or extended, and reduces the complexity so that the control software is easier to adapt and to maintain [Jamm05b].

3.3 Concepts for Distributed Control Architectures

The development of distributed control applications to enhance the complexity, flexibility, scalability, and reconfigurability of control software is a promising approach that reached considerable popularity in research [Leit09]. For the realization, a concrete design concept comprising design guidelines, behavior models, best practices, etc. are necessary. Three distributed control concepts that have gained relevance for manufacturing control applications are the IEC 61499 standard, Multi-agent Systems, and Service-oriented Architecture. In the following, their basic properties and existing applications for manufacturing control are presented.

3.3.1 IEC 61499

Basics of IEC 61499

The international standard IEC 61499 defines an architecture for distributed controllers and guidelines for its implementations [IEC05]. The basic idea is to distribute control applications on various cooperating devices that today are usually executed in one centralized controller (i.e., a PLC) [Ivan09]. Thereby, the control application is divided in several FBs that can be deployed on different platforms.

Based on the well-established PLC standard IEC 61131 the FB concept in IEC 61499 is expanded with event-triggered behavior. Besides the common data I/Os, FBs also comprise event I/Os that affect how and when data is processed (see Figure 3-1 left). By connecting the event inputs and outputs of FBs the execution sequence is specified explicitly [Zoitl09b]. Thus, a FB is divided in a head capturing the dynamics and the event ports and a body comprising the functionality and the data ports [Thra13]. An incoming event triggers the execution of the FB functionality by means of algorithms, which produce output events and data in respect of the current input and internal data.

The standard defines three types of FBs: Basic FB (see Figure 3-1 middle), Composite FB (see Figure 3-1 right), and Service Interface FB. The internal behavior of a Basic FB is described by a Moore automaton, which is called Execution Control Chart (ECC). An incoming event can effect a state transition of the ECC so that a new state gets active and its associated algorithms are executed. The algorithms may be programmed in any programming language, including the languages of the IEC 61131, but also Java, C++, and others [deSo10].

An application is built by interconnecting FBs to a Function Block Network so that the execution order is explicitly specified [Ivan09]. Thereby, event ports can be exclusively connected to other events ports and accordingly, data ports just to data ports. By wrapping a function block network to a Composed FB, applications can be designed in a hierarchical way and new functionality can be generated by aggregating already available functionality. Service Interface FBs represent the interface to sensors and actuators by containing device specific execution control.

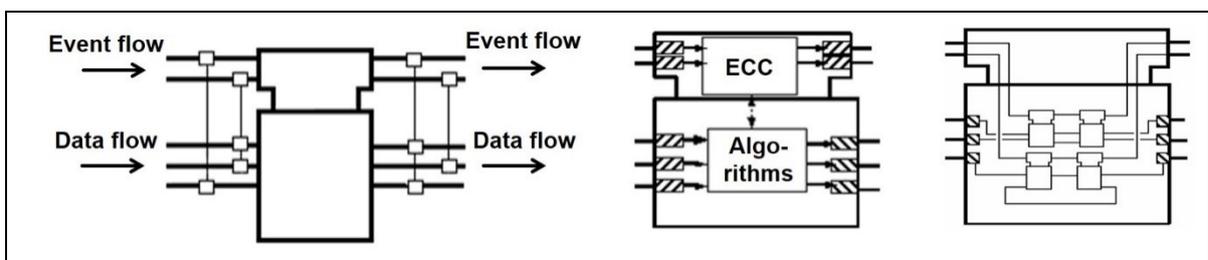


Figure 3-1: Inputs and outputs of an FB (left), Basic FB (middle), and Composed FB (right) [Chri12]

The application design is done by interconnecting the respective FBs in a platform independent way represented in the Application Model. To execute the application later on a distributed network of control devices, the properties of the computing platforms have to be taken into account [Zoitl09b]. The control devices of the network are modeled in the Device Model. Each device can comprise Resources that are functional units with an independent controller. The deployment of the control application is depicted in the Distribution Model, where each Function Block is assigned to a Resource so that connected FBs of the same application can run on different devices (see Figure 3-2).

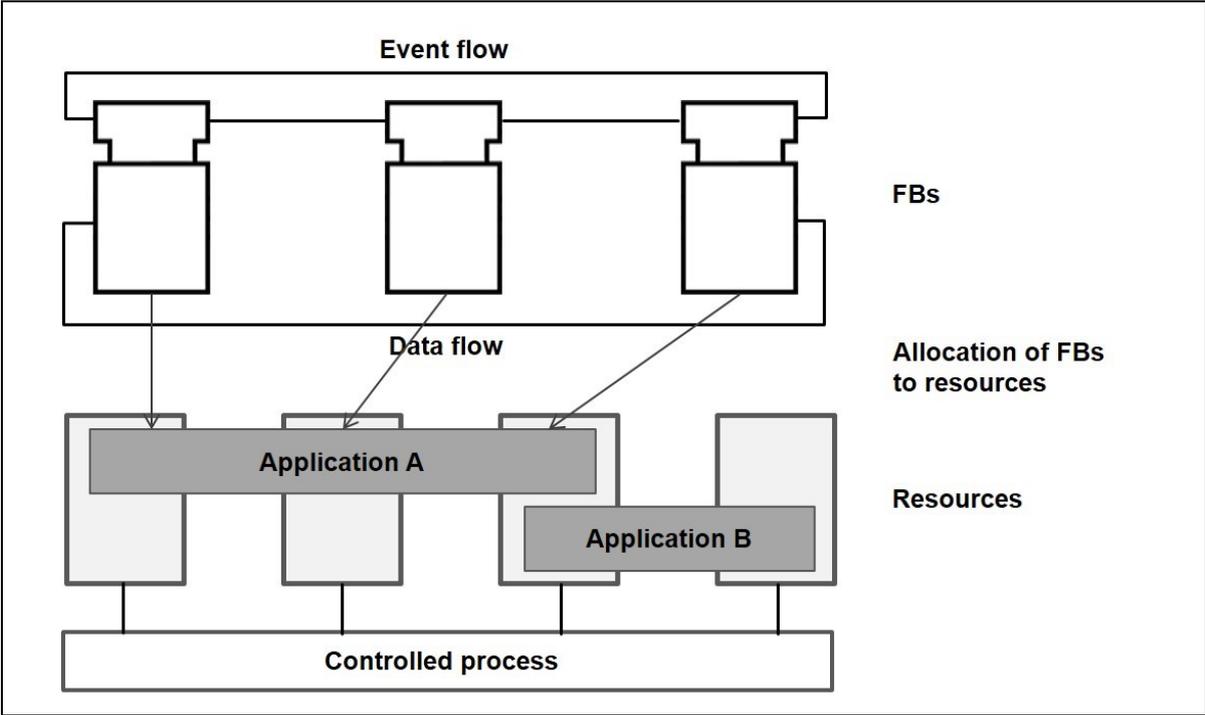


Figure 3-2: FB Network and distribution of FBs on devices [Chri12]

There exists several engineering tools supporting the development of IEC 61499 applications [Preu11][Vyat11]. A commercial tool is ISaGRAF which combines IEC 61499 and IEC 61131 development. Two IEC 61499 development environments for Java implementations are the wide-spread Function Block Development Kit by HOLOBLOC Inc. and Rockwell Automation and the open-source software tool FBench Project. Another open-source initiative provides the 4DIAC-IDE engineering environment based on Eclipse and the FORTE runtime environment for small embedded devices implemented in C++. The CORFU Framework provided by the University of Patras uses Borland’s Delphi IDE for implementation. Despite the number of different tools, most automation projects using these tools have just been applied in academic and research labs because an acceptable level of maturity for industrial use is not reached yet [Vyat11].

IEC 61499 Based Manufacturing Control

Since the IEC 61499 aims on automation as application field, there exist a great number of research activities concerning manufacturing control. A good overview of the state of the art is given in [Vyat11] and [Thra13]. However, the adaption of IEC 61499 in industry-related projects and by control system vendors is rather low to even nonexistent [Mend11]. This is in particular due to unresolved semantic issues, a lack of clear application and development guidelines, and missing industrial-grade implementation platforms [Zoitl09b]. To this end, a large set of research works deals with modeling and realization aspects, since the standard leaves many questions about the application design and the implementation open [Vyat11]. Various approaches make use of model-driven engineering methods using particularly UML but also other modeling languages (see Chapter 4.2.3).

Many research activities deal with the application of IEC 61499 in comparison with traditional solutions like PLCs and IPCs or migration paths between both worlds [Vyat11]. Hussain and Frey presented the migration of a PLC controlled centralized laboratory application into an IEC 61499 compliant distributed control application [Huss05]. In [Gerb08] a similar transformation of a customer-related test bed is presented and rules are defined to convert IEC 61131 function blocks to IEC 61499 compliant function blocks. Dai and Vyatkin describe three different design patterns to migrate IEC 61131 systems to IEC 61499 [Dai12]. How the IEC 61131 and IEC 61499 standard can be harmonized to integrate both approaches in one automation system is investigated in [Zoitl09a].

However, industrial adoption could not yet gained since the number of actual implementations, even prototypes, is very limited [Häst11][Thra09]. Two case studies to investigate the implementation and application of IEC 61499 to industry-related setups were done for a shoe manufacturing plant managed by ITIA-CNR [Coll06] and a laboratory manufacturing plant of the TORERO consortium [Ferr05].

3.3.2 Multi-agent Systems

Basics of Multi-agent Systems

Multi-agent Systems (MAS) is a recognized field in computer science since its inception in the 1980s [Müll14]. Originating from the research field Distributed Artificial Intelligence (DIA), MAS are being characterized by decentralization and parallel execution of activities executed by agents [Leit04]. An agent is defined as a software artifact (i.e., computer system) that is capable of autonomous actions in its environment in order to meet its design objectives [Jenn98]. Therefore, an agent senses its environment as an input and produces output actions that affect it (see Figure 3-3) [Wool99].

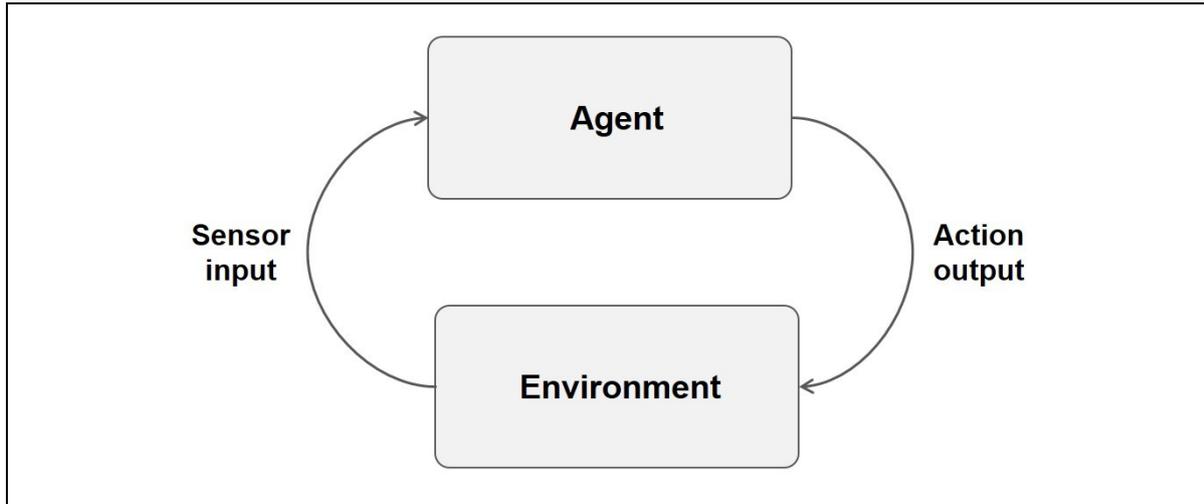


Figure 3-3: Interaction between agent and environment [Wool99]

A MAS is a network of cooperating agents, which are autonomously and intelligent acting entities [Lepu11]. The main characteristics of agents in a MAS are [Paol05][Wool95]:

- **Autonomy:** Agents perform most of their tasks without the direct intervention of humans and should have a degree of control over their own actions and their own internal state.
- **Interaction:** The capability of an agent to interact with its environment and other agents is the basis for achieving a joint global objective.
- **Responsiveness:** Agents perceive their environment and respond in a timely fashion to changes occurring there.
- **Proactiveness:** When responding to their environment, agents should exhibit opportunistic, goal-directed behavior and take the initiative when appropriate.
- **Adaptability:** An agent should be able to modify its behavior over time in response to changing environmental conditions.

Experience has shown that the realization of all of the characteristics is very demanding. For this reason, the minimum requirements for a software artifact to be an agent are autonomy and interaction, whereas the other three characteristics being responsible for intelligent behavior can be seen as optional [Beck13].

There exist several ways of how an agent makes the decisions to select the next action to perform. Four different categories of agents according to the way of decision-making can be distinguished [Wool99]: logic-based, reactive, belief–desire–intention (BDI), and layered agents. Logic-based agents act as the result of a symbolic reasoning through logical deduction. Reactive agents simply map perceptual input directly to actions depending on their current state. The decision making of BDI agents is based on an internal deliberating behavior. First, they identify the goal they see to achieve, being in a certain state and receiving a certain input, and then use their knowledge to reach it [Paol05]. In layered

agents, decision-making is realized via various software layers, each of which is more or less explicitly reasoning about the environment at different levels of abstraction.

Implementation methodologies and development environments are needed for the systematic development and implementation of MAS applications. The Foundation for Intelligent Physical Agents (FIPA) is an international organization with the goal to create agent standards promoting inter-operable agent applications and agent systems [FIPA02]. The work of the FIPA focuses mainly on the definition of a physical infrastructure named agent platform in which agents can be deployed [Paol05]. Since the FIPA standards describe an abstract architecture that cannot be directly implemented, additional implementation frameworks are needed. Various different MAS frameworks exist providing some predefined agent models and tools to develop MAS in compliance with the FIPA specifications [Fons02]. A well-known Java implementation platform for MAS is JADE (Java agent development framework) [Bell99].

Agent-based Manufacturing Control

Many different approaches arose that apply the MAS concepts to a wide range of production automation tasks, particularly production scheduling and planning [Lepu11]. Detailed state-of-the-art surveys regarding agent-based manufacturing are given in [Leit09], [Babi06], and [Shen06]. In the following, the most important research results regarding agent-based manufacturing control are summarized.

A manufacturing paradigm that is heavily influenced by MAS is Holonic Manufacturing Systems (HMS) (see Chapter 4.3.4). Despite large similarities between both concepts, they differ in the fact that holons can be composed of other holons. Thus, HMS control architectures often mix hierarchical and heterarchical control structures [Chri94]. Furthermore, holons can also comprise hardware parts of the system in contrast to agents. Although some approaches for agent-based manufacturing control are not explicitly called HMS but make use of these extended properties so that MAS with a mix of heterarchical and hierarchical structures emerge.

A well-known reference architecture for HMS called PROSA was developed at the university Leuven [Brus98]. It defines three types of holons: order holons, product holons, and resource holons. They are structured by using the object-oriented concepts of aggregation and specialization. Another HMS architecture is ADACOR (Adaptive Holonic Control Architecture for Distributed Manufacturing Systems) developed by Leitão [Leit04]. Within this architecture, manufacturing control is realized with supervisor holons that control operational holons representing the production equipment. This hierarchical control structure is supplemented by heterarchical interaction of the operational holons in order to dynamically react to disturbances and emergencies [Leit08].

Substantial research activities have also been performed at the Automation and Control Institute of the Vienna University. In [Vall09] an agent architecture is defined where each

automation agent is composed of a physical component and a software component. The software part is further decomposed in a low-level control (LLC) part controlling the mechatronic component and high-level control (HLC) part being responsible for the global behavior of the agent. Based on this architecture a framework for autonomous reconfiguration processes is developed in [Lepu11]. Thereby, the HLC determines the reconfiguration process and introduces new functionalities to the LLC. In prototypic implementations the IEC 61449 standard is used to implement the LLC agents [Hegn08].

Another active research group in the field of agent-based automation is the chair of Automation and Information Systems at the TU Munich (Prof. Vogel-Heuser). They propose a MAS architecture with two layers that differ in their functions and requirements, especially their real-time behavior [Ulew12]. The high level is located at MES level and the low level represents the field device level. For the realization of the field level agents a concept exists to implement agents directly on a PLC in the languages of the IEC 61131 [Schü11]. Moreover, a first approach for the model-based development of an agent-based automation system using SysML has been presented [Fran13]. Thereby, hardware-specific and software-specific planning information is separated in order to manage the complexity of the system design.

The EU project PABADIS (Plant Automation Based on Distributed Systems) and its follower PABADIS'PROMISE (PABADIS Based Product-oriented Manufacturing Systems for Reconfigurable Enterprises) dealt with the development of distributed control architectures using MAS in order to increase the flexibility of manufacturing systems [Feld09]. The basic principle for the design of control applications is based on predefined building blocks encapsulated as control agents [Pesc05]. A distinction is made between order agents that are responsible for the processing of a dedicated product and stationary agents representing the resources of the production system towards the agent world and enabling the access to the traditional control level.

Although a lot of research activities regarding MAS and HMS took place over more than 20 years, industrial applications are very rare and the implemented functionalities of existing ones are considerably restricted [Lepu11]. An analysis of this situation by Leitão particularly blames missing methodologies for the development of such distributed manufacturing systems and for the integration of physical devices with the control software [Leit09].

3.3.3 Service-oriented Architecture

Basics of Service-oriented Architecture

A concept for distributed computing that found its application particularly in the business process domain is *Service-oriented Architecture* (SOA). In this context, the main objective of SOA is to increase the capability of an enterprise to react to new requirements by reusing existing logic and providing a flexible IT infrastructure [Kraf05]. For this purpose, SOA

pursues synergies between the business and IT groups in an organization to offer greater flexibility [Bieb05].

There exist various definitions of SOA focusing different aspects; most of them refer to enterprise IT systems. A general definition according to Bieberstein defines SOA as a software architecture in which application functions are built as components (services) that are loosely coupled and well-defined to support interoperability and to improve flexibility and reuse [Bieb05]. A complementing definition with a more application-centric view by Erl describes SOA as a model in which the automation logic for executing a process is decomposed into reusable units of logic that are known as *services* (see Figure 3-4) [Erl05]. The operating principle within a SOA works according to the client-server-principle where the service is provided by a server (or service provider) and can then be used by a client (or service consumer). Therefore, each service needs a defined service interface and a service description to make the functionality of the service accessible.

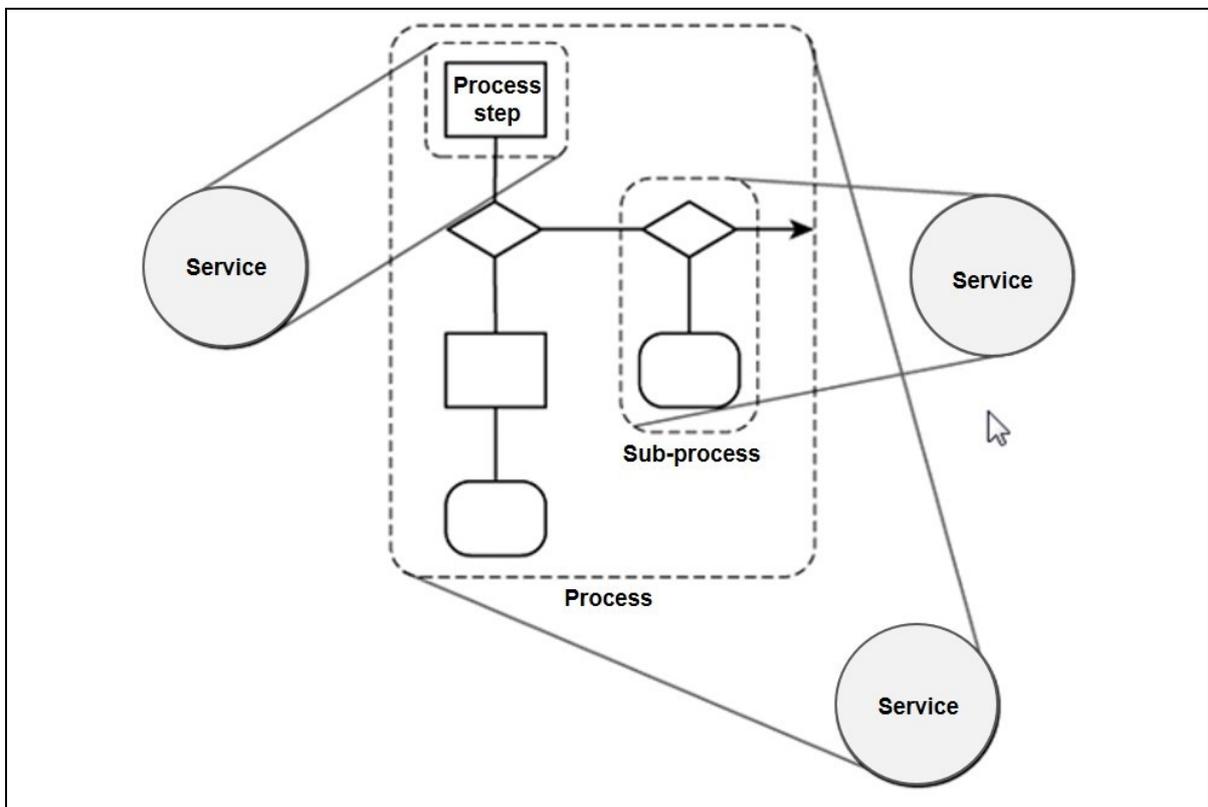


Figure 3-4: Encapsulation of a logic to services [Erl05]

Accordingly, the major items of SOA constitute services that are the building blocks for realizing a certain process. Services encapsulate versatile, different applications in a reusable and openly accessible way enabling a platform and implementation independent usage [Melz08]. The size and scope of the application represented by a service can vary. When building an automation solution consisting of services, each service can encapsulate

a task performed by an individual step, a sub-process comprised of a set of steps, or even the entire process logic (see Figure 3-4) [Erl05].

The main properties of services are [Bieb05][Erl05]:

- **Abstraction:** The implementation of the service is encapsulated, hidden from the outside, and wrapped by a service interface. This enables to focus on the most significant characteristics in a system and to manage complexity.
- **Autonomy:** Services have control over the logic they encapsulate.
- **Composability:** Composite services can be built by aggregation of existing services to form different levels of granularity and promote reusability.
- **Loose coupling:** Services are designed to interact without the need for tight cross-service dependencies.
- **Reusability:** SOA encourages services to be reusable so that new high-level functionality can be generated by using existing services.
- **Service contract:** The service interface and the terms of information exchange are described in an open-accessible service contract.
- **Statelessness:** Services should minimize the amount of state information they manage and the duration for which they hold it.
- **Integration capability:** Open-standard formats for communication protocols and service interfaces should be used to enable an interoperable service network.

The expected advantages of SOA are manifold. From a technical view, the SOA paradigm permits a high interoperability and reusability of software components based on the encapsulation principle. Focusing the software engineering, SOA decouples the design of the software architecture from the system architecture. By this means, the services can be independently designed from the system architecture so that they can be deployed very flexible on a the available resources [Kraf05]. So far, SOA has been widely adopted by ERP solutions to enable a horizontal integration of the enterprise with its business partners and the flexible adaptation to new market conditions [Math09b].

SOA for Manufacturing Control

Despite being the youngest of the three concepts regarding the application in automation, SOA gained a great popularity in research activities concerning industrial automation during the last years. Several research projects dealt with the development of comprehensive service-oriented automation systems, whereas very few research activities focus the application of SOA for manufacturing control.

There are a couple of related EU projects that dealt with the application of SOA in automation. During the SIRENA (Service Infrastructure for Real time Embedded Networked Applications) project pioneering activities took place concerning the application of the SOA paradigm on device level [Jamm05a]. A service-oriented framework was created for specifying and developing distributed applications in diverse real-time embedded

computing environments including industrial automation [Bohn06][Jamm05b]. Based on the results of SIRENA the SODA (Service-Oriented Device Architecture) project developed an ecosystem for the implementation of SOA on embedded devices [Souz08]. Using emerging standards from both the embedded-device and IT domains the SODA distributed software integrates physical devices into distributed IT enterprise systems [deDe06].

In parallel to SODA, the SOCRADES (Service-Oriented Cross-layer Infrastructure for Distributed Smart Embedded devices) project issued new methodologies, technologies, and tools for the development of networked systems for automation applications [Souz08]. One core task of SOCRADES dealt with SOA as an enabler for the vertical integration of enterprise systems with smart embedded devices [Karn07][Spie09]. The objective was the direct connection of enterprise systems and manufacturing devices on shop floor so that enterprise applications can be empowered to use shop floor data directly to dynamically report trends, causes, and analysis [Nguy08]. Therefore, concepts were developed to integrate existing legacy devices without SOA interfaces into the SOCRADES network [Bang09][Died08]. Furthermore, SOCRADES investigated the model-based development of control applications using High-level Petri Nets (see Chapter 4.2.2).

The follow-up project IMC-AESOP (Architecture for Service-Oriented Process-Monitoring and -Control) envisions SOA-based monitoring and control applications for batch and continuous production processes [Jamm14]. The objective is to create a new SCADA/DCS ecosystem where components can be dynamically added or removed and dynamic discovery for enabling the on-demand information combination and collaboration is provided [Dels12][Karn10]. A substantial role in all of these projects plays the SOA technology Devices Profile for Web Services (DPWS) (see Chapter 3.4.5). It is also used in other projects dealing with SOA for physical devices like LOMS (Local Mobile Services) and OSAMI (Open Source Ambient Intelligence) [Dohn10][Zeeb08].

Besides these collaborative projects, other research activities also work on concepts for realizing SOA-based factory automation. The chair of Process Control Engineering at the RWTH Aachen (Prof. Epple) works on concepts for applying SOA in process control systems. They investigate how the implementation of services for existing automation devices can be realized by the specification of communication and resource description models [Mers10][Merb11b][Merc12]. The approach Service-oriented Process Control (SOPC) was introduced that describes an implementation concept of services providing process control functions called Process Control Units (PCUs) [Yu10][Yu11]. The approach distinguishes among different types of PCUs: Single Control Units (SCUs) for individual actors, Group Control Units (GCUs) for actor groups or plant sections, and Action Control Units (ACUs) which represent the process-specific control sequences [Yu10]. The SCUs encapsulate the capabilities of the field devices so that their services can be used to build high-level functions in GCUs and ACUs by means of orchestration [Yu10].

Furthermore, Tan and Yi suggest an architecture to expose device services to enterprise systems based on the technologies DPWS and OPC UA [Tan10]. Feldhorst et. al. introduced the approach SOA for Devices for building control and monitoring applications that are built upon device services [Feld09]. Their work focuses on the integration of legacy devices that do not have the sufficient computational resources for exposing services by means of an integration layer. Another approach by Groba et al. proposes an SOA-based integration layer between control and MES layer to establish flexible maintenance applications [Grob08]. The doctoral thesis of Mathes describes a framework for Time-Constrained Services (TiCS) that focused the execution of Web Services (see Chapter 3.4.5) in industrial automation [Math09a]. Based on this framework a real-time SOAP engine with a low memory footprint for PLCs was developed [Math09b].

3.3.4 Comparison of the Concepts

Generally, all three presented concepts for distributed systems have in common that their purpose is to provide flexibility by decomposing an overall problem in various encapsulated components that interact with each other. Nevertheless, they have different characteristics and certain dedicated fields of application due to their respective origin. The IEC 61499 standard is clearly assigned to the automation domain because it is based on the established IEC 61131 standard extending it with an event-based function block specification [Mend11]. Its main purpose is the definition how distributed FBs on several executing devices can be synchronized in a heterarchical way. On the contrary, MAS and SOA are concepts from computer science for distributed computing. Agents originate from Artificial Intelligence and focus strongly on autonomy, self-organization, and proactivity describing the behavior of an agent [Ribe08]. Due to these characteristics, the adaption for automation applications is difficult due to the missing predictability of the system behavior. Driven by the domain of business IT systems SOA concentrates on generating high-level processes by using loosely coupled and reusable services. Whereas the IEC 61499 is a standard describing a behavior model tailored to automation tasks, the other two concepts are more general paradigms with a wide range of existing definitions. How agent systems and SOA have to be implemented is heavily dependent on the respective definition as well as the selected implementation concepts and technologies. Regarding this matter, the IEC 61499 has to be classified on a different concretization level.

Even if the concepts differ in objectives and characteristics in theory, their application in manufacturing control is often very similar. Concerning the definition of the basic components on the lowest granularity level—as FBs, agents, or services—all existing applications have in common that these represent the functionality of the field devices. This is due to the fact that a device is the last frontier where high level process workflows are transformed into a structured collection of physical actions to be invoked in a particular sequence [Când09a]. Additionally, the concepts don't generally exclude each other so that

several approaches combine the concepts. An example is the use of IEC 61499 as an implementation concept of MAS or SOA [Lepu11]. An approach to combine the characteristics of service-oriented and multi-agent methods is called Service-oriented Multi-Agent Systems (SoMAS) [Mend09b]. This shows that the interpretation of the concepts and their application can be very overlapping so that their commonalities reinforce regarding certain fields of applications.

All three concepts have not arrived in industrial practice of automation technology yet. For practical applicability mature technologies and proper reference models for the analysis, design, debugging, validation, deployment, and verification of the system are needed [Leit09][Thra06]. Besides this, a migration path is needed to enable a gradual introduction of new technologies starting from traditional PLC-based automation concepts [Preu11].

An often neglected aspect with huge impact on applicability is the availability of engineering methods to bridge the gap typically separating theory from practice [Paol05]. Today, well-integrated design methodologies facilitating component-based design throughout the entire design cycle of automation systems are still missing [Pang10]. The application of these emerging concepts and their technologies need to be accompanied by engineering methodologies to enable more flexible and reconfigurable production systems [Preu11]. A structured approach for the development of distributed control systems is necessary for assisting the control engineer in programming and in understanding the control programs [Sünd06].

In contrast to the other two concepts, SOA provides a methodological foundation supporting the design of software applications in a service-oriented manner. This is due to the fact that its objective is to generate adaptive business IT systems by a process-oriented composition procedure [Heut07]. Consequently, there already exist design principles and patterns that could help to overcome this lack of design methodologies for distributed control procedures.

3.4 Design and Implementation of SOA Systems

The SOA paradigm does not address design specification aspects nor implementation aspects [Bohn09]. Hence, the prerequisite to develop service-oriented architectures are methods for the design of collaborating services and technologies for implementation [Mend08b]. A firm set of design standards is critical to achieve a successful SOA providing reusability, composability, and agility [Erl05]. Since SOA has its origins in the business process domain, most of the existing approaches, literature, and best-practices focus on the design of business-driven IT systems. A selection of well-known methods for generally designing SOA applications, concepts for composition and specification of services, and technologies for realizing SOA are presented below.

3.4.1 Process-oriented Design Strategy

Today's enterprise IT systems require a flexible architecture that focusses on the business process in order to adapt processes quickly and with low efforts [Melz08]. SOA provides the foundation for adopting a process-oriented application design, which is particularly desired for business process management (BPM) [Kraf05]. This is reflected by the far-reaching goal of SOA to bridge the gap between analysis and implementation by linking the views regarding the desired behavior of the system (outer view) and the technological implementation of this behavior (inner view) [Erl05]. In the context of BPM this fact is expressed by the mismatch between business-related and technical concepts [Kraf05]. To improve this situation, a mapping of the business requirements documented as business logic and the IT organization comprising the applications for realizing the business process is required [Erl05]. By means of its abstraction and composition principles, SOA provides favorable capabilities to ease the mapping procedure. The services provide the backend functionality that is required to implement the desired process functionality [Kraf05].

The general process-oriented approach starts with the definition of the required process in form of a *process description* (see Figure 3-5) [Erl05]. Therefore, the business process is broken down into a series of granular process steps that are part of a workflow logic defining the sequence of the steps [Erl05]. A well-known modeling language to represent the process description is BPMN (Business Process Modeling Notation) that has been defined in order to support a standardized, graphical representation of business process diagrams [Kraf05].

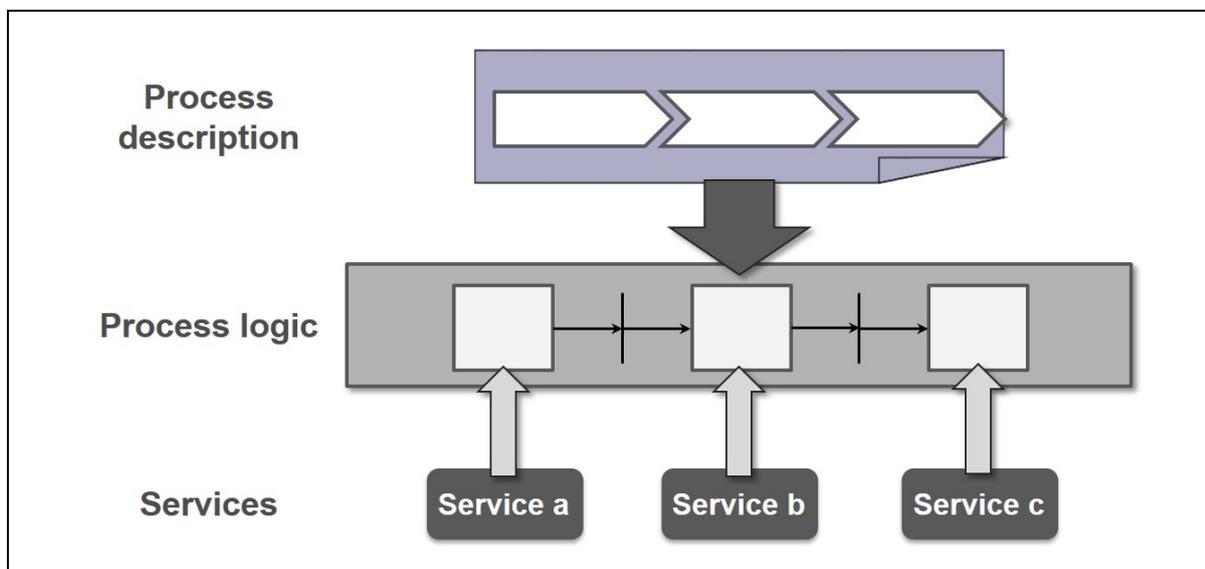


Figure 3-5: SOA mapping procedure

Since the individual process steps are supposed to be realized by services, the demanded functionality needs to be specified. To design a solution how the process can be realized, an executable *process logic* defining a concrete control flow by using control structures is

developed from the process description [Erl05]. To make the process steps executable, suitable services have to be bound to the process logic. This task can be described as a mapping between the specified requirements and available services forming the building blocks of the solution.

The most advanced mapping procedure with the highest degree of flexibility can be achieved by dynamic binding of services [Mend11]. In this case, services are discovered, selected, and bound dynamically during runtime. Therefore, sophisticated methods and tools are necessary that determine automatically the services needed from the abstract process description and the selection of the most suitable ones regarding current requirements. Eventually, a more common solution, which is sufficient for most purposes, is development-time binding where suitable services for enabling the desired process are selected before runtime [Kraf05].

3.4.2 Service Specification

Since a service works according to the Black Box principle, two aspects are required for its full specification: The external view represented by the service description and the internal details for implementing its behavior. The link between both views constitutes the service interface, which comprises a set of related *operations* to access the functionality of the service in a designated way (see Figure 3-6). Each operation represents a certain control logic to perform a unit of work and sends and receives messages to exchange input and output information [Erl05]. Thus, the control logic of the service implements the internal functionality of the service, which is hidden by the service interface (see Figure 3-6).

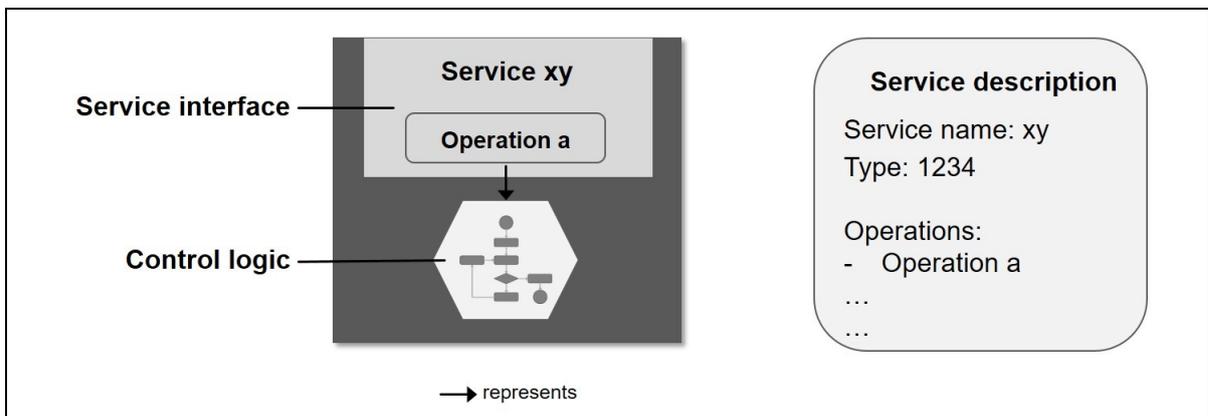


Figure 3-6: Structure of a service and its service description

The *service description* is needed so that potential service users are aware of the functionality of the service and how they interact correctly with it [Melz08]. For accessing a service the required operation and the exchanged information within the input and output messages have to be specified. Besides such general information about the functional properties of a service, a service description can comprise a wide range of further

information like details about the communication technology and non-functional requirements [Melz08].

Service descriptions are needed to find suitable services for current demands so that they constitute the enabler to execute a mapping from required functionality to services. The way how the service functionality is described mainly influences how convenient matching services can be found. For this reason, the name of the service and its operations should meaningfully express the functionality they provide. In [Erl05] guidelines for the naming of services are given. Dependent on the type of service a consistent naming scheme using a noun or verb or a combination of both is proposed (e.g., “Invoice”, “SalesReporting”, “VerifyID”) [Erl05]. The concrete expression should ideally be derived from an existing company standard or even better an industry standard.

3.4.3 Composition Principles

A composition principle specifies how services are aggregated to obtain new functionality that can be encapsulated to a high-level service again. Generally, there exist two different composition principles. The first one is denominated *orchestration* and constitutes the central coordination of process logic by sequencing and synchronization (see Figure 3-7 left) [Când09a]. Thus, each service is independent of the respective process logic where it is used within and all knowledge about the process flow is centralized in one master. To implement the desired order of execution, the control flow has to be implemented according to an executable orchestration language and it has to be executed with an orchestration engine. For business processes a process description in BPMN is then often combined with a service orchestration with the executable languages BPML (Business Process Model Language) or BPEL4WS (Business Process Execution Language for Web Services) [Bohn09].

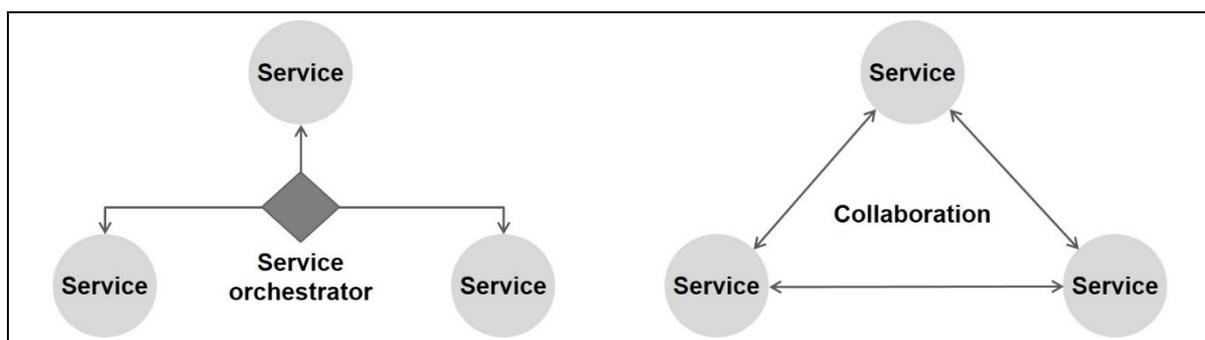


Figure 3-7: Service orchestration (left) and choreography (right)

In contrast, a complete decentralized workflow of cooperating services is called *choreography* (see Figure 3-7 right) [Bohn09]. Therefore, every service must know its own role within a process, i.e., what the service supports and how it reacts in a particular context [Când09a]. In fact, the complete process information is distributed over the services. The

absence of a central controller can help to make choreographies more robust and flexible than orchestrations because single services act more autonomously. Nevertheless, the major advantage concerning adaptability of the orchestration principle is that the orchestration logic is concentrated in one place and can therefore easily be changed [Kirk08]. Consequently, services can be designed independent of each other and the respective process leading to a loose coupling of services. On this account, the use of the orchestration principle is much more popular in practice than choreography, since the services are more flexible and reusable. Both principles do not mutually exclude each other so that software architectures mixing orchestration and choreography are also possible.

3.4.4 Reference Architecture for Services

By means of the mentioned composition methods, various levels of services can be created. The question arises how services are specified in an optimal way so that the mapping procedure between process description and available services runs as smoothly and straightforward as possible. Thereby, the determination of the optimal granularity of a service is a crucial task because a too high abstraction level is often contradictory to the reusability of a service [Bieb05]. However, fine-grained services increase the complexity of the superior process and the gap between process and implementation [Melz08]. The definition of a general service structure within a SOA is a precondition for the effective design of SOAs in order to break a complex application landscape down into manageable parts [Kraf05]. A basic concept for structuring the functionalities within a SOA constitutes the separation of concerns to enable reusability and clarity [Erl05]. This is enabled by means of the introduction of *services layers* and *service types* for defining a reference architecture (see Chapter 4.2.1) according to the respective application domain.

By means of composition of services to high-level services various specialized service layers can be built, whereby each layer can abstract a specific aspect of the overall solution [Erl05]. A four-layered SOA architecture is described by Krafzig (see Figure 3-8 left) [Kraf05]:

- **Enterprise layer:** contains application frontends to communicate with the user and public enterprise services to enable cross-enterprise integration.
- **Process layer:** comprises process-centric services
- **Intermediary layer:** consists of services that act as facades, technology gateways, adapters and services that add functionality to existing services.
- **Basic layer:** represents the foundation of the SOA providing core business logic and data.

A further subdivision can take place by the classification of services with the same characteristics as service types according to their kind of functionality and their properties (see Figure 3-8 right). In this instance, Krafzig defines several service types for each service layer, such as the basic layer contains *data-centric services* and *logic-centric services*

[Kraf05]. The specification of these basic services is particularly critical, since they are not composed of other services and therefore constitute the fundament of a SOA.

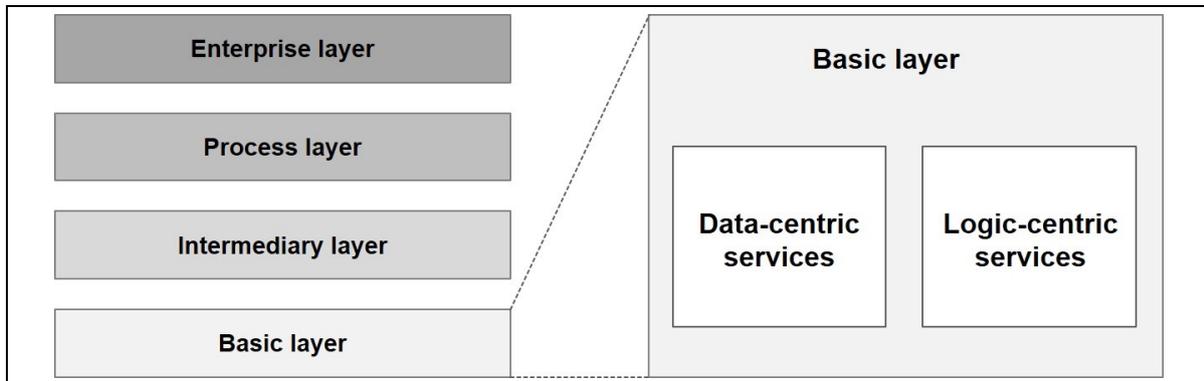


Figure 3-8: Classification of services according to layers (left) and types (right)

3.4.5 SOA Technologies

How these fundamental concepts and principles of SOA are realized in a respective software architecture depends on many implementation details. Several SOA technologies exist for making software components available as services in a network. Each of them provide a different set of features and uses distinct communication technologies. The most common SOA technology are *Web Services* that helped SOA to distribute and establish [Bobe08]. A lot of SOA definitions and application concepts consider Web Services synonymous with service-orientation so that this assumption often limits SOA as a purely technical concept [Luth12]. Still other technologies exist besides of Web Services for realizing SOA. In the domain of automation technology notably emerged DPWS and OPC UA [Kirk08][Souz08][Leit07].

Web Services

A Web Service can be defined as a software system designed to support interoperable machine-to-machine interaction over a network [W3C04]. This implies a family of technologies that consist of specifications, protocols, and industry-based standards that are used by heterogeneous applications to communicate, collaborate, and exchange information among themselves in a secure, reliable, and interoperable manner [Bieb05].

The World Wide Web Consortium W3C defines a basic set of technologies which consists of WSDL (Web Service Description Language), HTTP (Hypertext Transfer Protocol), and SOAP (originally Simple Object Access Protocol, not an acronym anymore) [W3C04]. Additionally, XML (Extensible Markup Language) serves as basic description language. A service description is represented by WSDL specifying the service interface as a machine-processable and human-readable XML file [Bobe08]. A WSDL file contains an abstract part for defining the service operations, message formats, and data types and a concrete part determining the technological details including transport protocols and network locations

[Melz08]. Other systems interact with the Web Service in a manner prescribed by this description using SOAP messages, typically conveyed using HTTP [W3C04]. SOAP is a XML-based messaging protocol that is used to encode the information in Web Service request and response messages before sending them over a network [Bieb05].

A basic W3C Web Service can be extended and specialized for different applications by various technologies that are summarized as WS-* specifications. Therefore, a Web Service profile defines the subset of protocols used for implementing a specific application, required adaptations of the protocols, and the way they should be used in order to achieve interoperability [Bohn09].

DPWS

The Devices Profile for Web Services defines a Web Service profile based on the W3C Web Service standard extending it with several WS-* specifications and profile-specific adaptations [Bohn09]. The objective of DPWS is to implement Web Services on small devices to enable Web Services on embedded systems [Bony11]. The research project SODA promoted DPWS as an OASIS (Organization for the Advancement of Structured Information Standards) standard and other projects like SOCRADES made use of it [OASI09].

The additional WS-* protocols are: WS-Discovery, WS-MetadataExchange, WS-Transfer, WS-Eventing, WS-Security, WS-Policy, and WS-Addressing (see Figure 3-9 left). The WS-Discovery protocol enables the dynamic publication and discovery of services during runtime [Bobe08]. After the discovery process WS-MetadataExchange and WS-Transfer are used to gather more information (i.e., meta data) about a DPWS device besides the pure endpoint address [Bohn09]. WS-Eventing offers mechanisms for event-based messaging so that events can be provided by services and subscribed by other services [Tan10]. A secure transfer of SOAP messages going beyond the capability of HTTP is enabled by WS-Security. WS-Policy is used to describe mandatory and alternative properties for interacting with a service in a generic way [Math09a]. By means of WS-Addressing a transport-neutral addressing scheme is provided so that the addressing of services and messages does not depend on the underlying transport protocol [Math09a].

DPWS defines the concept of a *DPWS device* and distinguishes between a *hosting service* of a device and a *hosted service* (see Figure 3-9 right) [Bobe08]. Each device can provide a number of hosted services—corresponding to normal Web Services—which are managed by the hosting service of the respective device. The hosting service offers metadata that describes the DPWS device and its properties like manufacturer, model name, model number, and serial number. The development of DPWS itself and tools for the application and implementation of DPWS were subject of various research projects. As a result, the initiatives SOA4D (SOA for Devices) and WS4D (Web Service for Devices) were initiated for providing open source DPWS development toolkits [Mens11][Zeeb10].

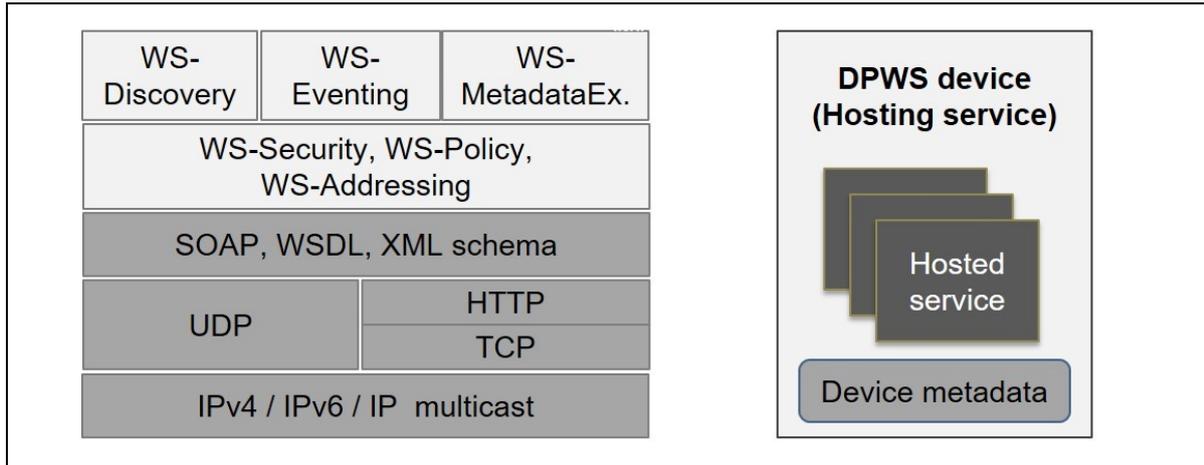


Figure 3-9: DPWS protocol stack (left), hosting and hosted service (right)

OPC Unified Architecture

OPC UA is mentioned here since it is currently gaining in popularity and acceptance in automation technology. Its basis is OPC, a wide-spread technology standard specified in 1996 for the uniform access from control and supervisory systems to field devices independent of the different field bus systems [Drat09]. Working on the server-client-principle OPC is mainly used for the horizontal and vertical integration of different automation systems like PLCs and high-level IT systems. A common application scenario is the visualization of shop floor data in HMIs. Since OPC has some drawbacks, such as the dependency on Microsoft Windows, the new standard OPC UA was developed and published 2006. OPC UA claims to offer a platform independent, secure, and performant data transmission between software applications [Leit07].

Apart from mere data exchange OPC UA can realize function calls with the concepts *Methods* and *Programs*. Programs model complex, stateful functionality that can be managed by calling methods. Methods represent basic functions that affect the behavior of a Program by causing specified state transitions [OPC07]. With these features OPC UA is able to realize encapsulated functions as services in terms of SOA.

The OPC UA stack defines how the OPC UA clients and servers have to be implemented. For coding and encoding of data UA XML and UA Binary are supported that are either used with the transmission protocol HTTP/SOAP for XML-based messages and TCP for binary messages [Kys11]. There exist different implementations of the OPC UA stack in C/C++, .NET, and Java that are developed by the OPC Foundation and other companies. The detailed OPC and OPC UA specification can be achieved from the OPC Foundation.

4 Concepts for Efficient Control Engineering

The development of control software constitutes today a highly complex task with growing importance and great room for improvement within the production engineering process (see Chapter 2.3.4). Besides engineering of traditional control systems, the establishment of mature engineering methods is still an open action item to leverage innovative distributed control architectures into industrial practice (see Chapter 3.3.4). Thus, innovative methods are needed to enable an optimal design and to lower the efforts of control engineering.

From computer science emerged numerous concepts, methods, and tools to improve the efficiency of software engineering. Some of them constitute a promising foundation to improve the control engineering in terms of the overall efficiency and clarity for the user. Since control engineering has special requirements and prerequisites compared to purely IT-related software solutions, software engineering methods need to be transferred and adjusted to fit the needs of automation technology. Special characteristics of control software are its strong dependency on the physical devices they control and stringent demands on the behavior of the system regarding real-time and reliability [Berg06]. Moreover, a basic difference between pure IT software and control software is the fact that IT software is already the desired result whereas control software is used as a means to execute a physical process [Berg06].

This chapter presents several existing approaches that can help to improve control engineering by integrating it with other planning domains, providing clarity for the design, and making engineering results continuously available and reusable. Besides some general concepts from software engineering the method of Model-driven Engineering is introduced. Furthermore, some comprehensive engineering concepts and helpful engineering standards are presented.

4.1 General Concepts from Software Engineering

Software engineering comprises methodologies for the analysis, design, and modeling of software according to existing requirements [Alva13]. Two major aspects of software engineering are programming paradigms to design software and procedure models that guide through the single phases of the software life cycle. Both have already successfully been applied for developing control software.

4.1.1 Programming Paradigms

There exist numerous programming paradigms representing a certain style with defined characteristics of the programs. Two well-known paradigms with growing impact on control engineering are modularization and object-orientation (OO).

A proven remedy to manage the rising complexity of software systems is modularization (see Figure 4-1). By means of a well-defined segmentation of the overall problem into single aspects, modularization improves the flexibility and comprehensibility of a system while allowing the shortening of its development time [Parn72]. The latter is enabled by reusable software modules that can be applied as partial solutions within various projects. Furthermore, encapsulating software to modules decreases the overall complexity for the user by hiding the implementation details of the module [Frie09]. Such an encapsulated software unit is then called *module* or *software component*, accessible via defined inputs and outputs, and its functionality is as a rule suggestible via parameters. A related concept is *Component-based Development* standing for a design approach that assembles software components from a variety of sources to software applications [Bieb05].

In control engineering, modular design principles are already widely accepted and established. The procedural IEC 61131 programming concepts FUNs and FBs provide tools to realize modular PLC programs to a certain degree (see Chapter 2.3.3) [Frie09]. Apart from applications on conventional control systems, advanced modularization concepts constitute the fundament for distributed control applications with collaboration of various modules running on different devices (see Chapter 3.3) [Fran11].

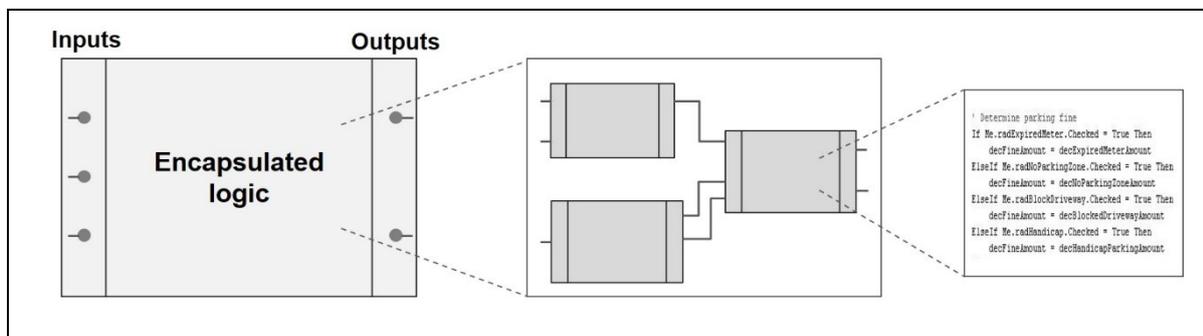


Figure 4-1: Encapsulation of software to modules

The second paradigm object-orientation is one of the most important programming paradigms and is based on a modular design [Berg06]. The single entities of an OO system are called *objects* which have their unique identity, own local state, and operations that can change this state [Wehr09]. Besides objects the core elements of OO are *classes*. A class represents a type of objects with the same characteristics and describes their structure and behavior [Weil08]. By instantiation a new object is generated with properties specified in the respective class. A basic concept of OO is *inheritance* indicating a hierarchical relation

between two classes whereby one subclass takes over the properties of a superior class [Berg06].

During the last decade many activities have been ongoing to establish object-oriented concepts for PLC programming [Wern09][Wits09]. Although the IEC 61131-3 already comprises some basic encapsulation concepts with Functions and Function Blocks, important characteristics of OO like inheritance have been missing for a long time [Thra11a]. The release of 3rd edition of the IEC 61131-3 introduces some object-oriented programming (OOP) features like interfaces and methods for FBs [IEC13][Vyat13]. For the time being, the PLC tool CoDeSys is the only known programming environment supporting these extensions [Ober15]. Until now, some PLC manufacturers also provide their PLC programming tools with object-oriented extensions, like IndraLogic from Bosch and TwinCat from Beckhoff, since most of them are based on the tool CoDeSys.

Apart from mentioned possibilities of modular and object-oriented programming of PLC programs, there exist numerous approaches that make use of OO as a general engineering paradigm. Since automation control is heavily dependent on the production equipment they control, these approaches are gathered under the term *mechatronic systems design* where physical devices and software are regarded as one unit (see Chapter 4.3.2).

4.1.2 Life Cycle Models

Life cycle (or procedure) models specify the single steps during a software development process from specification to testing and maintenance. They determine activities with the required input and output information and facilitate a structured process, which optimizes the engineering and data workflow and the resource utilization [VDI10]. The two most popular life cycle models are the Waterfall model and the V-model [Kleu10].

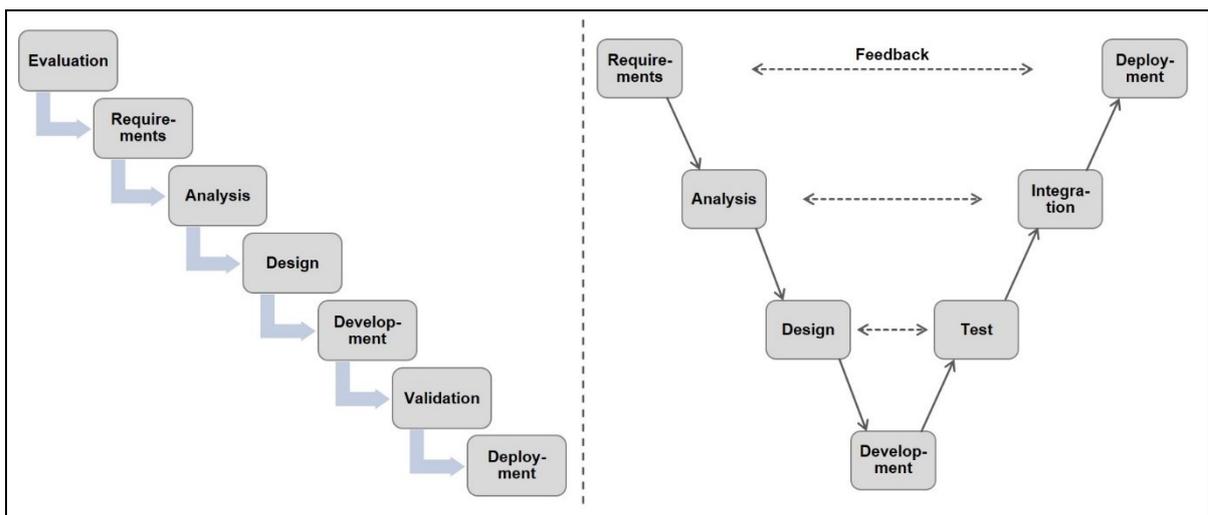


Figure 4-2: The Waterfall model (left), V-model (right)

The Waterfall model is a simple linear sequence of the phases evaluation, requirements specification, analysis, design, development, validation, and deployment (see Figure 4-2 left) [Royc70]. It is the oldest life cycle model and constitutes the foundation for all upcoming models by defining the general phases of software development [Rupa10]. Initially, all phases were arranged in a strict sequential manner from the first phase to the last phase. Later on, the model has been extended by arrows allowing to go back to previous phases to carry out improvements [Kleu10].

The V-model comprises similar phases as the Waterfall model but arranges them in the shape of a V (see Figure 4-2 right). The left leg of the V represents the evolution of requirements into ever smaller components through the process of decomposition and definition, whereas the right leg comprises the integration and verification of the system components into successive levels of implementation and assembly [Rupa10]. Thus, development phases are confronted with their related testing phases to enable an iterative development procedure. If problems occur during testing, all dependent design results need to be improved and tested again [Holz07]. The vertical axis depicts the level of decomposition so that the more complex a system is, the deeper the V shape gets with correspondingly larger number of stages [Rupa10].

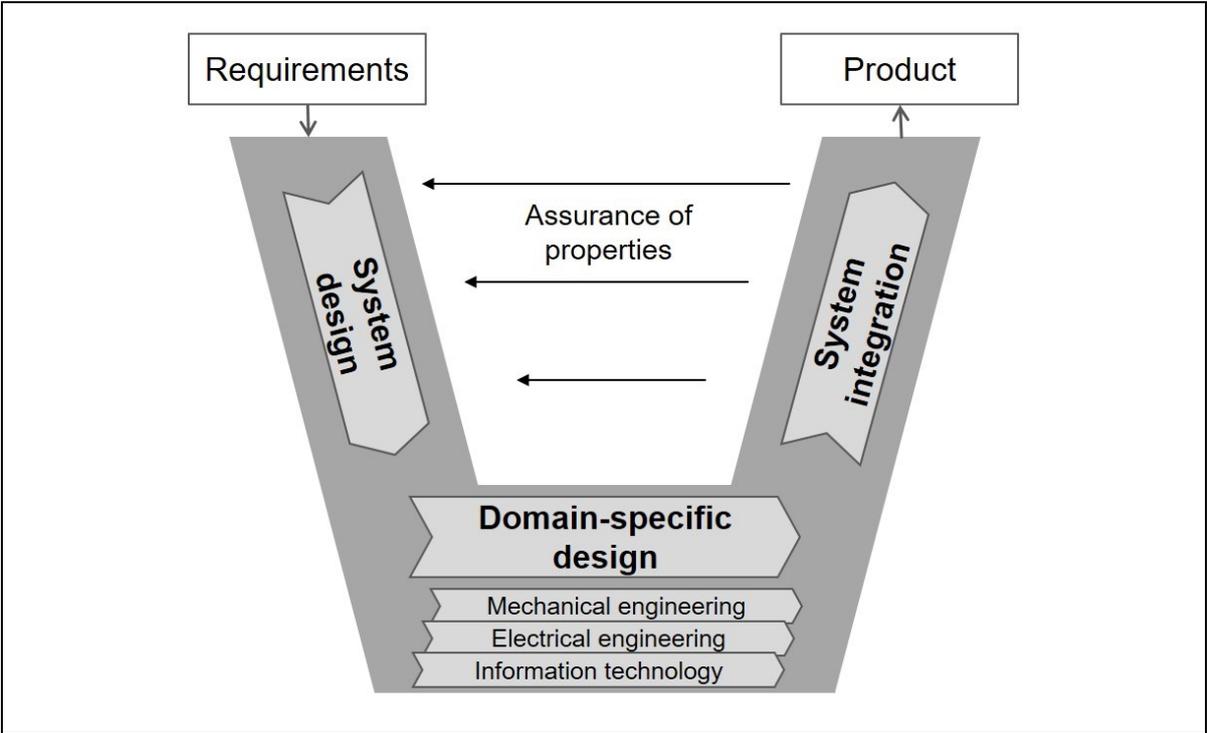


Figure 4-3: V-model for mechatronic system design [VDI04]

In opposition to normal IT software, the development of control software is a part of production engineering and dependent on other engineering disciplines, such as mechanical design (see Chapter 2.3.2). Thus, lifecycle models are used for the overall

engineering process and constitute usually of a temporary sequence of planning phases like the Waterfall model. Besides this, the V-model has also gained in popularity for application in production engineering processes due to its adaption for the development of mechatronic systems (see Chapter 4.3.2). With the guideline VDI 2206 a general design methodology is specified that can be applied, among others, to production systems [VDI04]. This V-model represents the logical sequence of the sub-steps: requirements, system design, domain-specific design, system integration, and product (see Figure 4-3). The methodology is inspired by Systems Engineering where the system as a whole is interdisciplinarily specified first (see Chapter 4.3.5). Afterwards, the domain-specific details are worked out in a preferably parallel way. One of the domains is information technology where control engineering tasks can be assigned to. During system integration, a continually assurance of properties takes place to check if the actual system properties coincide with the desired system properties. This cycle can be run multiple times for stepwise concretization of the product.

4.2 Model-driven Engineering of Control Procedures

From computer science emerged the concept *Model-driven Engineering* (MDE) for the specification, design, and implementation of software applications by using models. In literature, the term *Model-driven Development* (MDD) is mainly used in the same context as MDE so that for this thesis their equivalence is expected.

4.2.1 Basic MDE Concepts

MDE stands for software development methods promoting models as primary engineering artifacts that are gradually refined towards the running application based on design decisions made by engineers [Häst11]. Models enable a higher level of abstraction by hiding or masking details, bringing out the big picture, or by focusing on different aspects [OMG05]. Developers are empowered to concentrate on the required functionality and the overall architecture of the system instead of spelling out every detail of the implementation [Atki03]. Therefore, MDE technologies combine modeling languages to formalize the properties of the software and transformation engines to generate platform-dependent code [Schm06].

The usage of models continues the stepwise increasing abstraction degree for software development during the last decades (see Figure 4-4). Low-level programming languages like assembler languages and machine code are close to the respective computer hardware and require just few or even no compilation or interpretation for generating an executable program. Much more abstraction from the underlying controller architecture is given by procedural or object-oriented languages. A further step constitutes the use of graphical modeling languages where programs are designed graphically.

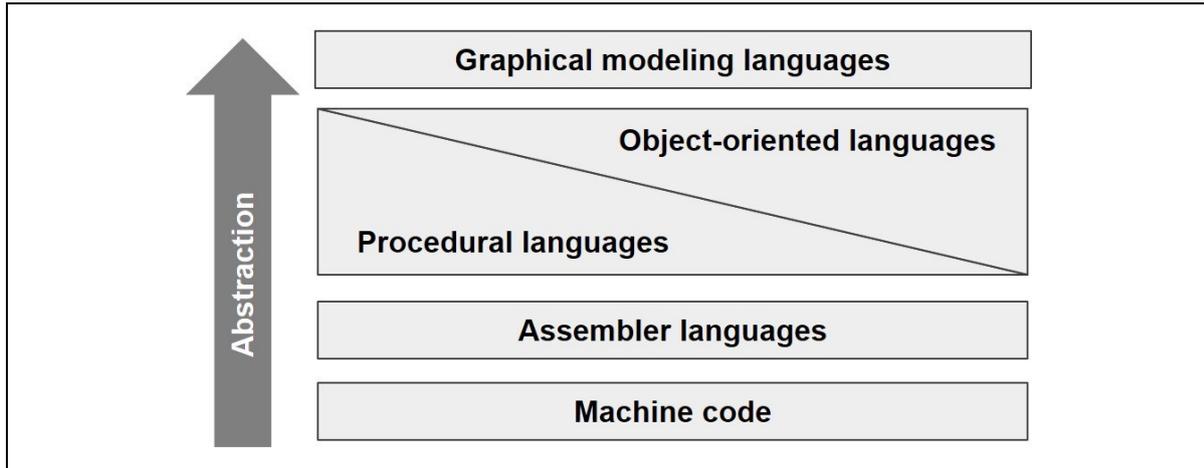


Figure 4-4: Increasing abstraction of programming languages [Weil08]

Since models help to understand a complex problem and its potential solutions by concentrating on certain facts, the major advantage of MDE is that the models are less bound to the underlying implementation technology and closer to the problem domain [Seli03]. This allows designing an application once and targeting it towards distinct software and hardware platforms that are still unknown during the initial development [Hovs06]. Applied to the control system development modeling promotes a better handling of the rising size and complexity of control software and a consideration of growing safety and quality requirements [Preu11].

The key foundations for the support of MDE are visual modeling languages, meta-level description techniques, and OO [Atki03]. Modeling languages, also referred to as description languages, serve for the representation of modeling aspects and consist of a syntax, grammar, and semantics [VDI10]. In particular, visual modeling languages with graphical notations support effectively human visual perception and thus, can be simply and intuitively applied by the engineer. The rules for creating a model are specified in *metamodels*, which presents the conceptual entities, their attributes, and the relations that comprise the vocabulary of a type of model [Clem10]. A distinction is made between general-purpose and domain-specific modeling languages. In contrast to general-purpose modeling languages like UML and SysML, domain-specific modeling languages are subject to a particular problem domain or field of application [Henn10]. Such domain-specific concepts can be defined by using OO, since it enables flexible language extensions by letting developers extend the set of available types that can be used for modeling [Atki03]. These concepts are captured in specific metamodels, which define the relationships among concepts in a domain and precisely specify the key semantics and constraints associated with these domain concepts [Schm06].

A formal approach for creating a generic modeling infrastructure that is able to describe different kinds of metamodels is the specification of a *meta-metamodel*. It comprises the

concepts, how they relate to one another, and which rules govern their existence and behavior for the definition of a metamodel [Hovs06].

A well-known meta-modeling architecture is the *Meta-Object Facility* (MOF), which is an OMG standard [OMG06]. It consists of four hierarchical levels where each represents the instance of the level above [Atki03]: The bottom level M0 holds the *user data* which comprises the actual data objects (instances) of a model as representatives of the real-world objects. On the next level M1 *user concepts* classes of the user data are defined so that customized models for a certain domain can be created. The M2 level defines metamodels that specify rules for the generation of models. The most prominent representative for this level is the specification of UML (see Chapter 4.2.2). Finally, level M3 holds *meta-metamodels* that represent the rules to define metamodels and furthermore, iteratively define their own structure.

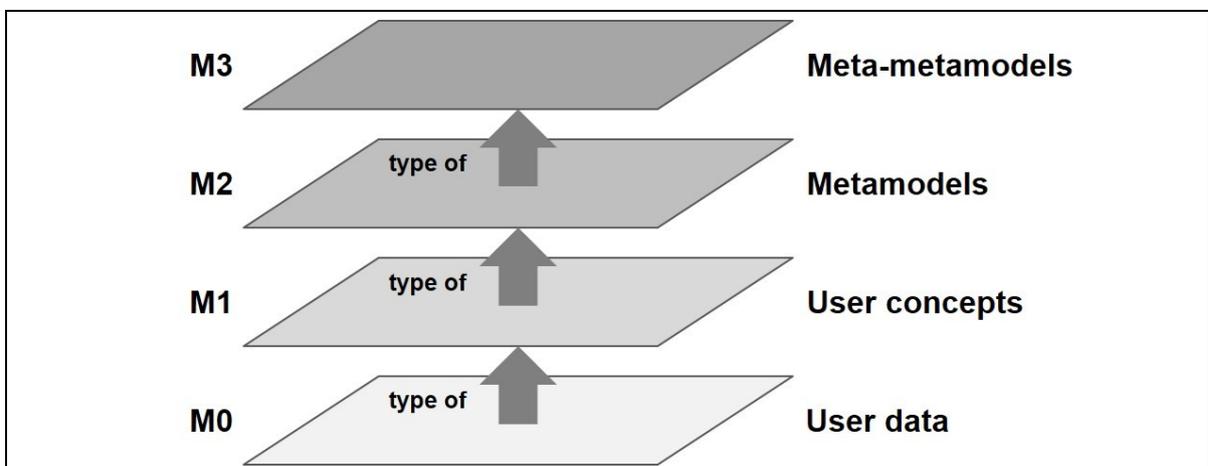


Figure 4-5: Meta-modeling architecture [Atki03]

Two further concepts supporting the applicability of MDE are *reference models* and *reference architectures*. Reference models provide an abstract framework using guidelines or specifications to enable the development of models within a certain environment [OASI06]. Whereas reference models rather describe the generic rules of the modeling process, reference architectures act like design patterns [OASI06]. The structures of the respective elements and their relations of a reference architecture provide templates for concrete architectures in a particular domain [Clem10].

4.2.2 Modeling Languages for Control Engineering

There exist various different modeling languages that can be used to support control engineering. They differ in their scopes and formalization degree so that they serve for different tasks starting with specifying the automated production process, over designing and testing automation systems and software, right up to generate executable control procedures.

UML and SysML

UML is a general-purpose modeling language based upon fundamental OO concepts standardized by the OMG [OMG11]. It uses a graphical notation and can be used for analysis, design, and implementation of software-based systems as well as for modeling business and similar processes [Bieb05]. The concepts of UML are based on MOF which serves as the UML metamodel (see Chapter 4.2.1). The OMG also defines the XML Metadata Interchange (XMI) standard, which is a data format for UML models using XML [OMG15].

Since UML is independent of the programming language, it is widely used in early development phases for requirements specification and structural design of the software system [Bell03]. To derive executable software, the platform-independent UML models have to be mapped and specified in detail to obtain platform-specific applications [Secc07]. An UML-related MDE standard for translating UML to code is the Model-driven Architecture (MDA) [OMG03]. The MDA process starts with the Platform-independent Model (PIM) for modeling the application, followed by the Platform-definition Model (PDM) for modeling the target system which is needed for designated technologies [Esté12]. Furthermore, a Platform-specific Model (PSM) assigns the PIM elements to the devices and platform-specific configurations for automatically generating the target-specific code [Zoitl09b].

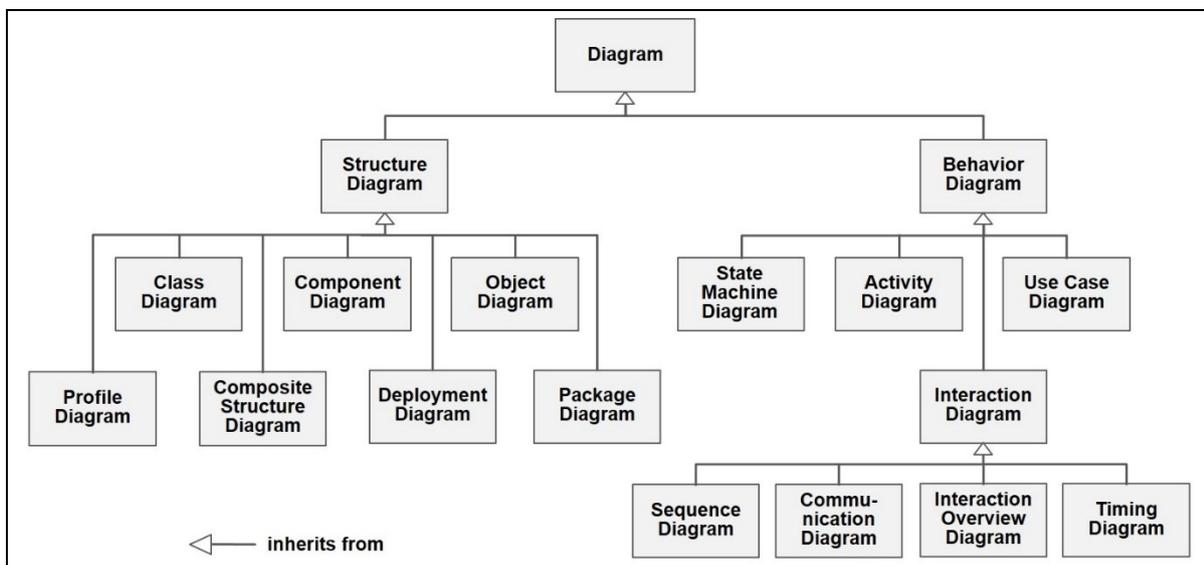


Figure 4-6: UML Class Diagram displaying all types of UML diagrams

UML supports a number of diagrams for representing different aspects of the modeled system (see Figure 4-6). They can be divided in two categories: diagrams that model structural information including the static properties of a system and diagrams that model behavior to depict functional capabilities of a system [Will07]. The backbone of each UML model is the Class Diagram depicting the static software structure with classes and their properties and relations to each other [Katz09]. An example for a behavior diagram is the

Activity Diagram which shows a process as a sequence of steps performing actions [Clem10]. Arrows and other control structure elements between the actions indicate the flow of control whose semantics operation principle is heavily inspired by Petri nets. By offering the profile mechanism customized extensions derived from the basic UML metamodel elements can be developed by adding new kinds of language elements or restricting the language [OMG03]. New elements are defined within a profile by stereotypes [Weil08]. As an instance the profiles SoaML and UML4SOA extend the UML by providing the possibility for behavioral specifications of services and focusing on transforming orchestrations down to code service orchestrations [Maye10][OMG09].

A standardized UML profile for Systems Engineering (see Chapter 4.3.5) applications is SysML (Systems Modeling Language). The intention of SysML is to unify the diverse modeling languages currently used on large systems projects with focus on the specification of requirements, structure, and behavior on systems properties [OMG12]. SysML defines language extensions for UML targeted to transform the base UML to a full-fledged, systems-centric language [Will07]. The biggest difference between SysML and UML constitute the concepts that represent single elements. In UML software entity classes and entities are modeled as *class*, *object*, or *component*. In contrast to this all structural elements—whether physical or logical, abstract or instance—are modeled in SysML as block [Weil08].

Modeling of Manufacturing Processes

Knowledge about the technical process is essential to support the planning, engineering, and commissioning of production systems in an appropriate way [Fell09]. In this regard, graphical process models specifically depicting manufacturing processes are a helpful tool to specify the system's behavior according to the planned objectives [Pint09].

The guideline “VDI 3682 Formalized Process Descriptions” specifies a notation with the aim to describe all information about a technical process necessary for engineering and normal operation throughout the life cycle of the system in a clear and structured layout [VDI05]. It contains essential symbols for different objects: product (P), energy (E), process operator (O), technical resource (T), and flow for connecting these objects (arrow) (see Figure 4-7). Generally, products and energies act as input and output values of a process step represented by a process operator. Technical resources facilitate the conversion of products and energies into new products and energies by a process operator. The process can be detailed in a top-down manner by decomposition of the process operators up to a final functional level where each object can be defined in a more detailed way.

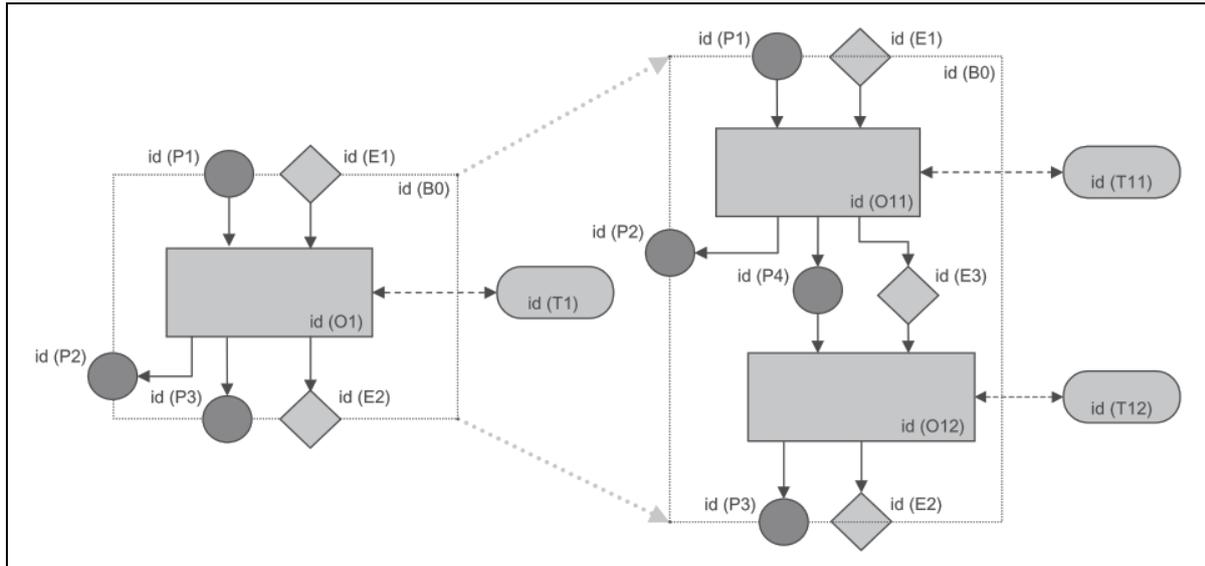


Figure 4-7: VDI 3862 Process model with two decomposition levels [VDI05]

Modeling of Sequential Control Procedures

The execution of manufacturing processes is usually characterized by sequential behavior where devices are turned on and off according to the current state [John99]. For automation control this behavior needs to be implemented in sequential control procedures. Instead of using text-based programming languages, graphical programming languages that are directly executable help to handle complexity and increase comprehensibility of the programs.

The most common modeling language for sequential control is GRAFCET (Graphe Fonctionnel de Commande Etapes/Transitions) which is specified in the standard IEC 60848 [IEC02]. GRAFCET is widely used since it constitutes the basis for the PLC programming language Sequential Function Charts (see Chapter 2.3.3), which is part of the IEC 61131 standard. A GRAFCET diagram consists of steps with associated actions, transitions with associated conditions, and oriented lines (see Figure 4-8) [Alva13]. The steps represent the states of the system and the transitions indicate which state changes are possible based on the current state of the system. An active step contains a token which is passed to the next step if the respective transition is active. The basic behavior of GRAFCET is inspired by Petri Nets, a formal modeling and analysis language for discrete-event and asynchronous systems with a graphical representation [Nof09]. There exist numerous variants and extensions of Petri Nets which increase its basic functionality by certain features. One of them are SIPNs (Signal Interpreted Petri Nets) that are based on Condition Event Petri Nets and enable processing of inputs and outputs [Frey02]. Therefore, transitions are associated with input signals and states specify the output signals [Huss05]. This feature makes it possible to use SIPNs as control procedures that can be formally verified and validated in contrast to common PLC programs [Frey06].

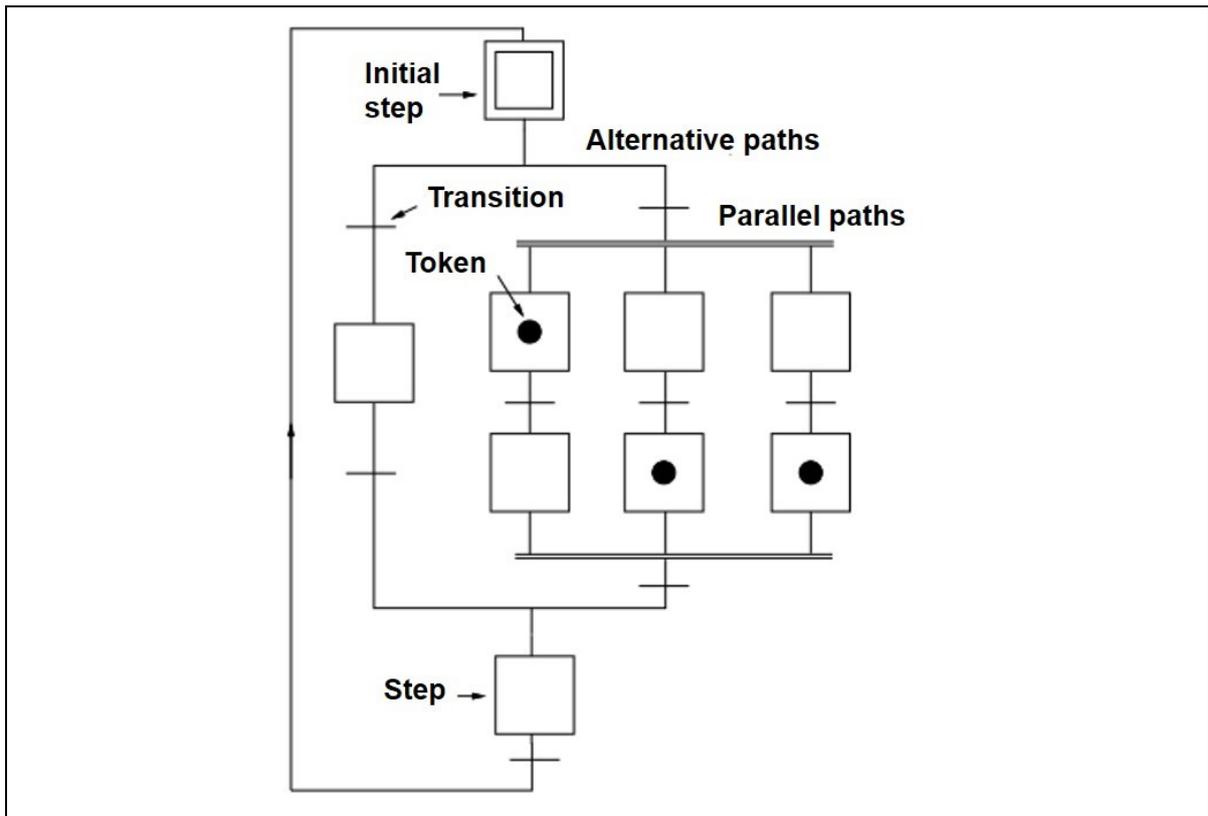


Figure 4-8: GRAFCET

Another graphical language with support for formal analysis is Grafchart aiming at supervisory control applications and batch control at process cell level [Arzé02][John98]. Grafchart is based on GRAFCET/SFCs and incorporates features from High-level Petri Nets and object-orientation [John08]. Available modeling features are steps, transitions, macro steps, alternatives, and parallel paths according to Grafcet [John99]. Furthermore, high-level programming features are defined like procedural steps, process steps, and the use of multiple tokens with attributes and methods (see Figure 4-9) [John99]. A tool for designing and executing Grafchart is the Java-based JGrafchart developed at Lund Institute of Technology [Olss05].

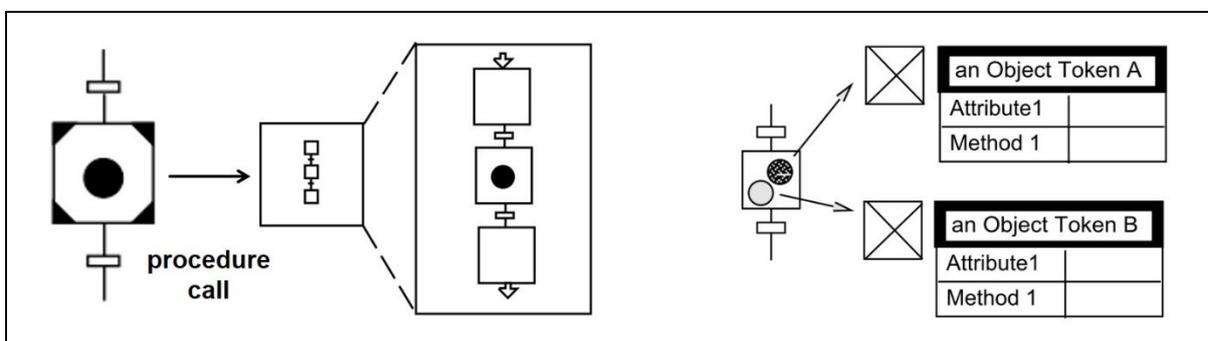


Figure 4-9: High-level concepts of Grafchart: Procedure call (left) and tokens with attributes and methods (right) [John99]

4.2.3 Existing Approaches for MDE of Control Procedures

Various approaches emerged for developing manufacturing control applications in a model-driven way. The subsequent sections give an overview of existing approaches of model-driven development of control procedures in IEC 61131 and for distributed control systems.

MDE of IEC 61131 Control Procedures

For designing PLC programs modeling can help to abstract from the low-level and platform-dependent PLC code to provide a better usability, portability, and clarity. By means of an automatic generation of executable PLC programs based on models engineering efforts can be reduced. Additionally, validation and verification tasks can be performed for the control procedures when formal modeling languages are used. Consequently, the application of model-driven engineering principles for developing automation control systems has become very popular so that various approaches emerged. For the development of control procedures, the IEC 61131-3 already comprises two graphical programming languages for designing the behavior of the control software, FBS and SFC. They constitute a good foundation but their abstraction level is rather low and they do not allow to design all aspects in a graphical way [Thra11b].

Especially UML and SysML are gaining in popularity for MDE for control systems design. At the University of Kassel and TU Munich substantial projects took place for generating executable PLC projects from UML/SysML models [Voge11][Wits10]. In this context the UML/SysML profiles UML-PA and the newer SysML-AT have been developed for the design of automation systems [Katz09][Voge14b]. A code generator was developed that transfers PLC projects as UML models from the tool Artisan Studio to the PLC programming environment TwinCAT from Beckhoff Automation [Voge05]. In a similar project a UML editor was developed for the development of UML/SysML-based PLC projects in the vendor-independent PLC engineering tool CoDeSys [Voge9a][Voge14b].

Another domain-specific UML profile, UML AP (UML Automation Profile), for modeling of automation and control applications based on UML was developed at the Tampere University of Technology [Häst11]. The profile extends UML and SysML concepts and covers requirements, automation concepts, distribution and concurrency, automation resources, and device interfaces [Rita07].

Thramboulidis and Frey examined how the IEC 61131 FB concept can be used in combination with UML and SysML for a MDE process [Thra11b]. They investigated which UML/SysML diagrams are suitable to represent different aspects of the control application, e.g., the Class Diagram for the PLC infrastructure and Activity Diagrams for the behavior between FBs. Furthermore, UML and SysML profiles were defined that contain certain stereotypes for the main key constructs of IEC 61131. Based on these results, a concretized model-driven development procedure for process control applications using Piping and

Instrumentation Diagrams as source of requirements was introduced [Thra11a]. For this purpose, the process control engineering requirements are represented in the CAEX (Computer Aided Engineering Exchange) format (IEC 62424) and transformed to SysML requirements diagrams. The requirements constitute an input for the SysML-based design process using the SysML4IEC61131 profile whose results are automatically transformed to the PLC program represented in the general PLC programming language PLCOpenXML [Drat09]. In subsequent work, extensions for the IEC 61131 standard are proposed as a meta-model for a better support of MDE and the deployment of 61131 FB diagrams in distributed execution environments [Thra12].

Besides UML/SysML other modeling languages are used as well for generating PLC code. As part of the research activities on SIPNs the tool SIPN Editor was developed, which supports the graphical implementation of SIPNs and the translation of SIPN to the model checker SMV as well as the PLC language IL [Klein03]. Apart from Petri Nets another formal model from computer science is Finite Automata that is used for PLC code generation as timed-message state graphs [Thap09] or PLC Statecharts [Wits10]. Moreover, the concept MeiA (Methodology for Industrial Automation) combines GRAFCET and UML Use Case Diagrams with GEMMA (Guide d'Étude des Modes de Marches et d'Arrêts), a model depicting all states of an automated system, for assisting the designer during the analysis, design, and coding phases [Alva12]. A XML-based method by Marcos and Estevez combines three different views (control engineering, electric engineering, and software engineering views) within one XML model for designing industrial control systems [Marc08]. Apart from these MDE approaches emerging from academia, the commercial modeling and simulation tool MATLAB provides PLC code generation with its Simulink PLC Coder [Math12].

Model-driven Development of Distributed Control Systems

Besides the engineering of classical control systems, a lot of research activities deal with the development of distributed control applications in a model-driven way (see Chapter 3.3). Since distributed control systems consist of various separated software modules, the higher abstraction level given by this modularization supports to close the gap between model and implementation and thereby, enables a high design performance [Vyat11]. Thus, the combination of model-driven engineering and distributed control concepts is a very promising approach for future control systems engineering.

UML is a popular modeling language for the design of distributed control systems. There exist numerous approaches that combine IEC 61499 system's design with UML. Dubinin and Vyatkin defined the UML profile UML-FB (UML for Function Blocks) to model the system's hierarchy as a class diagram [Dubi05]. The workgroup of Prof. Georg Frey delivered substantial results on model-driven engineering, automatic deployment, and validation of IEC 61499 control software resulting in the dissertations of Panjaitan [Panj07]

and Hussain [Huss09]. Their work makes use of the transformation of UML diagrams to an IEC 61499 representation: The external part including the interfaces and interconnections of a FB is described by Component Diagrams and Class Diagrams, the behavior of a FB's ECC is depicted as a State Diagram, Activity Diagrams are used to describe the algorithms of a FB, and the execution sequence of Devices is specified by using Sequence Diagrams [Panj06][Panj05]. Based on these modeling rules a development process following the V-model (see Chapter 4.1.2) is proposed including the automatic test case generation for formal verification (see Figure 4-10) [Huss06].

From the University of Patras emerged relevant contributions to MDE of IEC 61499 with engineering support systems, architectures, and development strategies in the context of CORFU (Common Object-oriented Real-time Framework for Unified Development) [Soft06]. The CORFU architecture 4LCA comprises four levels [Thra04]: The industrial process layer represents the real-world plant components. The layer above, i.e., the system layer, includes the software artifacts of the system on which the CORFU framework is implemented. The application layer comprises the software constructs of the automation applications and the HMI layer for developing HMI subsystems. For designing IEC 61499 control systems the CORFU development process adopts best practices from component-based development and therefore, utilizes specific UML diagrams that integrate with the FB concept [Thra07a]. The prototype system CORFU Engineering Support System provides tool support to demonstrate the applicability of the proposed process [Tran06].

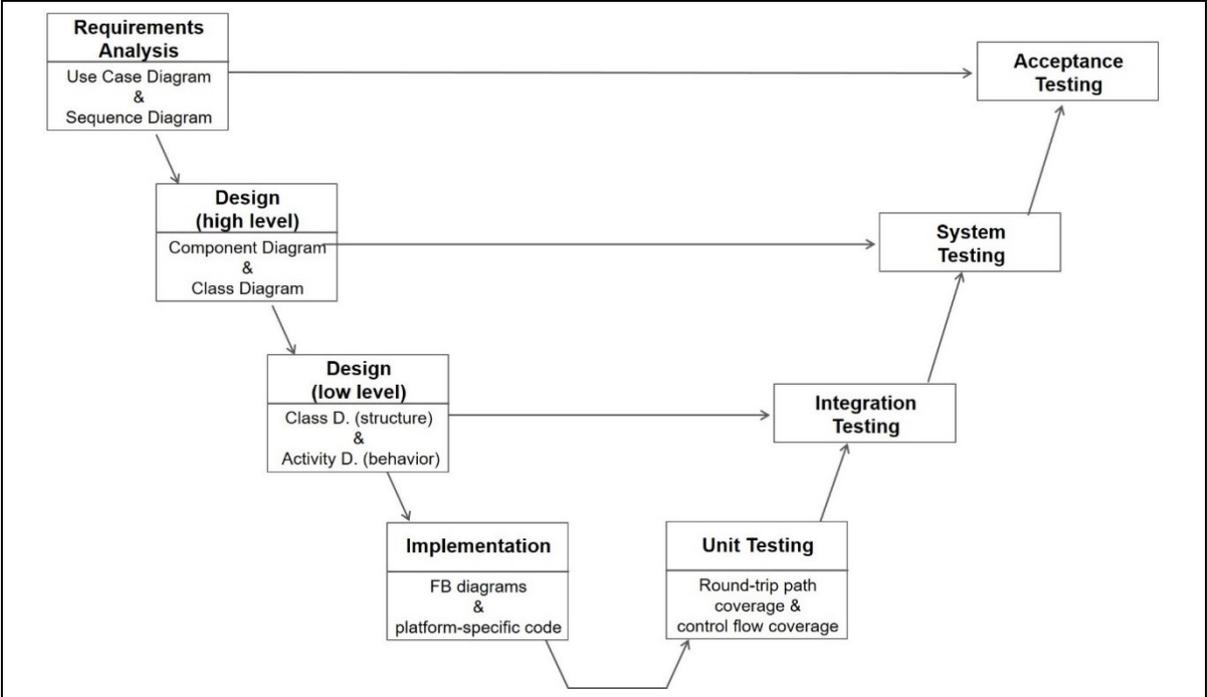


Figure 4-10: Process model for UML-based development of IEC 61499 software [Huss06]

Another common approach is to use existing process modeling languages for designing the control application and deriving the required FBs from the process model. The previously mentioned concept of Panjaitan was extended in a joint collaboration with the University of Sofia by using the ISA-88 standard (see Chapter 4.4.1) [Ivan09]. The batch procedures are specified as Procedure Function Chart that are transformed to SIPNs for formal verification and finally, translated to an implementation according to IEC 61499. Other approaches map the ISA-88 procedure models directly to an IEC 61499 implementation [deSo10][Pelt07][Thra07b]. Lepuschitz and Zoitl combined the concept of Automation Components (see Chapter 4.3.2) with the ISA-88 for developing process control applications in IEC 61499 [Lepu08]. Thereby, considerations concerning the equipment and the physical structure of the production plant are taken into account during the design of the procedural elements executed by Automation Components. Pang and Vyatkin describe an engineering process of Intelligent Mechatronic Components implemented in IEC 61499 by using CAEX as design language [Pang10].

The working group Automation Technology of the University of Halle-Wittenberg defined a hierarchical multi-layer architecture for distributed control systems. By software components controlling the plant elements (tasks) from process algorithms determining the functionality of the plant, the conflict between reusability and flexibility can be defused [Miss07]. In further works a methodology for a model-driven controller design based on a formal plant model was developed. Thereby, the control algorithms of IEC 61499 FBs are derived directly from the specified plant behavior [Hani09].

Leitão and Colombo developed a methodology for the design of agent-based automated production systems using High-Level Petri Nets (HLPN) [Leit06]. First, the dynamic behavior of the automated manufacturing component is modeled in a top-down manner. Thereby, the system is decomposed in reusable units by means of exploding transitions that represent encapsulated Petri Nets again. After the design the model is validated and implemented either according to IEC 61131-3 for low-level control or JADE for high-level control.

In subsequent work HLPN were also used for the design of control architectures based on SOA within the SOCRADES project [Mend08b]. Therefore, different types of components were defined that interact in a service-oriented manner. Four types of components are defined: Mechatronic Components that provide atomic services, more complex Smart Mechatronic Components with a build-in logic control, Process Control Components (PCC) for coordinating processes, and Intelligent Support Components for supporting the control activities like exception handling [Mend08a]. To execute the desired production process, the logic controller of the PCC interprets a process model represented as a HLPN and calls the necessary services according to the orchestration principle [Mend08c]. For implementing the services of the control components the DPWS technology was used (see

Chapter 3.4.5). A prototypical software, the Continuum Development Tool, was developed to design and execute the HLPN control processes including a DPWS interface for calling the services [Mend09a].

4.3 Comprehensive Production Engineering Concepts

One of the discussed deficits of today's situation of control engineering is its insufficient representation and integration in the production planning process (see Chapter 2.3.4). Despite this fact, there already exist concepts for a more comprehensive engineering process by expressing interdisciplinary views and connections among the planning disciplines.

4.3.1 Integrated Engineering

The term integrated engineering collects methods that permit seamless and lossless information exchange between different planning disciplines and phases. Each discipline focuses on the design of certain engineering aspects of the overall system. Due to dependencies between planning disciplines information has to be exchanged between them to coordinate interdisciplinary engineering tasks [Fay12]. Besides classical input and output relations of sequentially executed engineering phases, more and more tasks are done in parallel calling for continuous information exchange [Drat11]. The exchange of engineering data can either happen manually or with a much higher efficiency in a semi-automatic or fully automated way.

Today, engineering IT structures are heterogeneous so that tool chains are characterized by various different engineering tools with specific and often proprietary data formats [Schl08b]. The control engineering already comprises the planning of various hardware and software properties where special tools for designing control strategies, programming, hardware configuration, testing, etc. are used [Esté12]. Usually, the individual tools—especially those from different vendors—have no common interfaces for data exchange so that the user must perform manual transformations from one tool to another [Esté12]. This so-called “paper interface” is error prone and requires high efforts [Nof09][Voge05]. Consequently, information exchange between planning disciplines is characterized by inconsistent information, information losses, additional effort, and time losses.

To reduce efforts, electronic data files are more and more used to save planning results in a machine-processable way. However, the data formats are usually not standardized so that the information has to be transferred to the respective input format. Mechanisms for a consistent data exchange are required in order to ensure a smooth workflow during the whole production engineering [Pang10]. A fully automatic data exchange enables the direct exchange of engineering data so that the consistency of data within a tool chain is guaranteed, no additional efforts are necessary, and conversion failures can be avoided. This can be achieved by agreeing on data standards to ease the cooperation of different

engineering tools (see Figure 4-11). Recently, the most common approach goes towards XML-based data formats [Nof09]. A vendor-neutral solution for the standardized data exchange based on open XML data formats is proposed by the AutomationML standard [Drat09].

Nevertheless, standardization of data formats does not solve the problem when contradictions appear during a parallel design of different engineering domains. A more wide-ranging approach for an integrated engineering to optimize the coordination of the disciplines involved and to ensure consistency and interoperability between them, are cross-discipline planning models [VDI10]. This implies that uniform information models exist that include all disciplines and interfaces between them, resulting in a cross-discipline view of the plant. This kind of modeling constitutes the fundament for an *integrated digital production engineering*, which enables the simultaneous work of mechanical, electrical, and control engineers using up-to-date, complete, and consistent data sets [Schm05]. Approaches to realize such integrated engineering models are presented in the following sections.

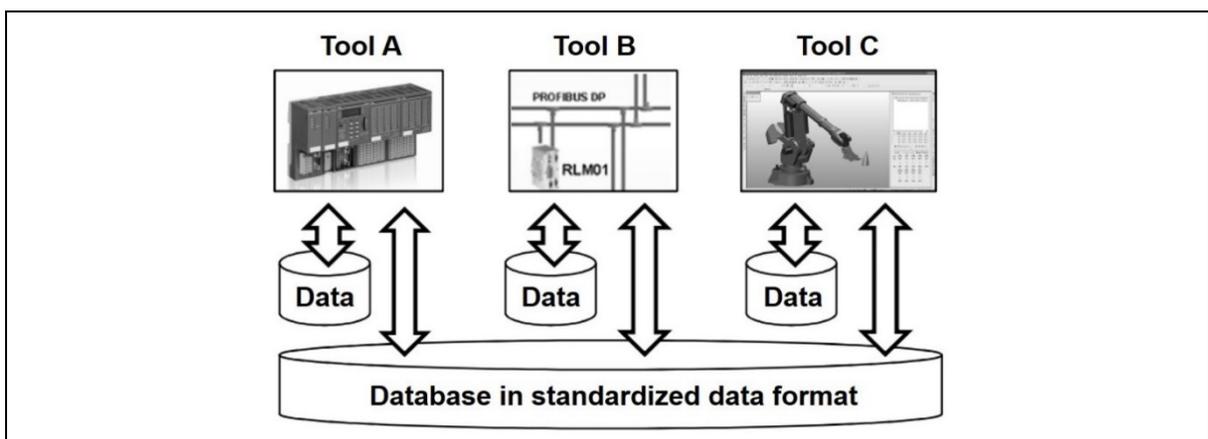


Figure 4-11: Data exchange between tools by standardized data format [Drat11]

4.3.2 Mechatronic Systems

A mechatronic system is an aggregation of mechanics, electrics, and software parts [Vall11]. This interdisciplinary combination is typical for today's automation technology to realize technical processes with production equipment consisting of mechanical and electrical hardware which is controlled by software [Voge09a]. A concept becoming increasingly popular is the design of production systems as a structure of mechatronic components by combing component-based automation with a modularized hardware structure. Thus, the key feature of a mechatronic component is a combination of physical and functional modularization. Vyatkin defines Intelligent/Smart Mechatronic Components that are equipped with their own embedded computers (see Figure 4-12) [Vyat03]. The general idea is that physical components come with pre-programmed software

implementing various programmable control functions and also providing some network interfaces and memory capacity [Pang10]. A similar concept by Sünder introduces Automation Components (AC), which comprise production equipment, embedded devices, and software to implement the logic and diagnostics for the functionality it provides [Sünd06].

By means of modularization, similar advantages for the development process as those for software systems can be leveraged (see Chapter 4.1.1). The complexity of engineering can be reduced by combining and reusing mechatronic production modules that can be selected from a pre-defined repository and adapted by distinct parameters [Voge09b]. Thereby, the development can be driven from a functional-oriented design where the physical details of the system are specified in a later concretization step [Weyr11]. There exist numerous approaches following this principle.

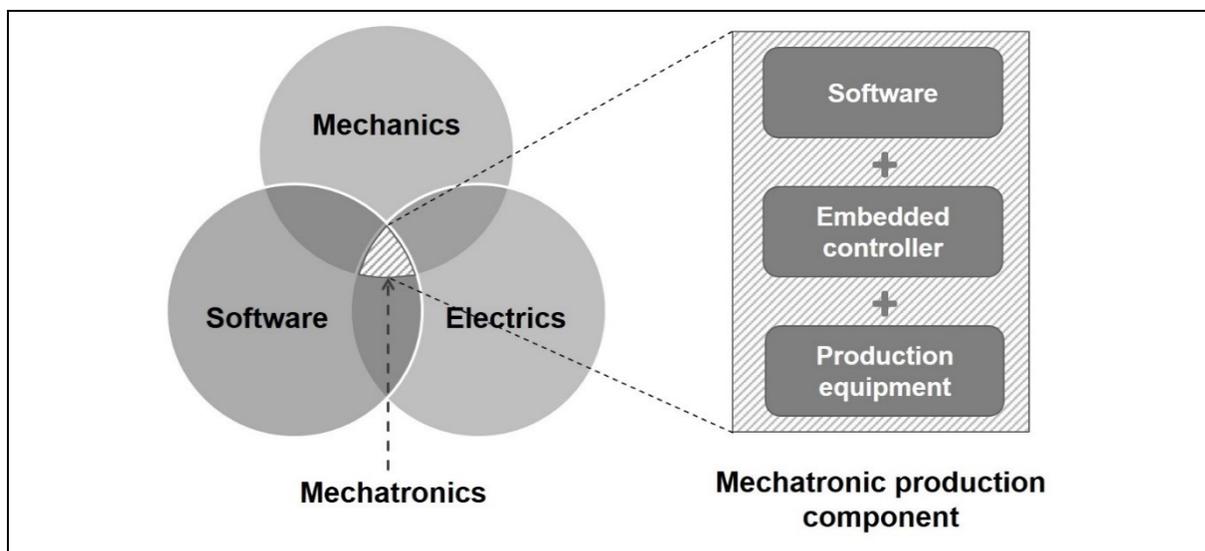


Figure 4-12: The elements of mechatronics in general (left) and of a mechatronic production component (right)

A mechatronic engineering process using mechatronic units was developed at the University of Magdeburg [Lüde10]. It starts with the process planning and the specification of the manufacturing functions, followed by a mapping of these functions to physical manufacturing resources. A mechatronic-oriented digital production engineering process for the automotive industry developed by Kiefer uses a central planning and data platform by a mechatronic plant model [Kief08]. This model promotes parallel execution of different planning activities and validation of the planning results with digital simulation tools [Kief06]. Another development process for mechatronic systems (MTS) specified by Thramboulidis adopts the typical phases from software engineering [Thra05]. The whole production system is designed as an aggregation of interconnected MTSs that collaborate to provide the required system behavior implemented as distributed FBs according to IEC 61499 (see Chapter 3.3.1).

4.3.3 Object-oriented Engineering

Primary, OO is known as a programming paradigm (see Chapter 4.1.1). However, it is a general principle that allows the design of any modular system architecture promoting high reusability [Secc07]. Furthermore, object-orientation can be applied to describe systems consistently in several abstraction degrees so that the system can be concretized step-by-step [Voge09a].

Applied to the engineering of production plants, object-orientation is heavily related to mechatronic design, since objects are well suited to represent mechatronic units in an abstract way (see Chapter 4.3.2). This permits a functional description of the system across all planning disciplines to promote an integrated system design on multiple abstraction levels and a high reusability of planning aspects [Schü09]. In lower detail levels certain planning aspects can be handled as individual objects that represent physical or logical planning entities like a robot or a function block of a PLC program [Drat09]. For planning a concrete production system, the individual objects can be first designed in an abstract way and stepwise concretized by allocating respective classes from a library (see Figure 4-13) [Drat09].

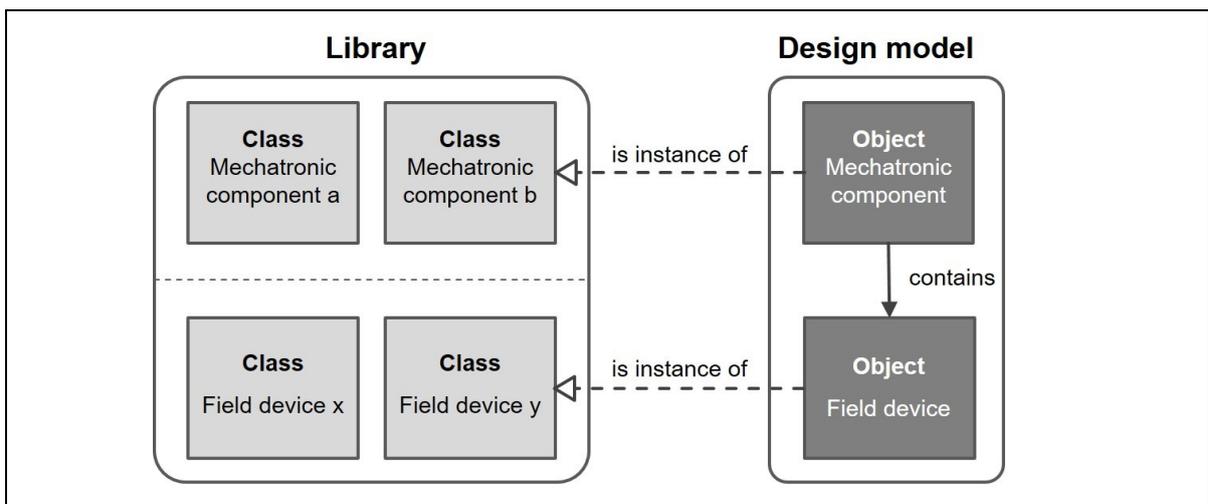


Figure 4-13: Principles of object-oriented engineering

Several approaches exist that use aspects of OO for plant engineering methods, often in connection with UML (see Chapter 4.2.2). As an instance, the German research project “Increasing Efficiency and Quality in PLC Programming by Object Orientation and UML” an object-oriented engineering method for control applications in CoDeSys V3 was developed where classes represent mechatronic units [Voge09a]. Another example is the dissertation of Bergholz which defines object-oriented factory planning by structuring production systems in interacting and encapsulated objects that represent production units [Berg06]. The concept of OO is used to promote reusability and gradual planning by designing and instantiating hierarchical factory objects represented as classes that are created and

connected by applying the OO principles of inheritance and association. The fundamental architecture of the object-oriented modeling is based on the PROSA architecture stemming from holonic control design (see Chapter 4.3.4).

The existing approaches have in common that the object-oriented engineering is strongly driven by the physical structure of the plant consisting of similar or even identical components [Preu11]. This is expressed in the fact that objects mainly represent hardware components (e.g., “machine”) that dispose several functions as the objects methods (e.g., “machine.method”).

4.3.4 Holonic Manufacturing Systems (HMS)

In 1968, the Hungarian author and philosopher Koestler described the concept of Holonic Systems [Babi05]. He observed that most biological or social systems are built on a hierarchical structure so that he describes the units of these systems as holons. The word “holon” stems from Greek combining the two words “holos” and “on” meaning “whole part” as a composed term. In the 1990’s the paradigm of Holonic Manufacturing Systems (HMS) was developed by a consortium in the framework of the Intelligent Manufacturing (IMS) program. The aim was to improve the understanding of the requirements for future-generation manufacturing systems and to enable easy configurations, extensions, modifications, and higher flexibility to satisfy these requirements [Brus98]. Regarding manufacturing a holon is an autonomous and cooperative building block of a manufacturing system that consists of an information part and often a physical processing part [Chri94]. A holon acts autonomously, cooperates with other holons within a holarchy and can be part of another holon. The aggregation principle and the combination of hardware and software enable an interdisciplinary view on manufacturing systems similar to mechatronic systems.

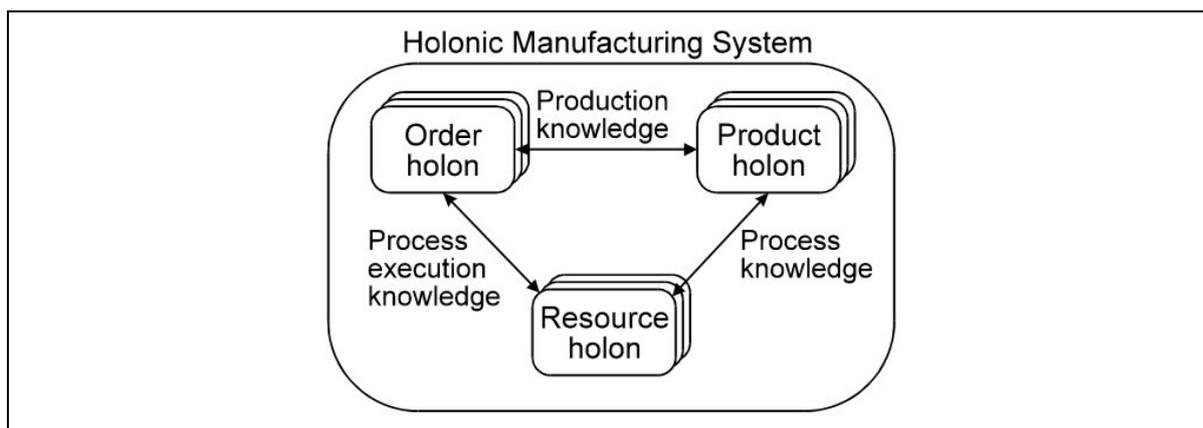


Figure 4-14: Basic elements of the PROSA architecture [Brus98]

Existing applications of HMS focus on the design of distributed control systems. Since most approaches use agents (see Chapter 3.3.2) to implement HMS, research activities regarding HMS and MAS are strongly related [Babi05]. A well-known reference architecture

for HMS named PROSA comprises three different types of holons and their relationships (see Figure 4-14) [Brus98]. An overview of other existing HMS activities is given in a summary of Babiceanu and Chen [Babi06]. Two very similar concepts to HMS are Bionic Manufacturing and Fractal Factories [Thar98]. Bionic Manufacturing is inspired by biological systems and uses parallels between cells in an organism and manufacturing units. The Fractal Factory concept invented by Warnecke describes a manufacturing company as composed fractal entities that collaborate in a dynamic and self-organized way [Warn93].

4.3.5 Systems Engineering

Systems Engineering (SE) is a multidisciplinary approach to develop balanced system solutions in response to diverse stakeholder demands including the application of both management and technical processes [Frie11]. It focuses on the initial development of complex products based on the definition of customer needs and required functionality early in the development cycle [Brec11]. Thereby, the development process starts with the definition and documentation of system requirements and ends with the verification of the system to check the compliance with these requirements [Weil08]. During the development, all the disciplines and specialty groups are integrated into one team forming a structured development process.

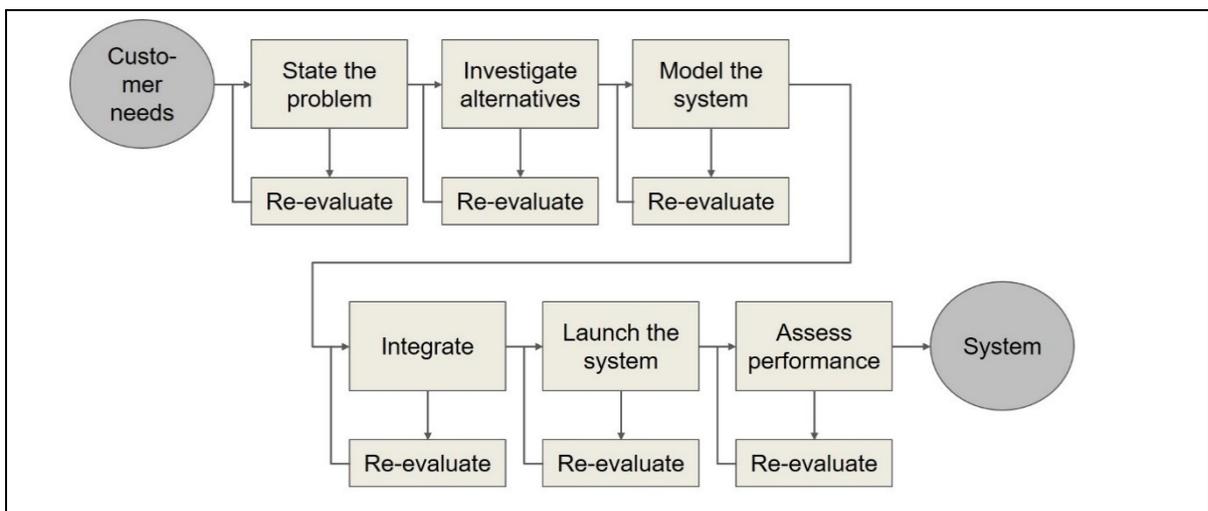


Figure 4-15: The SIMILAR process [INCO15]

A SE process describes the interacting activities which transform the inputs, i.e., the requirements, into outputs, i.e., the dedicated system [Ramo10]. It usually comprises the following seven tasks: state the problem, investigate alternatives, model the system, integrate, launch the system, assess performance, and re-evaluate, abbreviated as SIMILAR process (see Figure 4-15) [INCO15]. Thereby, the whole SE development process is strongly driven by the requirements on the system. A requirement describes one or more properties or behaviors of the system that always have to be met [Weil08]. A distinction is made between functional and non-functional requirements like usually done in software

engineering [Gals08]. Functional requirements define how a system reacts to certain inputs and how it should behave in particular situations [Wehr09]. In contrast, non-functional requirements contain all requirements that are not directly dealing with the function of the system but which influence them, for example modularization, performance, safety, security, etc. [Fran11].

The International Council on Systems Engineering (INCOSE) initiated an effort with the OMG to extend the UML for full-lifecycle systems engineering resulting in the modeling language SysML (see Chapter 4.2.2) [Bock06].

4.3.6 Planning of Service-oriented Factory Control Systems

The paradigm of SOA has already been applied in many research activities in the field of production automation (see Chapter 3.3.3). The dissertation of Pohlmann also makes use of a service-oriented automation approach as the basis of a methodology for process-oriented planning of factory control systems [Pohl08]. Hereby, a factory system consists of loosely coupled services in order to achieve a high degree of adaptability and interoperability.

The methodology defines three types of services:

- **Device service:** represents the functions of the sensors and actuators in order to couple the system design to the technical equipment
- **System service:** enables the development of system individual automation software
- **User service:** provides functions for the interaction between users and the factory system, which can be used for the development of human-machine interfaces

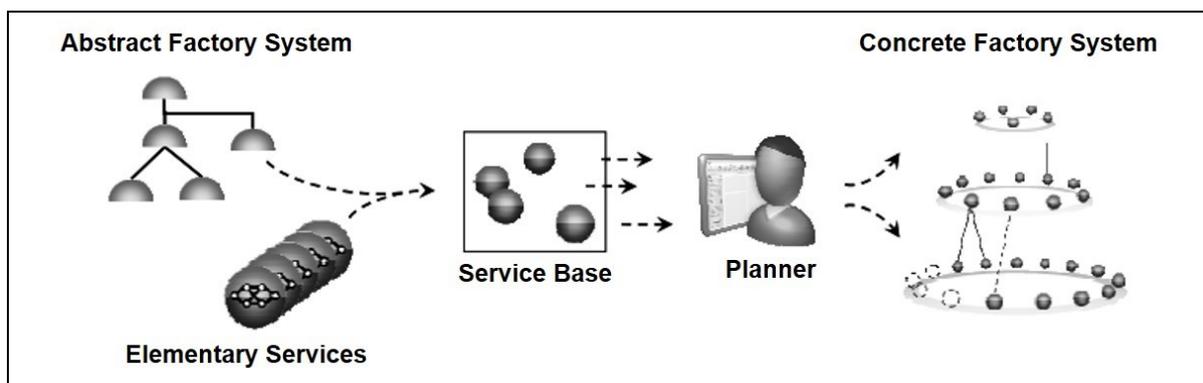


Figure 4-16: Service-oriented factory planning method [Pohl08]

The description of a service consists of an abstract and a concrete part to support a planning procedure with two abstraction levels (see Figure 4-16). During the abstract planning the production process is described in a process model using the mere functionality of a service. This functional planning of the production process takes place in various levels of detail according to the top-down principle and is independent of the specific hardware. Later on,

the actual hardware realized as production modules is allocated so that the services are concretized with device-specific information during the concrete planning.

The expected outcome of this methodology is a reduction of planning efforts by employing reusable production modules represented as services and a reduction of planning time through the parallelization of planning tasks during the detailed planning phase. Furthermore, the process-orientation supports a better integration of business planning with production engineering, which also leads to a higher planning efficiency. However, the prerequisite to leverage these benefits is an already modularized production equipment that provides its functionality as encapsulated services. Additionally, the services have to be implemented directly on the production hardware with a standardized SOA technology. Since these prerequisites have yet to be established, the methodology needs to be further developed and detailed for gaining practical suitability. So far a prototypical implementation of the planning methodology has been realized in form of an experimental demonstrator of the *SmartFactory*^{KL} with Web Service technologies [Pohl08].

4.4 Engineering Standards and Guidelines

In the following, a selection of engineering standards and guidelines are presented that can be supportive to apply the before mentioned concepts for an efficient production planning and control engineering.

4.4.1 Reference Architectures According to ISA-95 and ISA-88

To support the modeling of automated production processes, reference architectures are helpful to provide blueprints for the structure for an instance model. The international standard ISA-95 „Enterprise-Control System Integration“ defines hierarchical models of production organizations and provides concepts for the integration of control systems with business IT systems of the enterprise [Vrba09]. Thereby, an equipment hierarchy model represents classes of physical assets involved in the manufacturing of an enterprise [ISA00]. The model depicts how equipment entities from lower levels are combined to form entities on higher levels in the hierarchy. The terminology of entity types on the lower levels varies depending upon the type of industry they apply to (see Figure 4-17) [John08].

A related standard is ISA-88, which particularly addresses batch control and aims to standardize batch control systems [ISA95]. The standard describes batch control from different viewpoints [Olss05]. The functional view is represented via a process model and the hardware view specified by a physical model. Additionally, recipes uniquely define the manufacturing requirements for a specific product [Virt10]. One aspect of the recipe is the procedure that defines how the production equipment needs to be controlled to produce the desired product.

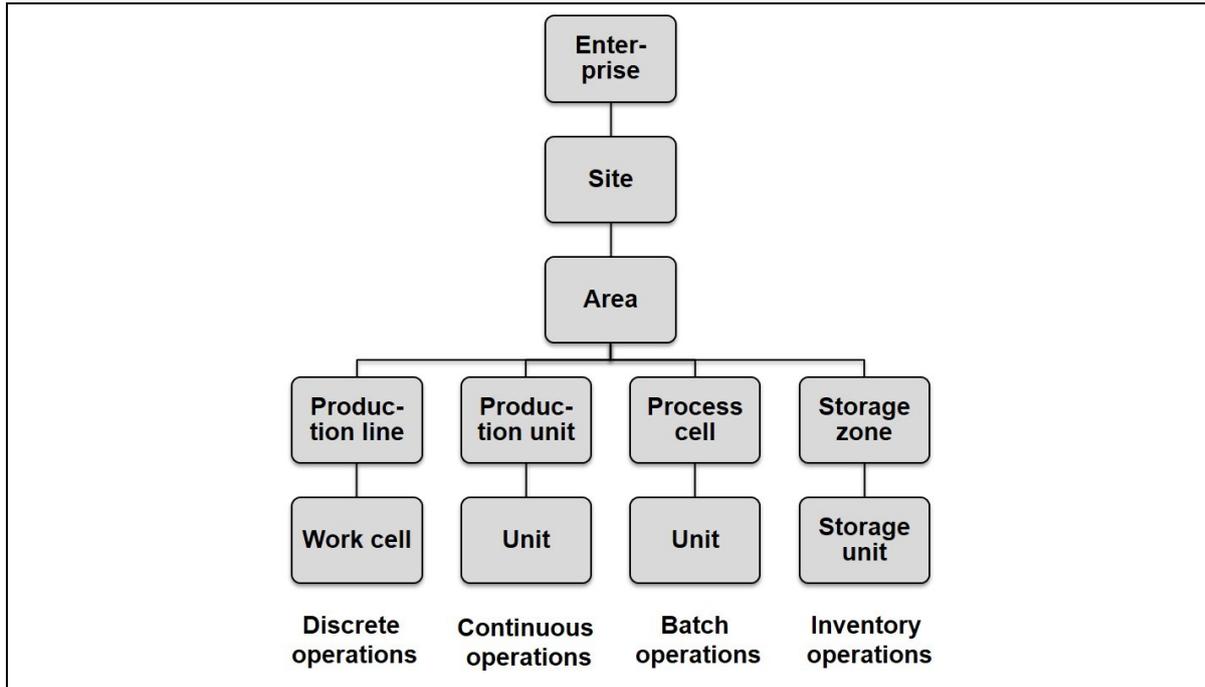


Figure 4-17: The ISA-95 equipment hierarchy [ISA00]

The reference architectures for the models, recipes, and procedures have several hierarchical layers representing different degrees of granularity. Thereby, recipes (recipe model) correlate with production equipment (physical model) on several layers to deliver functionality for executing a dedicated process (see Figure 4-18). Furthermore, the standard defines the relations between single classes of the reference models. Hence, the ISA-88 standard combines different views on a batch system and points out the dependencies between physical equipment, control procedures, and the batch process to execute.

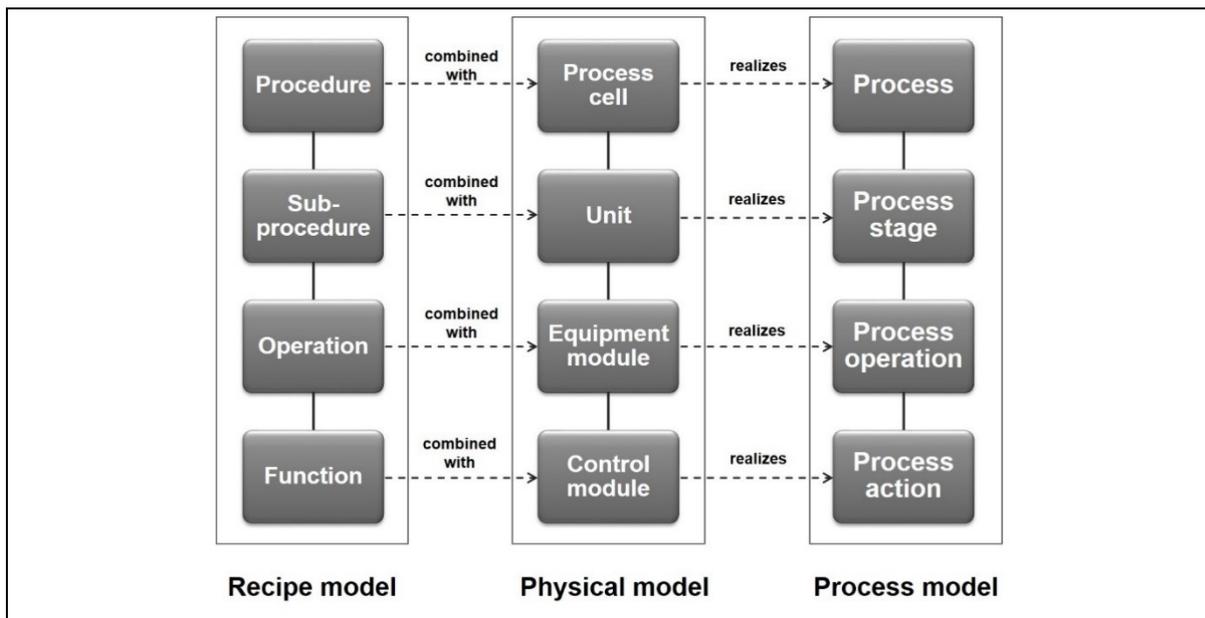


Figure 4-18: ISA-88 recipe model, physical model, and process model [ISA95]

4.4.2 Standards Providing Uniform Terms for Modeling

During modeling the question usually arises which terms for the objects to be modeled should be picked, in this instance of the previous chapter terms for manufacturing equipment and functions. Unfortunately, a general and open standardized terminology covering the whole automation domain is missing. Big companies often have their specific terms defined by their own automation standards like the Integra standard of Daimler [Siem04]. In contrast to these proprietary standards, some universal and publicly accessible automation standards exist that could provide at least a foundation for a standardized terminology.

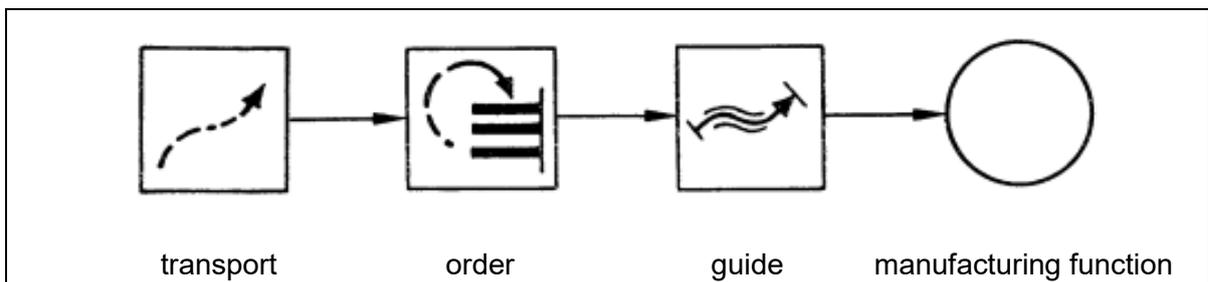


Figure 4-19: Example of a graphical representation of a manufacturing process according VDI 2860

The German standard DIN 8580 “Manufacturing processes — Terms and definitions, division” defines types of manufacturing functions to produce geometrically defined solid objects [DIN03]. It contains six main groups of manufacturing processes: primary forming, deforming, cutting, joining, treating, and change of material property. Each main group refers to a number of groups that are divided into subgroups again. Altogether, the standard defines manufacturing functions on three detail levels and links to other standards that describe respective sub groups in detail.

Within the VDI guideline 2860 assembly and handling functions are defined including graphical symbols for them [VDI90]. The main function “assembly and handling” is divided in five sub functions: store (keeping quantities), modify quantities, move (generating or change spatial arrangements), lock/maintain (keep spatial arrangements), and check. A further subdivision of these sub functions takes place with elementary functions and composed functions. The elementary functions act as atoms that have no further sub functions, whereas composed functions can be subdivided into elementary functions. The guideline also proposes to model manufacturing processes by using the assembly and handling functions in combination with the manufacturing functions of DIN 8580 and specifies the graphical representation for the single types of functions and manufacturing processes (see Figure 4-19).

The PLCopen standard “Motion Control” provides a concept for realizing standard PLC libraries for motion control applications that are reusable for multiple hardware platforms

[PLCo11a]. Part 1 of the standard “Function Blocks for Motion Control” specifies a standardized IEC 61131 FB library and a state diagram defining eight individual states and how FBs lead to change of state [PLCo11b]. The input variables, output variables, and the behavior of each FB is explained and examples of interaction of various FBs are given.

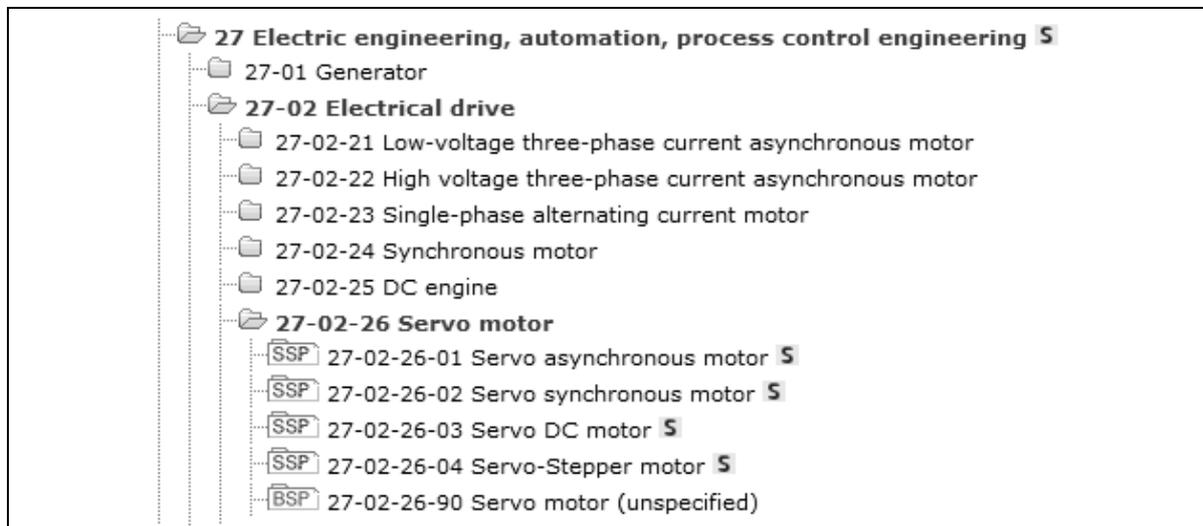


Figure 4-20: eCI@ss example for the group “Servo motor”

For naming production equipment in a uniform way, the eCI@ss standard provides an extensive library of product classes and properties. Besides automation technology and manufacturing equipment eCI@ss comprises many more domains like commerce, crafts, and food. Its main purpose is to be a cross-industry product data standard for classification and clear description of products to support company-wide applications such as procurement, controlling, supplier management, and engineering [eCI@14]. The classification system is structured in four hierarchy levels: segment, main group, group, and sub-group.

4.5 Assessment of the Concepts

Several concepts have been reviewed to promote and support new methods for a more efficient control engineering. A promising proposal is to combine innovative concepts for a more comprehensive production engineering with well-established concepts from computer science to enhance the actual programming. By this, an ideal connection between existing IT concepts and their application for the AUT domain can be generated.

For this purpose, the two well-known and proven programming paradigms modularization and OO have been presented. They provide important principles for encapsulation of code so that a higher abstraction degree for the development of control procedures can be achieved. Moreover, life cycle models for the software development have been introduced. They can help to transfer today’s typical bottom-up development of control procedures to a structured top-down development method with defined development steps. A further

concept from software engineering for rising abstraction is MDE. It promotes models for capturing the modularized engineering results in a comprehensible way, which can be detailed in several concretization steps to support the top-down development. For general modeling purposes, particularly UML and SysML constitute widely accepted modeling languages.

For linking the development of the control procedures with the overall production engineering, several approaches for a more comprehensive and integrated design have been reviewed. The concept of Systems Engineering provides ideas how the requirements on the overall production system can be specified and the properties for the individual planning domains can be derived during the early design phases. Thus, the control engineering can start in an earlier point in time and in parallel with the other engineering domains. This strengthens the consideration of the control engineering within the production engineering process.

A popular and promising interdisciplinary design concept is the mechatronic system design. A production system as mechatronic system is split up in a number of mechatronic components that comprise mechanical, electrical, etc. hardware and automation functions. Such a modular system architecture can be well combined with an object-oriented design principle. Thereby, mechatronic components are represented as objects that are concretized within several steps. Moreover, the application of library concepts enables a high degree of reusability of planning results. Since control programs are characterized by a strong dependency on the production equipment they control, the engineering domains cannot be executed completely independent from each other. However, existing mechatronic engineering approaches mainly focus on the modularization of the hardware to build mechatronic components whereby the automation functions are defined subsequently for each mechanical module. This impairs a parallel execution of the domains as described by the V-model for mechatronic system design [VDI04]. In contrast, a strong functional-driven planning approach is given by the methodology for a process-oriented planning of control systems [Pohl08]. The combination of this methodology with the other mentioned concepts seems promising to develop an enhanced control engineering process that considers also its dependencies to early design phases and to other engineering domains like the mechanical design.

5 Problem Statement, Objective Target, and Procedural Method

5.1 Problem Statement

The ability to build and adapt production systems quickly and efficiently according to today's agile market conditions constitutes a major competitive advantage of producing companies (see Chapter 2.1). This demand requires advanced production engineering processes which allow a lowering of efforts for design and realization as well as parallelization of planning tasks for a shorter time-to-market (see Chapter 2.3.2). One engineering domain with much room for improvement regarding both requirements and with a simultaneously increasing importance is the domain of control engineering (see Chapter 2.3.3). Today, it is typically executed at the very end of the detailed planning phase sequentially after the hardware design (see Chapter 2.3.2). Efforts for developing and changing control software are high due to a low implementation level and monolithic program structures (see Chapter 2.3.3). Furthermore, missing abstraction mechanism and methodological design procedures lead to a bottom-up development of the program code with an insufficient integration with other engineering domains. Altogether, a mix of inadequate engineering methods and the technical restrictions of conventional control systems lead to this inadequate situation of control engineering (see Chapter 2.3.4).

A promising approach to improve today's situation is the further development of control systems by applying concepts from distributed systems. This implies component-based software architectures enabling highly flexible and adaptable control architectures and a better handling of complexity through encapsulation as an abstraction mechanism (see Chapter 3.2). Three concepts for realizing distributed control architectures are the IEC 61499 standard (see Chapter 3.3.1), Multi-agent Systems (see Chapter 3.3.2), and Service-oriented Architecture (see Chapter 3.3.3). Although research applications exist, all three concepts have not arrived in industrial practice yet. This is particularly due to the lack of mature technologies and design methods that support the specific requirements of automation applications (see Chapter 3.3.4). SOA offers good prerequisites since its fundament is strongly characterized by a process-oriented design pattern, which enables a straightforward top-down development (see Chapter 3.4). Moreover, detailed design methods and guidelines to develop business IT applications as SOA already exist.

This reflects that besides the characteristics and the accompanying technical possibilities of the control system itself, the development methodology is of crucial importance. Thus, a systematic design method considering the integration of control engineering within the

overall production planning process is essential to optimize the overall development time and efforts. There exist several promising engineering approaches supporting a more comprehensive view and consistent planning of production systems (see Chapter 4.3). However, those concepts describing concrete engineering methods—especially mechatronic systems and object-oriented engineering (see Chapter 4.5)—focus on designing the mechanical structure by focusing less on the software development.

Most of those comprehensive approaches have in common that they make use of proven software engineering concepts. Particularly model-driven engineering and object-orientation are very promising concepts to improve the development of control procedures (see Chapter 4.1 & 4.2). Although these concepts have already gained a lot of attention, the use in practice is often restricted, which is in turn due to today's technical conditions of control systems. For example, a seamless model-driven engineering is hindered by the gap between the modeling of the application and the final implemented and executable control application [Tran06].

Altogether, many potentials to enhance control engineering are already available but not fully utilized yet. A holistic and systematic engineering method for control procedures making use of the advantages of distributed control systems in combination with comprehensive engineering concepts and software development methods is still missing.

5.2 Objective Target

The objective of this thesis is the development of a model-driven engineering methodology for the development of service-oriented control procedures for automated manufacturing processes. Therefore, the basic principles and design methods of SOA are applied to control engineering and combined with concepts from software engineering, specifically, MDE.

To overcome today's gaps for an improved efficiency for control engineering the following requirements are derived:

- **Reduced programming effort:** Control procedures are created on a higher abstraction level according to the building block principle with services as pre-programmed components. Additionally, a high reusability of services enables the use of the same services for different use cases.
- **Better handling of complexity:** A clear separation of logic for the production equipment and logic for the process logic as well as the possibility to create various granularity levels enhance the comprehensibility and scalability of control software.
- **Increased adaptability:** Modular program structures can be changed and extended much more straightforward than monolithic programs. Thus, control procedures built as service compositions are highly adaptable and permit reconfigurations of the whole production system with lower efforts.

- **Top-down engineering:** Applying SOA principles enables control system development according to a process-oriented design driven by the specification of the production process. Control procedures can then be first specified in a solution-neutral way and gradually refined with hardware-specific details. This top-down approach facilitates various concretization steps with an integrated information flow.
- **Parallelization of engineering domains:** The before mentioned top-down engineering can act as the enabler to parallelize tasks during production engineering. First, the specification of the production process determines the functional requirements on the whole production system and constitutes the starting point for the detailed planning phase. Afterwards, the details of the respective engineering disciplines are specified. A parallel execution of the domain-specific design according to the V-model for mechatronic design can be obtained by determining the dependencies and defining concrete links between the individual disciplines (see Figure 5-1).

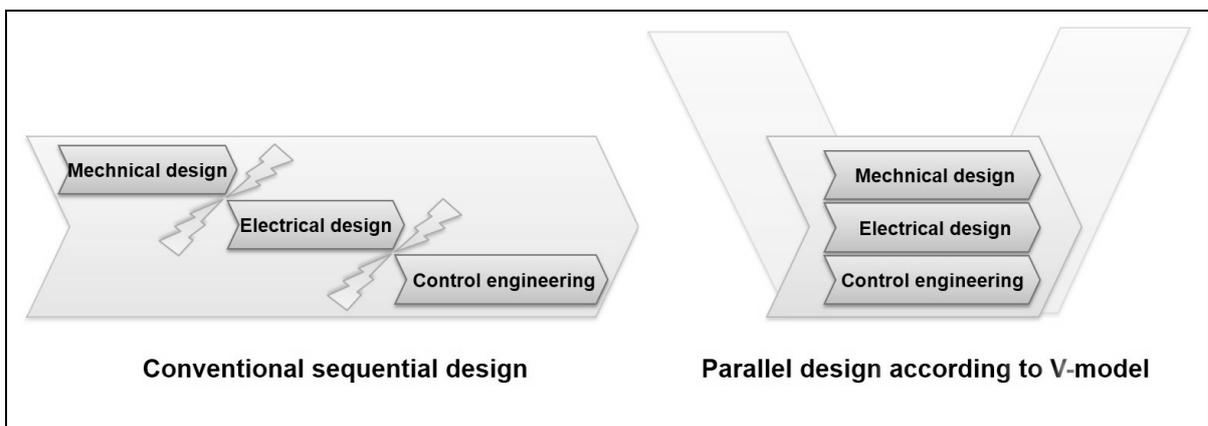


Figure 5-1: Sequential vs. parallel domain-specific design

These potentials are leveraged by a methodology that defines how service-oriented control procedures are developed. The process-oriented design and orchestration concept of SOA are key drivers to enable a top-down design with several concretization steps. To obtain an integrated design procedure with respect to the overall production planning process, the links of control engineering to other planning phases and domains need to be considered. Therefore, the basic ideas of Pohlmann's concept for developing service-oriented factory control systems (see Chapter 4.3.6) are picked up and are further developed with focus on manufacturing control procedures. Therefore, innovative aspects of other comprehensive production engineering concepts—particularly Systems Engineering—are used to define a systematic development process.

Combining these design concepts with proven software engineering methods enables a structured engineering process with a comprehensible presentation of planning results. Especially the consequent use of models hereby improves the clarity and comprehensibility

of complex design tasks. The abstraction principles provided by SOA permit exploiting the capabilities of those concepts.

The overall focus of this thesis is the generic design of the software architecture. Thus, the methodology is independently defined on specific tools, data formats, or the technical properties of the control system. In order to show its applicability, an implementation concept is developed that proposes how the individual development steps can be realized and how the designed software architecture can be transferred to a respective system architecture. The concluding proof of concept is executed in an exemplary use-case where the achievements are evaluated.

Altogether, the expected result of this thesis comprises the following aspects inspired by the three criteria of an efficient factory planning process (see Chapter 2.3.1):

- **Modeling and structural concepts:** The general development procedure is described as reference model which defines the planning steps and the meta-models defining how the planning information is depicted.
- **Design concepts and architectures:** Reference architectures specify structural blueprints of the models according to the scope of application, namely control procedures for manufacturing processes.
- **Procedures, methods, and tools:** An application concept defines how the theoretical concept can be applied for concrete problems by using existing standards, guidelines, modeling languages, and software tools.

5.3 Procedural Method

The scientific content of this thesis is structured within six chapters (see Figure 5-2). The chapters 2, 3, and 4 constitute the technical foundation for the methodology to be developed. Since the field of application constitutes control engineering of automated manufacturing processes, an introduction into automated production, industrial automation systems, and specifically programmable logic controllers is given in Chapter 2. The second half of the chapter explains the position of control engineering in the factory/production planning process and how it is characterized. At the end of the chapter an analysis is given about today's situation of control engineering, current drawbacks, and existing potentials for improvement.

Chapter 3 addresses the research topic of distributed design systems. Current trends in automation are described leading to the idea of distributed automation systems. Three concrete design principles for distributed control architectures are described and subsequently compared with each other. After that the concrete design and implementation concepts provided by Service-oriented Architecture are described in more detail.

Many research activities deal with the development and application of new concepts for an efficient control engineering described in Chapter 4. First of all, this includes methods from

software engineering like programming paradigm, life cycle models, and particularly model-driven development. Furthermore, there exist several concepts for a more comprehensive production engineering with the goal of a better integration of the individual engineering domains. Finally, a selection of engineering standards and guidelines is presented which can support the before mentioned concepts.

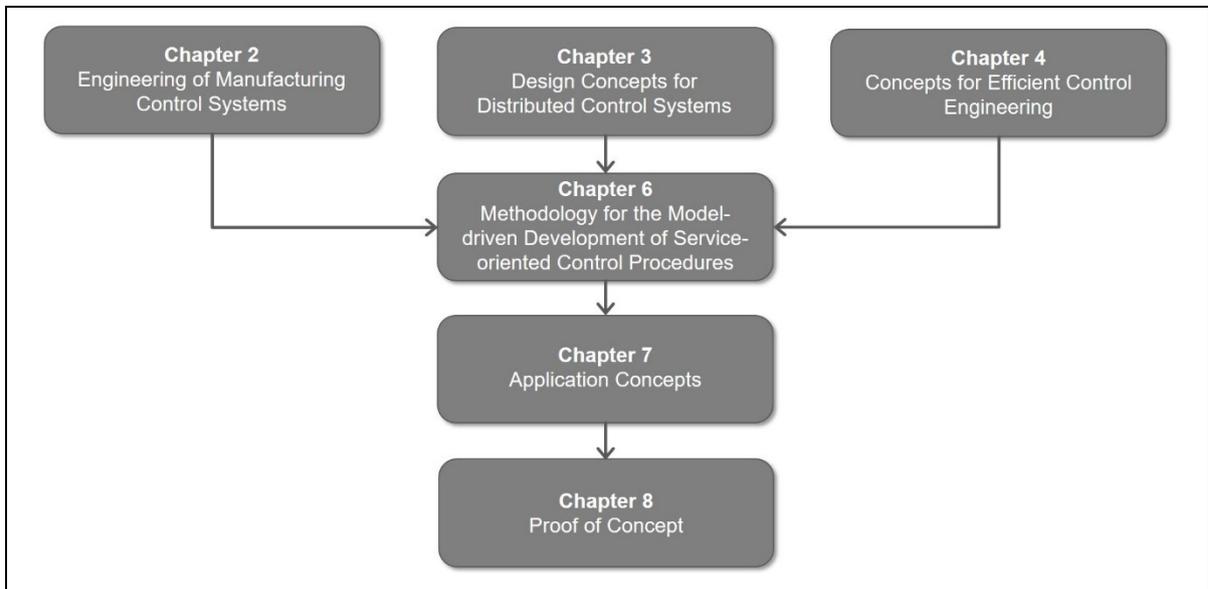


Figure 5-2: Structure of the thesis

The core of this thesis constitutes Chapter 6 which specifies the theoretical concepts of the engineering methodology. Its fundament is created by first transferring the paradigm of SOA from the business IT domain to automation. Based on this a reference architecture for service-oriented manufacturing control is defined considering the key requirements on developing industrial control procedures. By means of a process-oriented design concept an approach for a service-oriented engineering is developed which describes a top-down specification of control procedures and, moreover, an integrated production engineering workflow. Finally, the engineering methodology is particularized in terms of a reference model by describing how each planning step is executed by using which model.

To execute the engineering methodology for real problems an application concept is described in Chapter 7. In the first part, recommendations for the standardized naming of planning objects is given. The second part deals with a suitable system design to implement the service-oriented control architecture on running platforms.

6 Methodology for the Model-driven Development of Service-oriented Control Procedures

Within this chapter the methodology for a model-driven development of service-oriented control procedures—*MDE for SOA-AT*—is developed. The basis for the methodology is provided by the definition of general principles of service-oriented automation, concepts and reference architectures for the design of services, and a procedural model for the engineering process.

6.1 Service-oriented Automation

In production automation, the paradigm of SOA is already established within the highest automation layer where the automation tasks are integrated to the overall business processes of a company. Extending the application of SOA to all levels of the automation domain provides the opportunity to create innovative automation systems where each automation function is encapsulated in a service. Each participant provides its functionality as services and makes these publicly available to others. High-level automation applications can then be generated by composing existing services according to the current demands and conditions. An automation system featuring these properties can be seen as a *Network of Automation* rather than the traditional, hierarchical pyramid (see Figure 6-1).

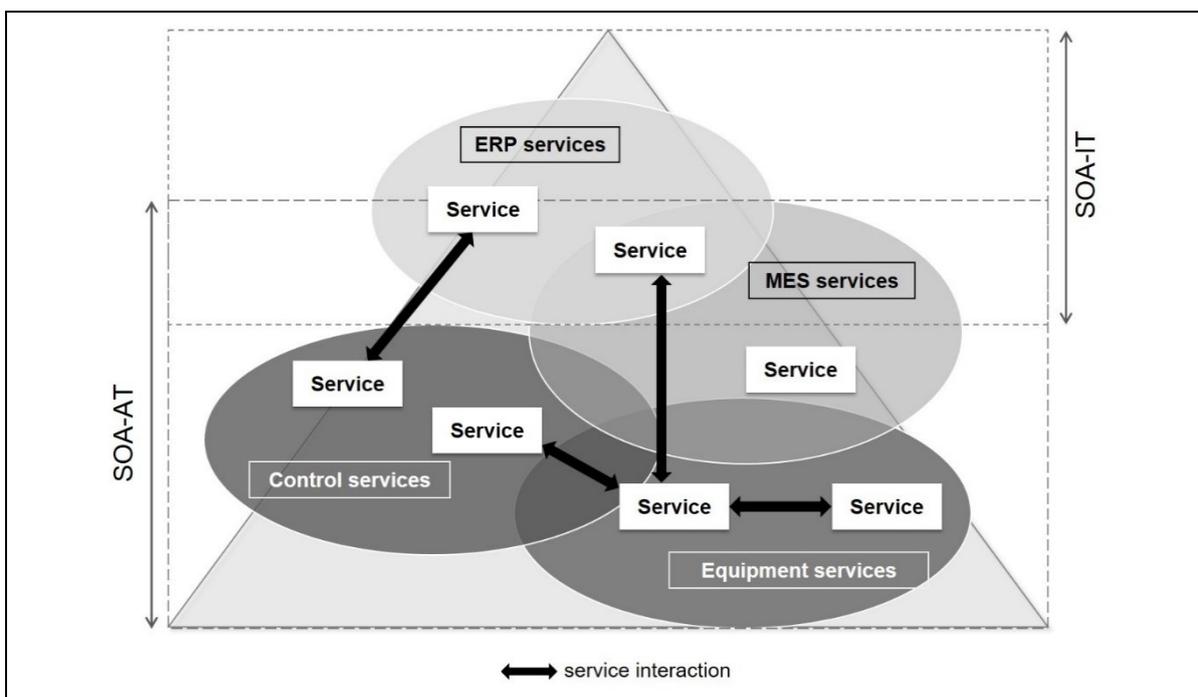


Figure 6-1: The Network of Automation according to the SOA paradigm

The drawbacks of traditional automation systems can be eliminated by strengthening the following properties:

- **Flexibility:** Automation applications can be composed according to the building block principle.
- **Reusability:** The individual services constitute reusable modules that can be used in various automation applications.
- **Scalability:** Service composition enables the abstraction on multiple levels of granularity.
- **Interoperability:** Open-accessible service interfaces and standardized communication technologies enable both horizontal and vertical integration.

Although these general ideas make the application of SOA in automation appear very promising, applying established implementation methods, tools, and technologies one to one is rarely sensible. Automation tasks differ from IT applications in considerably different characteristics and requirements so that the application of SOA in production automation (SOA-AT) needs to be distinguished from those for business IT processes (SOA-IT) (see Figure 6-2).

In SOA-IT services encapsulate pure software functionality, whereas SOA-AT services represent or influence mechatronic functionality that triggers the physical actions of a piece of hardware. Generally, the location where the service is executed doesn't matter as long as existing requirements are met. For pure software, this means that the service can be executed on any processor within the SOA network, if demands on computational properties, time response, etc. can be fulfilled. In case of mechatronic services, the purpose of a service within a process is bound to the respective physical device executing the service. For instance, there can be multiple motors of the same motor type within a production line providing the same services but driving different axes. The impact on the production process can even be heavily dependent on the exact position of the equipment executing the service, for example when a cylinder extends from a starting point to a destination point. If a service is suitable for the current demands, it will therefore not only depend on the service description specifying its functionality but also on hardware-related properties of the physical device like geometric dimensions and the location of the device.

Moreover, the hardware dependencies lead to the fact that multiple and simultaneous use of a service of the same hardware are often restricted, as in the case of a valve that cannot open and close simultaneously [Yu10]. Furthermore, dependencies between hardware and software influence how far loose-coupling can be obtained. For software systems, loose coupling can be acquired by software modularization and a high degree of cohesion. Besides designing the software according to these principles, SOA-AT systems additionally need to be aligned with the hardware structure in terms of a mechatronic design (see Chapter 4.3.2). In conclusion, a general difference can be noted in the overall objective.

The final product of SOA-IT is a software application which is realized as a collaboration of distributed modules within a network. Although the goal of SOA-AT is also to generate software for implementing automation tasks, the main objective is the execution of a technical process which is indeed controlled by the corresponding software.

	SOA-IT	SOA-AT
Field of application	business IT processes	production automation
Service definition	encapsulation of software	encapsulation of software and/or mechatronic functionality
Location of service	usage is independent of location	usage can be dependent on location
Loose coupling	modular programming with high cohesion	modular programming + mechatronic design
Service specification	software functionality	software / mechatronic functionality + hardware description + location
Main objective	distributed execution of a business process	execution of a technical process

Figure 6-2: Comparison between SOA-IT and SOA-AT

Altogether, the general SOA concepts constitute the fundament for each SOA application but for a realization specific development methods are needed. Thereby, existing ones should be transferred to the new application domain with adaptations to its special needs.

6.2 Reference Architecture for SOA-AT

The first step to introduce a systematic and optimal design method constitutes a SOA reference architecture that defines a service structure for SOA-AT with focus on manufacturing control. Regarding the pyramid of automation this reference architecture comprises the functionality of the field devices in layer 1 and the control procedures in layer 2 for implementing the manufacturing process. Today, the implementation of the functionality in software of both layers happens within the controllers on layer 2, particularly in the programs of PLCs and contingently with subordinate controllers (see Chapter 2.2 & 2.3.3). The reference architecture explicitly splits up the control functions into two service layers according to the automation layers. Furthermore, it defines the characteristics of both service layers, their service categories, and the composition strategy according to general SOA design principles (see Chapter 3.4).

6.2.1 Equipment Services

The fundament of service-oriented automation is built by a service layer that comprises the services of the production equipment and is defined as *equipment layer*. Services of this

layer—named *equipment services*—represent the electrical, mechanical, pneumatic, sensory, etc. functionality of the equipment and thus, constitute the interface between the automation network and the technical production process. A basic conceptual question is how these services and their operations are defined regarding the physical devices they belong to.

Generally, a service bundles a set of related operations to carry out a specific functionality. Thus, an equipment service represents the technological functionality of a hardware component that can be accessed via the operations of the service. Within the SOA-AT network the service interface makes the service accessible and hides its implementation details. To make the operations executable, the internal details of the service need to be implemented. Each operation represents a logic that realizes a certain function of the hardware component (see Figure 6-3). The implementation of the logic is equivalent to the programming of the field device functions in PLC code (see Chapter 2.3.3) and thus, it also depends heavily on the respective field device and its communication interface. To control a field device by the logic of a service, a direct information exchange is needed so that actions of the device can be triggered by the logic and the device can give feedback to the logic respectively.

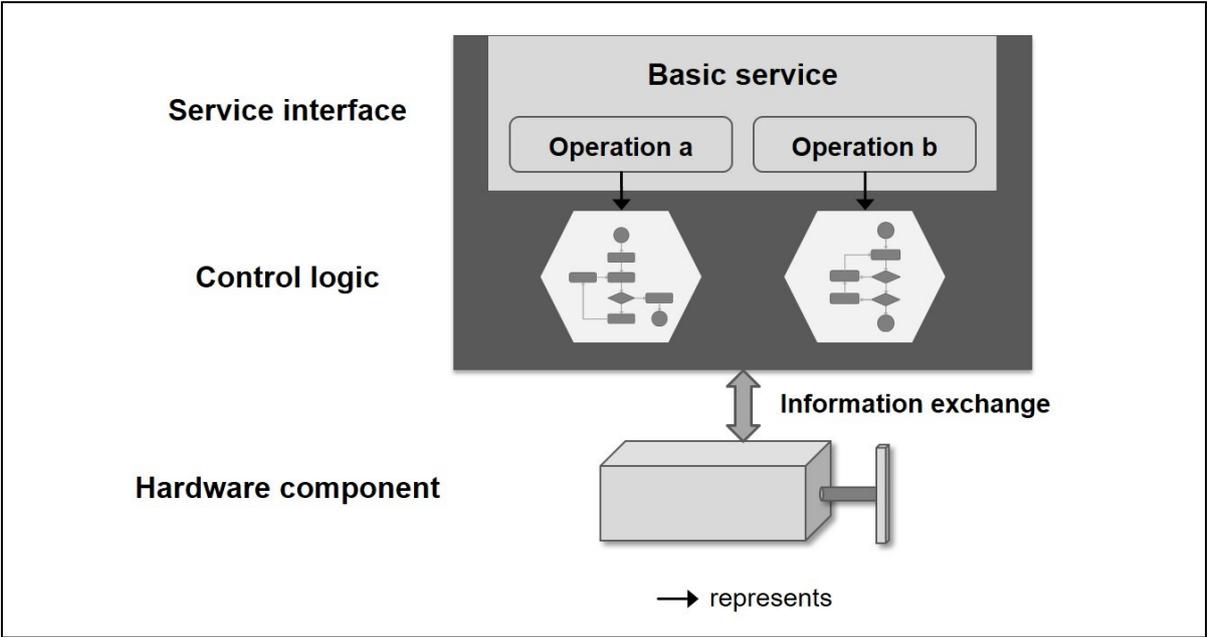


Figure 6-3: Characteristics of a basic service

Equipment services can be defined in several levels of granularity regarding the scope of the functionality they encapsulate by making use of the service composition principle (see Chapter 3.4.3). Thereby, the specification of the services on the lowest level is a crucial task, since they act as the atoms the whole SOA-AT system is built upon. These services are defined as *basic services* and are characterized by a direct interaction with the

production equipment (see Figure 6-3). Thus, each basic service implements the functions as described above and communicates directly with a certain hardware component to influence or access the technical process. Due to this, equipment services are strongly linked to the production equipment so that resulting dependencies to the hardware need to be considered.

From the engineering point of view, the smallest active hardware units are the field devices, which are offered by field device manufacturers. By applying the mechatronic design principle (see Chapter 4.3.2), it makes sense to encapsulate the functionality of each field device to a basic service so that the smallest mechanical units (field devices) match with the smallest functional units (basic services). According to the characteristics of the field device, a basic service either represents an actuator function in order to influence the production process or a sensory function to collect information about the process. As an instance for an actuator, a cylinder provides its functionality of translational motion as a service which comprises operations for moving the cylinder in and out. An example for a sensory function is a service of a temperature sensor for receiving the current measured value (see Figure 6-4 left). Depending on the respective application, it can make sense to deviate from this guideline and to create basic services for bigger equipment units comprising several field devices.

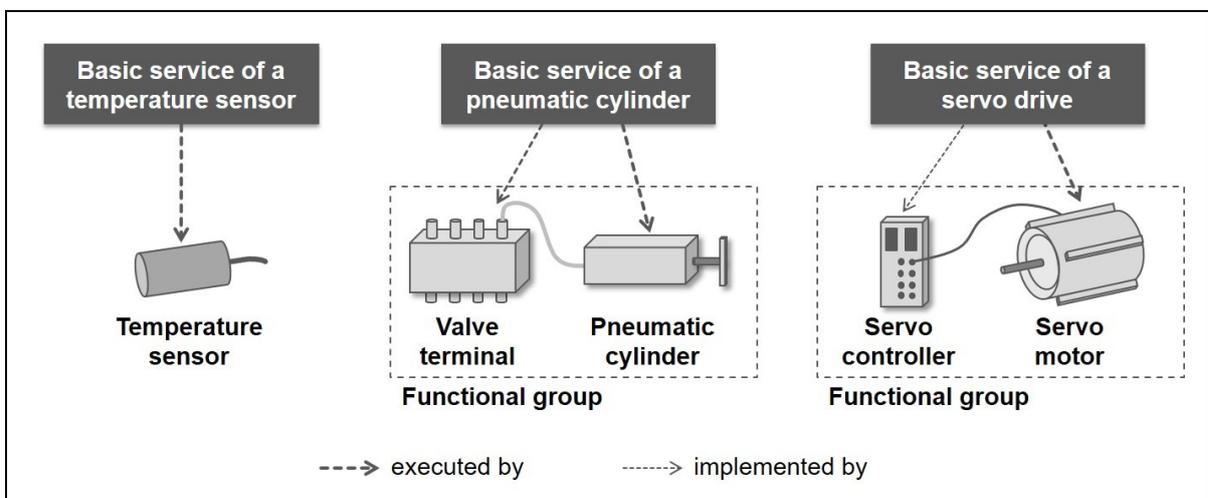


Figure 6-4: Relationships between equipment services and hardware components

Besides the obvious relation between basic services and the hardware components that execute their functionality, additional relationships to other field devices can exist. This occurs particularly for actuators that cannot be directly controlled via control signals of PLCs or other controllers. This implies that intermediate devices are needed that process the control signals and operate the actuator by physical quantities like compressed air or electrical power. In this case, the service would be still executed by the actuator but the functionality itself is implemented by the intermediate device. Since the related devices are

both required for executing a certain function, such a collection of field devices is defined as a *functional group*. Well-known examples of functional groups are pneumatic devices like a pneumatic cylinder that is supplied and controlled by a valve terminal or a servo drive which consists of the motor and a servo controller (see Figure 6-4 center & right).

Besides basic services, another category of equipment services is introduced which is based on the composition of other equipment services. Field devices and manufactured components are typically assembled to bigger physical units as special machines, production modules, etc. These units are built to generate a certain high-level production functionality, which can be also carried out as a service again. Such high-level equipment services are defined as *composed services* (see Figure 6-5). Since the production equipment can be physically structured as hardware components with several granularity levels, multiple levels of composed services can be created. Generally, composed services break the functions of the overall production equipment down for a better handling of complexity and to make special functions of production equipment reusable. This constitutes the prerequisite for realizing highly modularized production systems which consist of mechatronic units combining physical and functional encapsulation (see Chapter 4.3.2). Consequently, composed services can increase the reusability of single production modules as well as the reconfigurability of the complete production system.

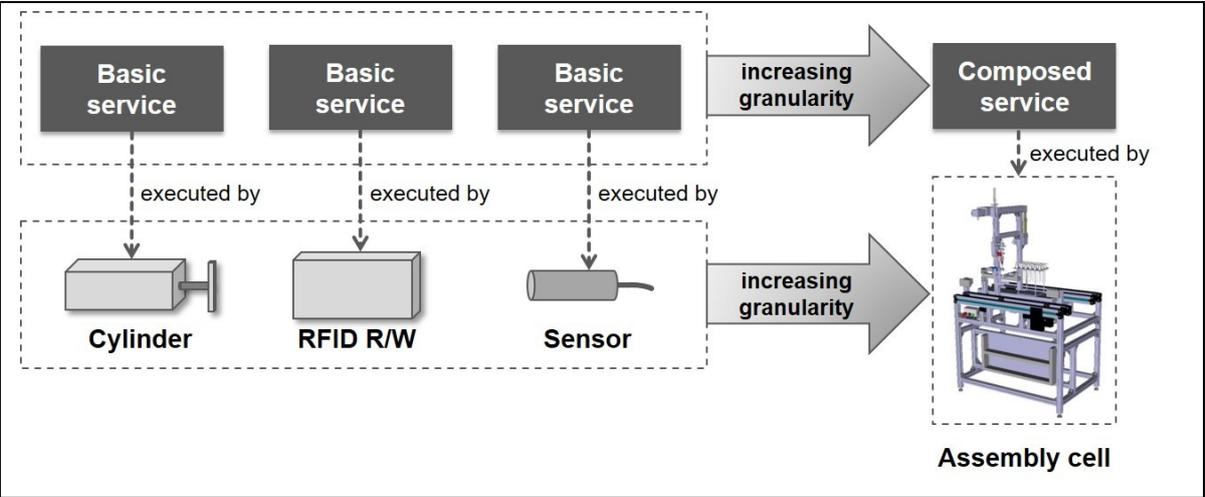


Figure 6-5: Basic and composed equipment services

Composed services don't interact with the production equipment directly but via basic services of the field devices the unit comprises. This necessitates a control strategy determining how several services work together to create added value in form of a more complex service [Când09a]. Thereby, a general question is if equipment services should comprise knowledge about the process they are used within. This would be required when new functionality is built according to the choreography composition principle (see Chapter 3.4.2). In this case, the control strategy is distributed among the involved services. This

implies that each time the context changes within which the service is used, the logic of the services needs to be adapted as well.

Obviously, this contradicts the desire for universal services that are defined and that are independently applicable on the current application. Equipment services are hardware-oriented so that they should provide the functionality of the production equipment as independent as possible of the application they are used for. Thus, high-level functionality needs to be composed in a central way via service orchestration in order to empower the reusability of the single services. This implies that only the composed service contains the knowledge about the scope of the functionality it represents. The internal logic of a composed service implements the orchestration logic for calling the services that are needed to execute the single actions in the right point of time (see Figure 6-6).

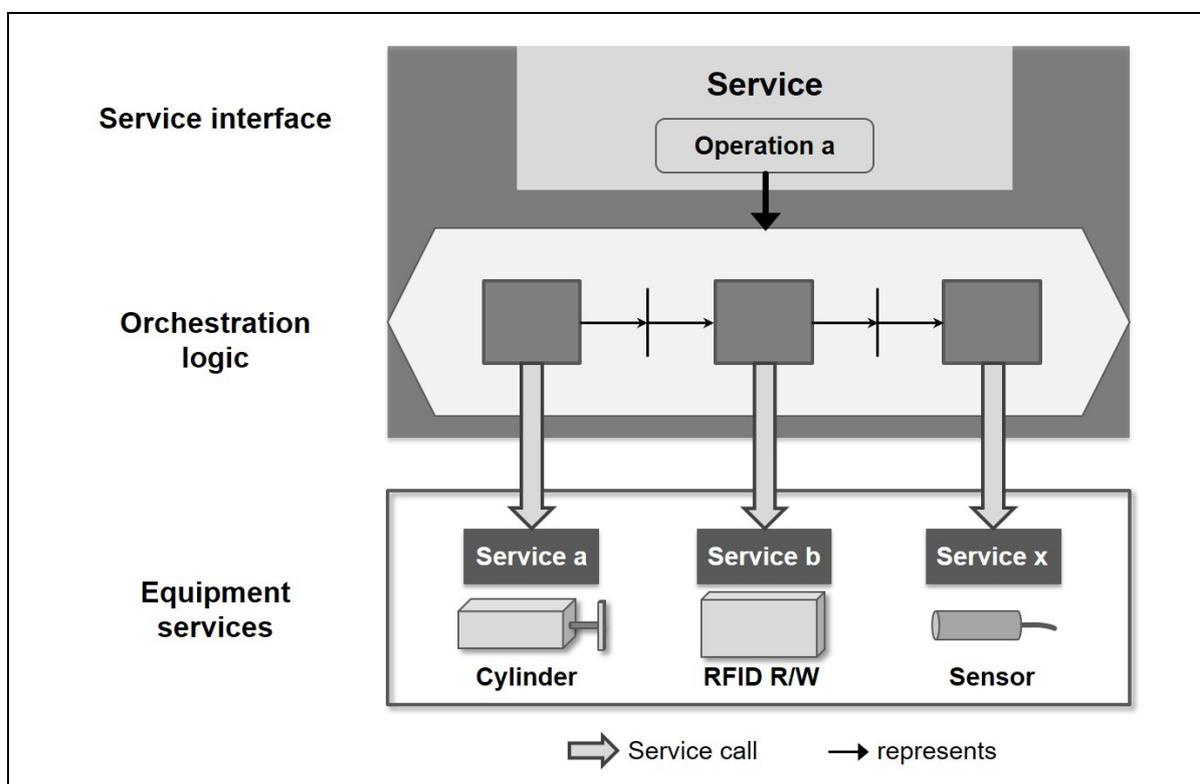


Figure 6-6: Generation of high-level services via service orchestration

6.2.2 Control Services

Apart from the equipment layer a second superordinate service layer for building the control procedures is defined as the *control layer*. Services on this layer—named *control services*—generate control procedures for executing the desired production process. In contrast to an equipment service, a control service is rather process-oriented than hardware-related. However, control services can also be assigned to hardware components, like a production line or cell, which is controlled by the very fact.

The main task of a control service is to implement a process logic which calls equipment services in the desired sequence and with the right parameters for executing the technical process. This internal logic of a control service is realized as a service orchestration of equipment services similar to composed services (see Figure 6-6). Therefore, the control logic establishes bindings to the equipment services to make the orchestration logic executable. This implies a mapping between the two service layers where suitable equipment services fulfilling the demanded functionality of the control procedure are allocated to the respective process steps (see Chapter 3.4.1).

Obviously, there is a relation between the granularity of equipment services and the orchestration logic of control services. The lower the granularity of equipment services is the more detailed the control logic of the control service needs to be (see Figure 6-7). A coarser control logic can be sufficient, if equipment services with a high granularity have been generated. The question about the ideal granularity level is indeed complex and the right answer depends on the specific case. However, in the ideal case both granularity levels fit with a minimal information gap so that the mapping between services on both layers can happen as seamlessly as possible.

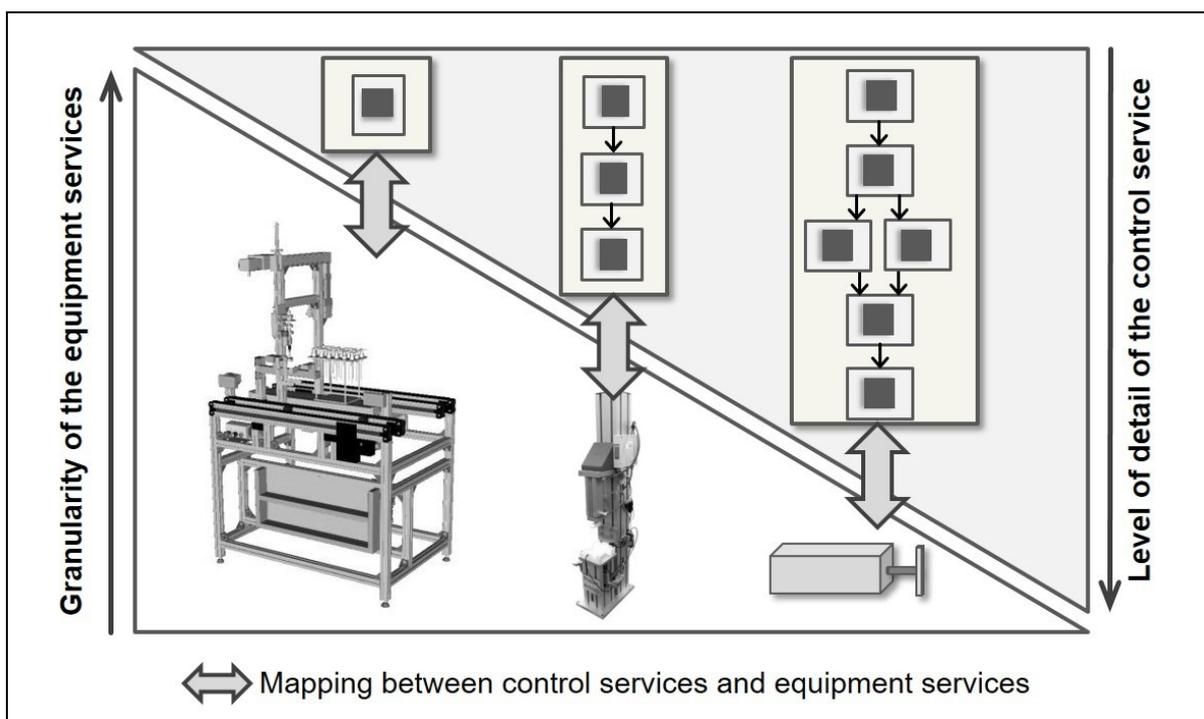


Figure 6-7: Granularity of the logic of equipment services and control services

Besides calling equipment services, control services are responsible for all other control functions to ensure a stable production process. This includes particularly the configuration of the production process according to various product variants, managing the material flow, startup and shutdown routines, supervisory control tasks to synchronize all production units, safety routines, and SCADA and HMI functions. Implementing all of these functions into one

service would result in a complex and rather monolithic control program similar to today's PLC programs excluding the device functions (see Chapter 2.3.4).

For enabling a better scalability, more flexible control logics, and a higher reusability of services also on the control layer, it makes sense to split the control procedure into several services. Besides the well-known advantages of modular software design (see Chapter 4.1.1), outsourced control logics can be designed on a higher abstraction level. Therefore, concepts for a late or even dynamic service binding are applied (see Chapter 3.4.1). First, the bindings to equipment services within the separated service are determined in an abstract way. Thereby, an abstract service is defined in terms of specifying the requirement on the functionality of the service (see Chapter 6.3.3). Later, the main control logic takes care of establishing the service bindings to suitable equipment services. As an instance, a separated control service "sub control" requires the use of a drilling service but leaves open which hardware component executes the service (see Figure 6-8). The main control service calls the "sub control" service and determines that the abstract service is realized by the currently available equipment service "drilling_3M".

This brings the advantage that control services can be defined independently of the concrete equipment services that will be available during runtime. Furthermore, the binding can be decided globally in the main control service or even dynamically based on current circumstances. This procedure of assigning abstract services within a control service is defined as *abstract service allocation*. The prerequisite for exploiting its full potential is a standardized procedure to determine service names and their types as well as a service library for collecting possible service templates (see Chapter 6.3.3).

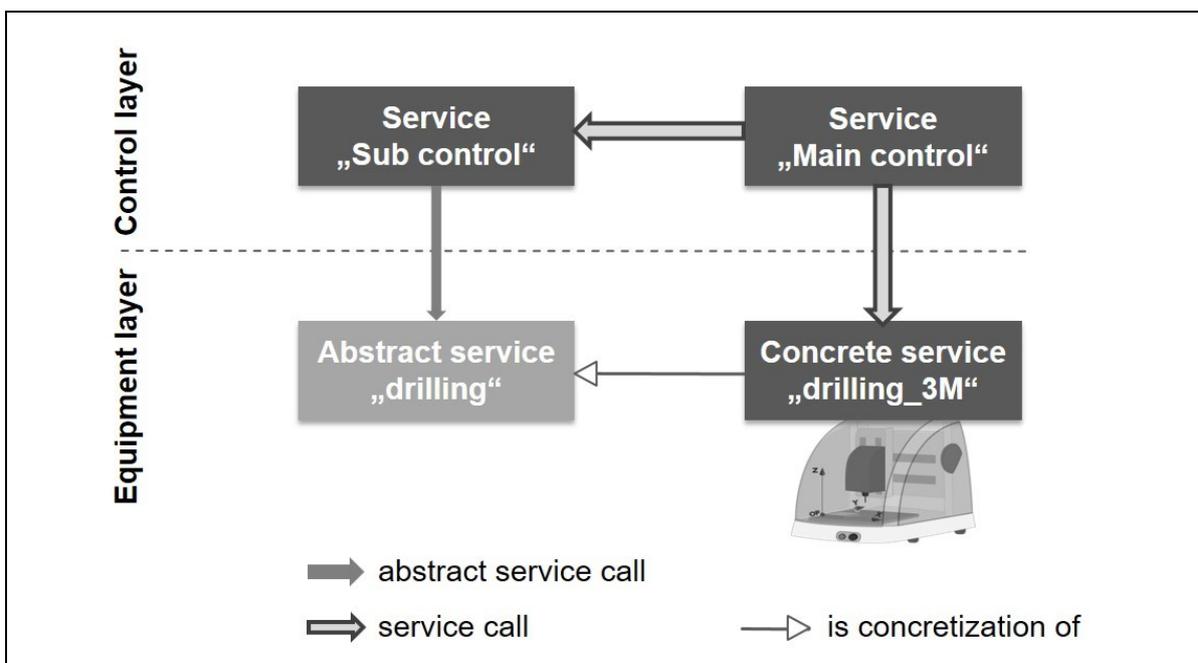


Figure 6-8: Late binding through abstract service allocation

Regarding the automation pyramid, there exist numerous different functions implemented on the control layer (see Chapter 2.2) that can be structured according to certain service sub-categories. For the scope of this thesis three service categories of design control procedures are defined:

- **Process services:** For realizing a service-oriented control procedure, the control layer has to contain at least one mandatory process service. It directly implements the above-mentioned control functions or it acts as a master for managing all underlying control functions that are outsourced to other services. Furthermore, it determines all bindings of its own service orchestration and those who have been specified via abstract service allocation in other control services it uses. Subordinate process services can also be generated to split complex control logics into several services.
- **Product services:** The demand on flexible production systems being able to produce diverse product variants or even different product types with similar production processes is continuously rising. Control functions that depend on the respective product type can be outsourced to product services. The process service calls the respective product service according to the current production order.
- **Supporting services:** Any pure software function that is needed for the overall control procedure and that is easy separable from the main control logic can be outsourced to supporting services.

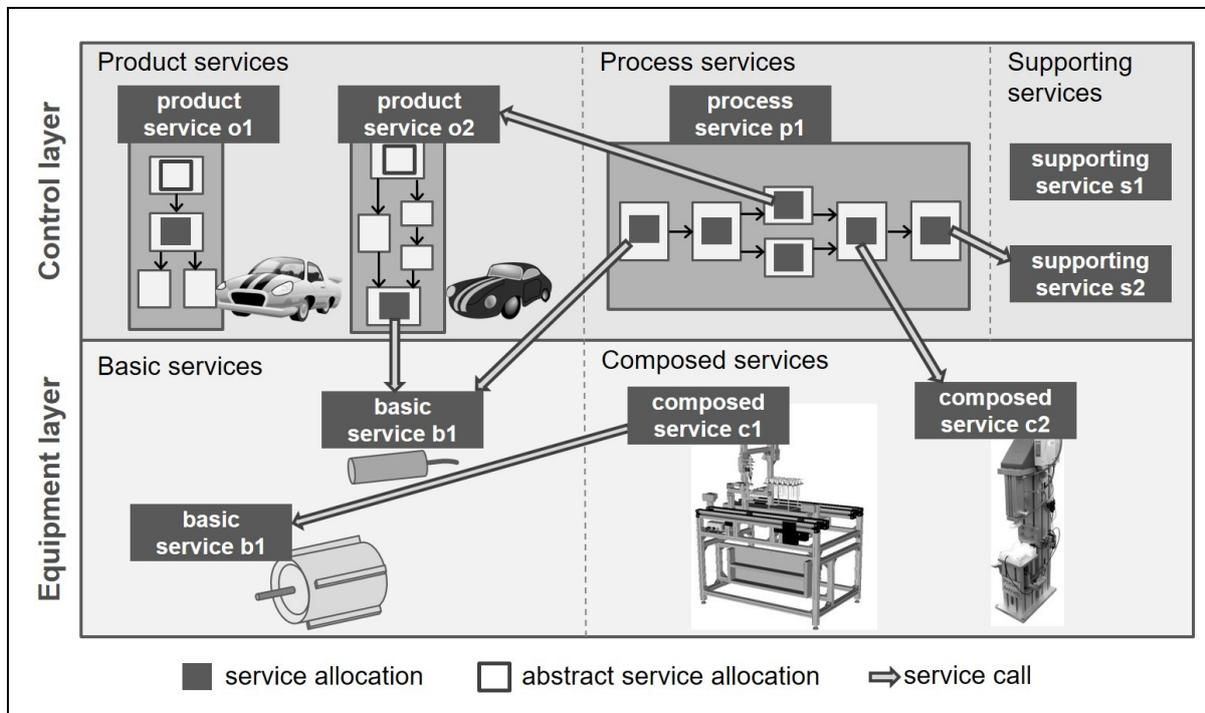


Figure 6-9: Reference architecture for service-oriented manufacturing control

The above defined control service categories can be complemented with others of course, if a reasonable distinction of characteristics is necessary. Altogether, the service categories of the control layer and the before mentioned services of the equipment layer form together a reference architecture for service-oriented manufacturing control applications (see Figure 6-9).

6.3 Specification of Equipment Services

The fundament of a SOA-AT system constitutes the equipment layer, which comprises the building blocks of the services on the control layer. Hence, the specification of equipment services is a crucial task influencing heavily the later design of the control application. For this reason, the individual aspects to specify an equipment service are investigated and guidelines are presented to ensure an efficient control engineering.

6.3.1 Service Description

The service description comprises the service name and the specification of the service operations. Furthermore, a service description can comprise several attributes to express certain characteristics of the service. Generally, any additional information that is important for potential service users should be captured within such attributes. Attributes can be used to specify the functionality of the service or to describe details regarding the communication technology the service interface is using.

How the service description is specified influences heavily the usability of a service and how seamless the mapping processes between requirements and matching services can be executed (see Chapter 3.4.2). Within a service-oriented control architecture this particularly concerns basic services, since they act as the atoms within the service architecture that are responsible for controlling the production equipment.

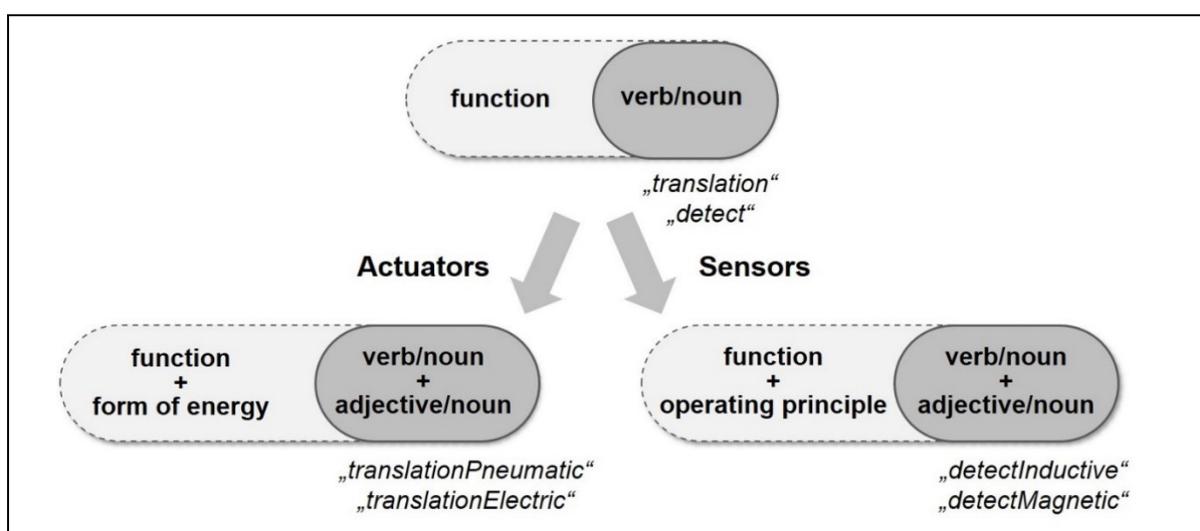


Figure 6-10: Naming scheme for basic services

Based on the recommendation from Erl (see Chapter 3.4.2) a naming scheme is defined to enable meaningful and comprehensible service names for basic services (see Figure 6-10) [Erl05]. The first building block is the verb which stands for the encapsulated technological functionality of the field device, for example, the function of a sensor as “detect.” In some cases, the function can be described more exact by a noun, for example, a translational motion of a cylinder represented by “translation” instead of the much more general “move.” The second part of the name further specifies the function by an adjective or noun. In case of the basic service of an actuator, the information about the form of energy for executing the function is added by an adjective or noun, for example, “translation” would be further detailed to “translationPneumatic” for a pneumatic cylinder or “translationElectric” for an electrically operated cylinder. For sensors, it is essential to know how the sensory function is realized according to the operating principle. Thus, the respective information is added, as instance “detect” is extended to “detectInductive” for an inductive sensor or “detectMagnetic” for a reed switch. The scope of composed equipment services is more specific and depends on the particular application domain so that their naming scheme is more general. Similar to basic services the first part consists of a verb or noun representing the function that can be combined with an adjective or noun for additional information.

Besides the service name another important item of the service description are the operations. Since they represent the access points to the individual functions encapsulated by the service, their names can be derived from the first part of the service name. As instance, for the service “translationPneumatic” the operations “translationDirection1” and “translationDirection2” are defined to move the cylinder in each possible direction. For devices of high complexity, such as servo controllers, additional operations can be added to make the complete functionality available. To exchange information operations can comprise input and output variables. For example, the service “translationPneumatic” could also have just one operation “translation(in: direction)” with an input variable specifying the direction (see Figure 6-11 left).

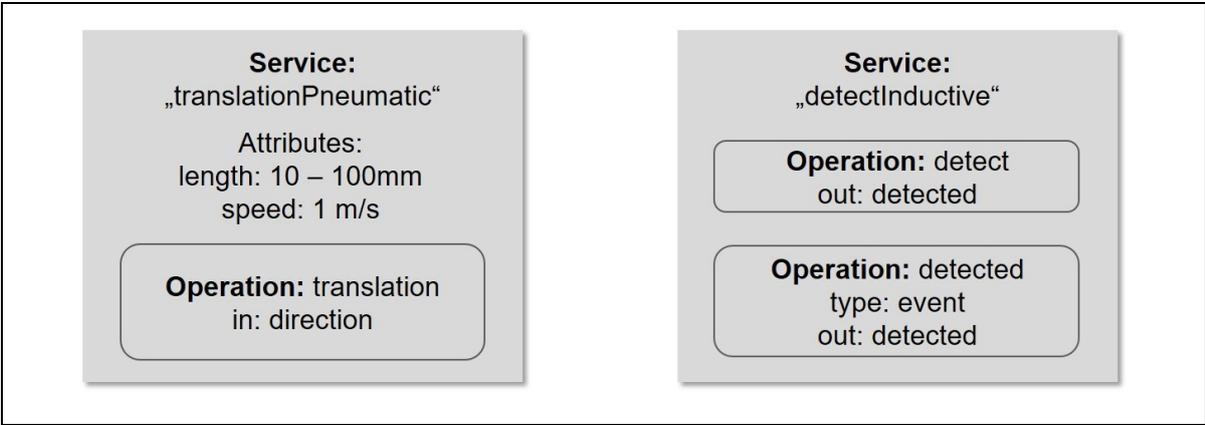


Figure 6-11: Examples of services descriptions with service operations and attributes

The usual working principle of a service works according to a request-response pattern. When an operation is called, its task is executed once and the result is handed over as soon as the task is finished. Besides this, the introduction of another working principle is useful known as *eventing* (see Chapter 3.4.5). Calling operations that work according to the eventing principle trigger the subscription of a certain event. Each time the event happens the operation gives feedback as long as the event is subscribed. In this way, a steady polling of sensor values can be avoided when a certain event within the production process is of interest. This applies for example when a temperature limit causing an alarm is observed or when the material flow is triggered by the presence of an object at a certain position. Eventing operations are characterized by the additional attribute “type: event” (see Figure 6-11 right).

6.3.2 Design Aspects of Equipment Services

To specify an equipment service, several aspects have to be considered during the design. Generally, a service is specified by its external and internal properties in terms of the service description and control logic (see Chapter 3.4.2). Besides these items, additional aspects have to be taken into account to design a service for manufacturing control.

In contrast to basic services, composed services make use of external services (see Chapter 6.2). Thus, their functionality is composed of the functionality of the external services that are called within their service orchestration. Since the availability of the respective services is essential to execute the service orchestration, the dependencies between an orchestrated service and the external services it uses have to be depicted. Moreover, the relationships between services and production equipment need to be taken into account (see Chapter 6.2). Similar to services, the hierarchical dependencies between hardware modules on different granularity levels have to be considered. Superior hardware components, for example production cells or lines, are composed of smaller hardware components like field devices.

These design aspects have to be determined for each equipment service of the control system. To permit a better handling of complexity the aspects can be divided into three separated views on the production system for which the control system is developed (see Figure 6-12):

- **Functional view:** The services of the control system are designed as components that represent the individual functions of the production system. For each service a number of operations and attributes are defined regarding a service description. Relationships between services indicate when one service uses another service for its execution. In this way, a functional modularization of the production system in several granularity levels is obtained.

- **Dynamic view:** The behavior of each service is specified as control logic. Therefore, the operations of the service components defined in the functional view are linked to the respective control logics that are executed as soon as the operation is triggered.
- **Hardware view:** The physical modularization of the production system is depicted as hardware components on several granularity levels that are related via containment relationships. Links to the functional view illustrate which hardware components execute which services.

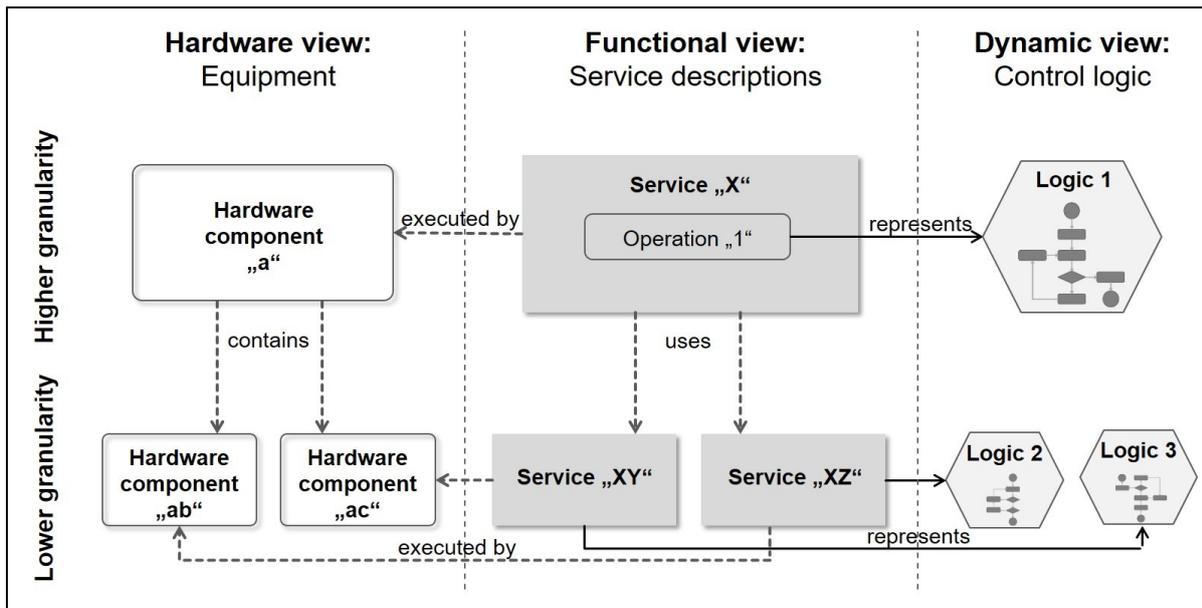


Figure 6-12: Design aspects of an equipment service

6.3.3 Library Concepts

A library collects templates of any kind of planning objects that can be used for different applications. Thus, they constitute a tool to enhance reusability and standardization of engineering results. By an efficient use of a service library, the efforts for designing control services can be drastically decreased. Previously defined equipment services can be collected and stored as *service templates*, which can then be reused for other applications. Furthermore, these templates can be systematically arranged within several levels of detail so that the engineer is optimally supported on finding the right service.

The concept of a service library enables an object-oriented design approach (see Chapter 4.1.1). Thereby, the elements of the library represent service templates which act as classes. For a specific design task, a suitable service template is chosen from the library and instantiated as a respective planning object (see Figure 6-13). Services are connected to the service templates from which they are instantiated via the *is type of* relationship. According to the defined design aspects of a service, the elements of the service library represent primarily the service descriptions. However, the service templates can also comprise a control logic and information about the type of the related hardware component.

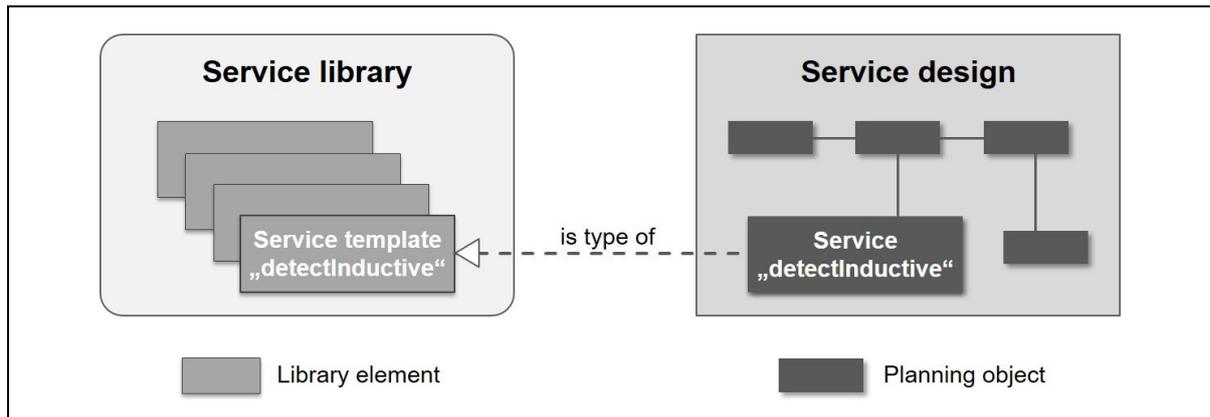


Figure 6-13: Service design by using a service library

The definition of service types and the application of the standardized naming scheme (see Chapter 6.2 & 6.3.1) support a consistent structure of the library that comprises several levels of detail. Generally, each element takes over the properties of an element on a higher level of the library and adds further details so that the concretization degree increases on lower library levels (see Figure 6-14). In object-orientation this task is known as inheritance (see Chapter 4.1.1) which is applied here in a weakened way, since the adopted properties can also be modified, for example the name, inputs, and outputs of the service operations. This kind of concretization of one element to another is indicated by the *is concretization of* relationship. All elements having this relationship to the same superordinate element belong to the same category. The elements on the highest level of the library represent the service categories of the reference architecture. The service library particularly makes sense for basic services, since the demand on their degree of reusability and standardization is explicitly high. Thus, the structure of the service library for basic services is defined in greater detail in several hierarchical levels (see Figure 6-14).

The mother element of all basic services is the service template “basic service”, which has the two child elements “sensor service” and “actuator service.” Below these the service templates are distinguished according to the technological functionality they represent. This is indicated by the first part of the full service name regarding the naming scheme like “detect” or “translation” (see Chapter 6.3.1). Until this level all library elements are indicated by an additional attribute “type: abstract” because they serve purely for structuring purposes and cannot be instantiated. Below this level the service templates are fully specified by adding the respective form of energy or the operating principle and can be instantiated as planning object for a certain design task. Another level of detail can be generated when services need to be adapted to certain devices. For example, “detect” has among others the child elements “detectInductive” which has again “detectInductive_P+F” as child with special features.

Similar library concepts are already common for hardware components that are listed in *device catalogs*. This is particularly applied by device manufacturers or device vendors to

present which commercial parts they offer to potential customers. The customer can then choose from this catalog and buys a respective instance of the catalog item. Bigger manufacturing companies often have internal device catalogs to standardize the equipment they use for their production lines. Today, there exist already initiatives for standardized cross-vendor equipment catalogs like eCI@ss (see 7.1.2). During the design of services for manufacturing control such device catalogs are useful to determine hardware components that execute the respective services.

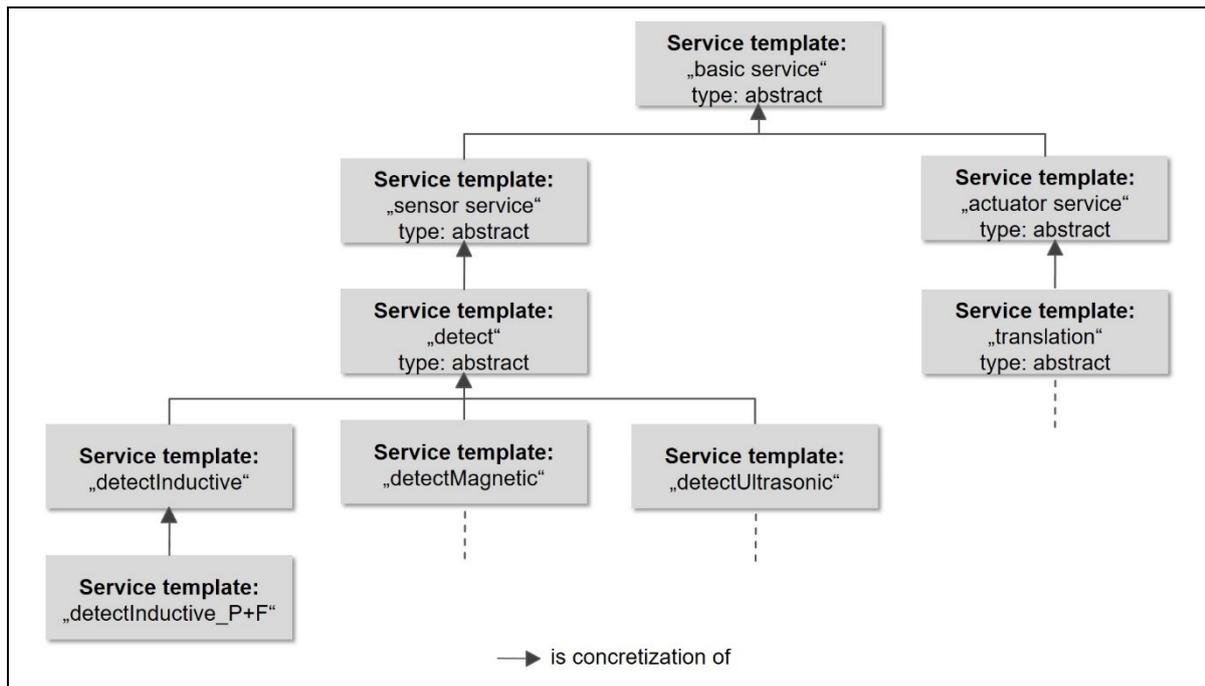


Figure 6-14: Service hierarchy for structuring basic services

6.3.4 Abstract and Concrete Specification of Services

Like for all planning tasks a stepwise proceeding with several concretization steps supports the engineer's way of thinking and is beneficial for the comprehensibility and reusability of planning results. This applies particularly during the design of the control services wherefore suitable equipment services need to be determined. For the specification of the required equipment services, two major concretization steps are defined by using OO principles (see Figure 6-15):

- **Abstract Specification:** First, the service is specified independently on the concrete device that is used later. A suitable service template is picked from the service library and instantiated as planning object. If an appropriate service template is not available, a new one is created. Besides this, a hardware component is generally determined in terms of a device category like "inductive sensor" or "pneumatic cylinder." Moreover, a general control logic is designed with instructions how the functionality of the service has to be implemented.

- Concrete Specification:** The results from the abstract design are detailed regarding the concrete device that will be used. Therefore, a suitable device will be chosen from a device catalog of a certain vendor and the control logic is worked out accordingly. For instance, the inductive sensor from Pepperl+Fuchs with the type identifier “NBB0” will be picked. If the service description doesn’t cover all functions of the device, the service description is adapted to a device-specific one like “detectInductive_P+F.”

Generally, adapting the service description regarding device-specific details should be avoided and just be applied with caution. Preferable are vendor-independent and uniform service descriptions to make the control application more independent of the respective hardware. Besides a higher reusability of the services, this brings the advantage that the hardware can be exchanged more flexibly without changing the service orchestration of the control services. However, field device manufacturers usually want to obtain a competitive advantage by providing products with unique selling propositions to stand out against other products from competing companies. This fact makes it often necessary to deviate from a vendor-independent standardization of service descriptions to cover special features of the device.

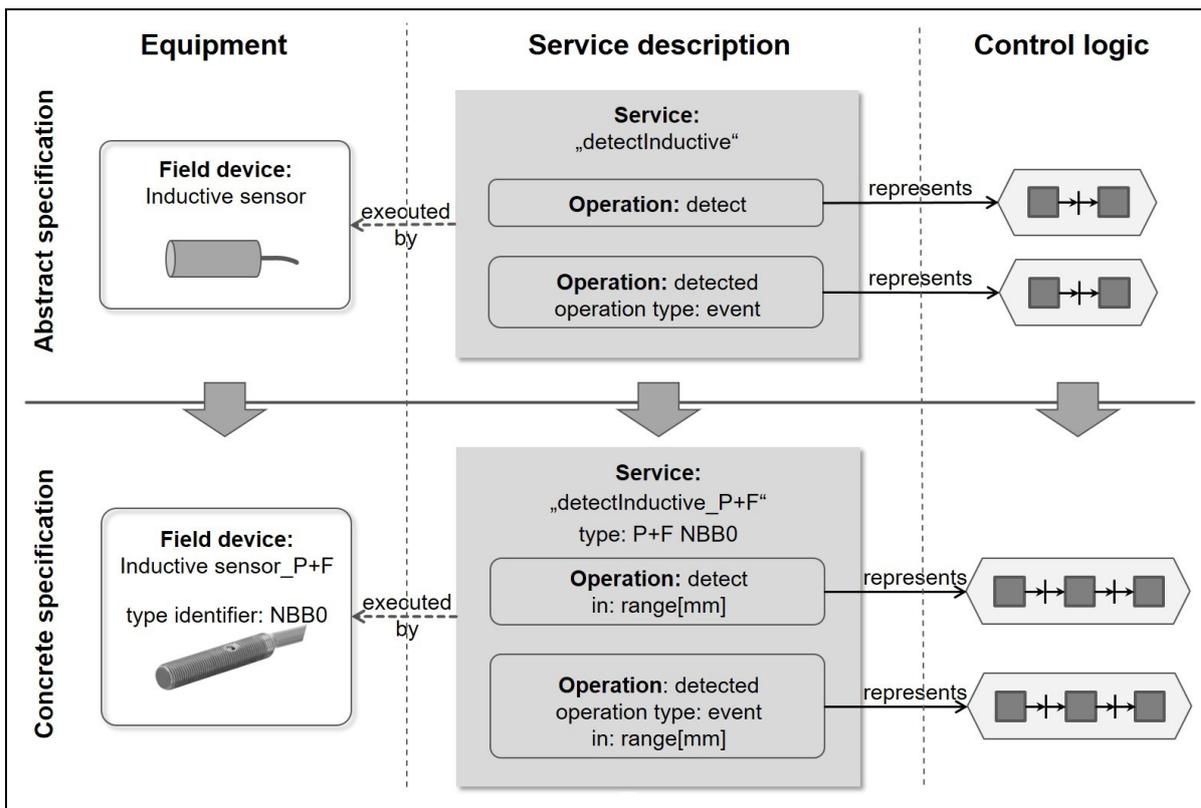


Figure 6-15: Abstract and concrete specification of an equipment service

6.4 Development of Control Services

After discussing how equipment services are specified, the development of control services is investigated next. First, some guidelines for their specification are defined and afterwards, a process-oriented development method is presented. Therefore, principles of the process-oriented design strategy for SOA systems (see Chapter 3.4.1) are applied to define an integrated design method to determine control services and their required equipment services.

6.4.1 Specification of Control Services

The conditions for the specification of control services can be decided more loosely compared to equipment services because they are more independent of the production equipment and the demands on their reusability is lower. Moreover, they highly depend on the respective application scenario and characteristics of the production process which leads to a more individual design. Nevertheless, some guidelines for their design are helpful for a well-structured engineering so that the same design aspects of equipment services (see Chapter 6.3) are examined for control services.

Control services also have to be specified via a service description. A similar naming scheme as for equipment services (see Chapter 6.3.1) can be applied that starts with a verb to describe the main action and that gets combined with adjectives or nouns to add further information, for example “assembleCarDoor.” The operations of a control service trigger the execution of the control tasks. Which operations are needed depends heavily on the type of control service and its individual properties (see Chapter 6.2.2). Usually, the main process service is continuously executed as long as the respective production process is running so that the operations reflect the control mode changes like “startAutomaticMode”, “startStepMode”, and “stop.” Product services and supporting services are called by the process service just in certain events so that their control logic is usually executed one time. Hence, their operations trigger the single execution of a certain control logic, for example “calculateSettingsCarX.”

The design aspects of equipment services that are displayed within the three defined views (see Chapter 6.3.2) can also be applied for control services. Generally, product and supporting services do not directly relate to hardware components so that the hardware view can be neglected. In contrast to this, the hardware view is applied for process services. They are allocated to the hardware components that execute the production process which is implemented by them. Due to this relation to hardware it can make sense to develop process services according to the two concretization steps (see Chapter 6.3.4). This is particularly beneficial to enable a parallel design when the technical details of the hardware are not determined yet, but the design of the control process can already be started. Also adding control services to the service library is recommended for any service that is

potentially reusable. In the ideal case, each control service can be directly reused within other applications. However, the adaption to a certain use case is probably needed for many services so that the library items are used as templates that are modified according to the current use case.

6.4.2 Process-oriented Development of Control Procedures

A service-oriented control procedure is implemented in form of multiple services on the control layer. This includes the design of the individual control services and their service orchestrations which specify the control logic and the bindings to the equipment services. The prerequisite is that the required equipment services have been determined based on the requirements on the desired production process. Thus, an essential question for the development of control services is how the required equipment services can be directly derived from these requirements. By applying principles of the process-oriented design strategy (see Chapter 3.4.1) a seamless design method including the mapping between the control layer and the equipment layer can be achieved. The design process is divided into seven steps that can be partly executed in parallel (see Figure 6-16).

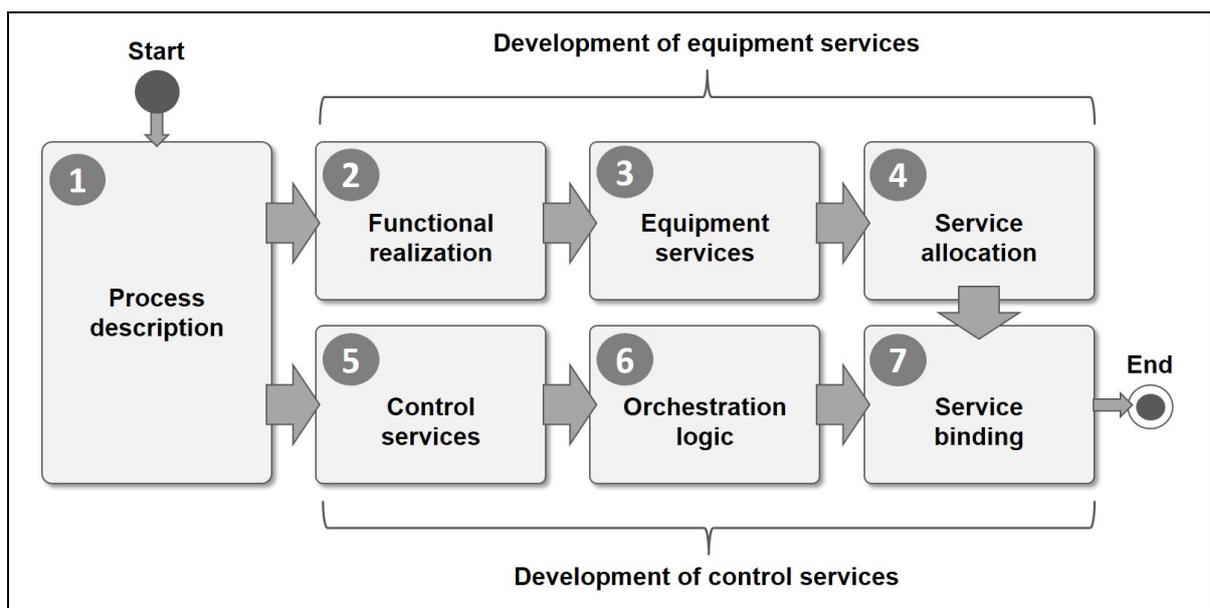


Figure 6-16: Overview of the steps for the process-oriented design

The determination of the required equipment services is covered by step 1-4 (see Figure 6-17):

- 1. Process description:** First, the requirements on the control procedure are formulated that act as input for the further design steps. Since the objective of the control system is the execution of a certain production process, the basic requirement can be formulated as a description of it. Therefore, the individual steps and their time sequence need to be specified that are needed to produce a certain product. This

should be done by focusing on “**What** needs to be executed?” without already including technical solutions. The process description can comprise several detailing levels. Process steps are decomposed in more fine-grained process steps until the lowest level of *elementary process steps* is reached. For example, a process step on the lowest detail level is defined as “Stopping bin in front of lidding device.” (see Figure 6-18).

2. Functional realization: The challenge to derive the required equipment services from the process description is solved by this intermediate design step. After the process description, has been elaborated, the functional realization of each elementary process step is determined. Thus, the designer needs to answer “**Which** functions are needed to execute the process step?” in terms of individual functions that are still independent of a technical realization. For example, the process step “Stopping bin in front of lidding device” requires the functions “detect” for making the production system aware that the bin has reached the position and “stop” for stopping the motion of the bin (see Figure 6-18). After that, functions that will be executed by the same hardware component are bundled to a group. For this *function bundling* the functions of all process steps need to be considered because different process steps can be executed by the same production equipment.

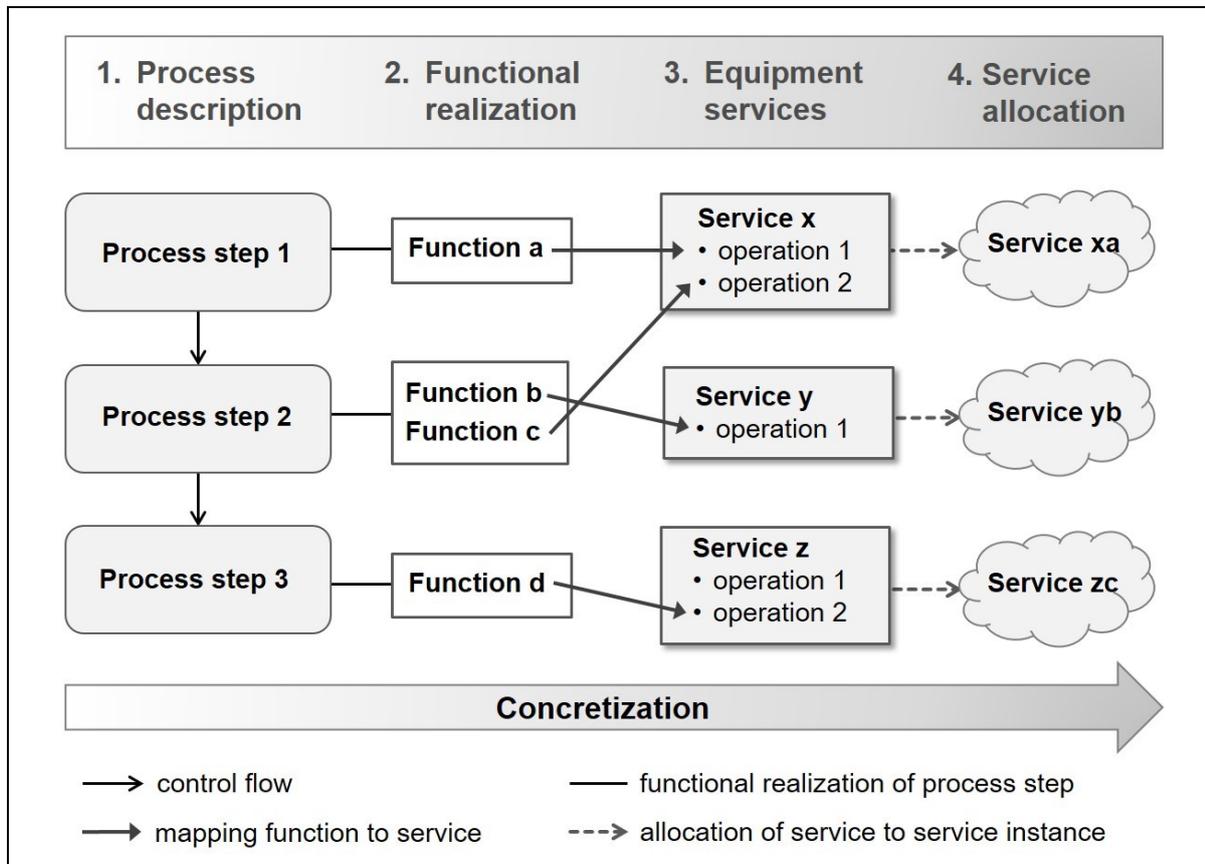


Figure 6-17: Process-oriented development of equipment services

3. **Equipment services:** For each group of related functions and each standalone function a technological realization is defined as an equipment service. Therefore, the service is specified regarding the question “**How** is the function technologically realized?”. This task comprises two sub steps according to the abstract and concrete specification of a service (see Chapter 6.3.4). Furthermore, the operations of the service are selected to execute the respective functions. For example, the function “detect” will be realized by sensing the bin via an induction loop so that the service “detectInductive” is chosen with its operation “detect.” Ideally, the services are already defined and can be chosen from the service library (see Chapter 6.3.3). Otherwise, the services need to be defined first according to the design guidelines and then be included into the library.
4. **Service allocation:** Until the end of step 3 the services are defined as pure planning objects so that in a last step an implemented service instance needs to be allocated. This implies a unique address or identifier to access the respective service and refers to the question “**Where** is the service executed?”. For example, the service “detect.inductive” is implemented as a is Web Service and accessible under a certain IP address.

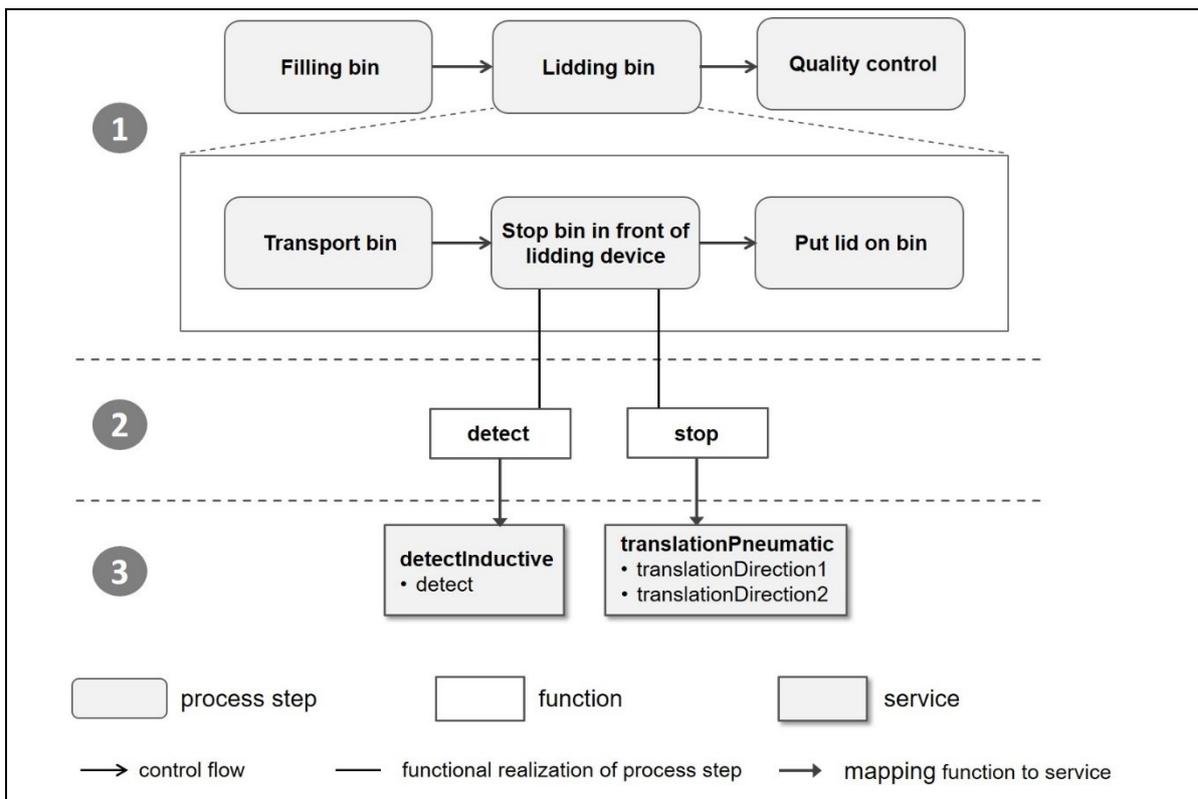


Figure 6-18: Example for the determination of an equipment service

In parallel or after the specification of the required equipment services (see Figure 6-16), the control services and their orchestration logic are specified within the steps 5 to 7. Also

here the process description provides a suitable starting point followed by three design steps (Figure 6-19):

- 5. **Control services:** The individual process, product, and supporting services are specified for realizing a desired control strategy. Thereby, requirements regarding product and process variants, overall complexity, and additional automation functions have to be considered. If the process description is already complex, this indicates that a separation of the control logic in several services on control layer should be introduced.
- 6. **Orchestration logic:** For each control service an orchestration logic is specified to implement its behavior. Enabled by the higher abstraction degree of service orchestrations in contrast to PLC code, a rough control logic can already be derived from the process description. Therefore, the individual process steps from the process description that belong to the particular control service are transferred to the orchestration logic. The required equipment services to be called within the logic can be derived from step 3. In this step the equipment services have already been determined for the piece of process logic which is realized by the respective control service. The obtained rough control logic then needs to be detailed and adapted according to the requirements on the control service.

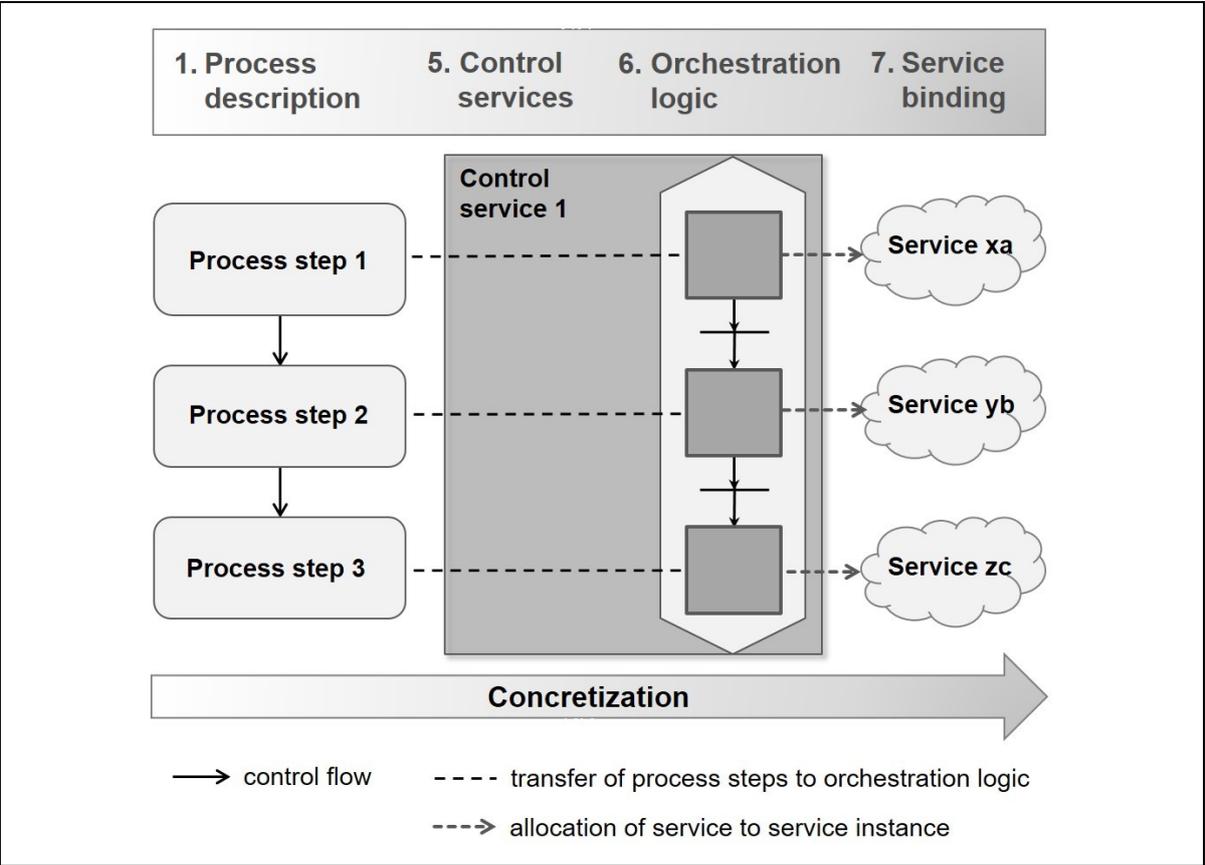


Figure 6-19: Process-oriented development of control services

- 7. Service binding:** To make the control services executable the service bindings between the service orchestrations and the equipment services need to be established. Therefore, information about the accessibility of the service is required which is delivered by step 4 where the allocation of the service instances took place.

How straightforward and seamless the flow from the process description to the determination of suitable services can be executed, depends heavily on guidelines for the definition of the functions and of the service descriptions (see Chapter 7.1).

6.5 Engineering Process

The prerequisite to efficiently develop service-oriented control procedures is a consistent and systematic engineering process. Therefore, a control engineering process is defined with distinct planning phases that are based on the general methods for the design of equipment and control services (see Chapter 6.3 & Chapter 6.4). Furthermore, this control engineering process is included within the overall production engineering process. Thereby, effects on the planning phases are analyzed and opportunities for improvements of existing drawbacks are pointed out.

6.5.1 Control Engineering Process

To support the fluent development of service-oriented control procedures an engineering process according to the Waterfall model (see Chapter 4.1.2) is defined. Four phases with dedicated planning results describe the development process for the process-oriented engineering of service-oriented control procedures, abbreviated as PES COP (see Figure 6-20):

- **Analysis:** During the analysis phase the preliminary work is executed for providing the required input to design the control system. The results constitute the process description (step 1 of Chapter 6.4.2) and the functional realization of the elementary process steps (step 2 of Chapter 6.4.2). Although the planning should be initially as independent of hardware properties as possible, several requirements on the production equipment can exist and thus, should be also considered during the further planning steps. For example, a rough plant design can already assign certain production cells to process steps.
- **Design:** The design phase comprises the definition of the services of the control system (step 3, 5 & 6 of Chapter 6.4.2) and the determination of the required production equipment. It is divided in two phases according to the principle of abstract and concrete specification of services (see Chapter 6.3.4):
 - **Abstract Design:** First, the abstract specification of services is executed based on the results from the analysis phase. In this stage, the planning objects represent abstract services and components that are hardware-independent and act as requirements for the following concretization. According to the three design

aspects of a service, the planning results are represented in three parts. The service structure comprises the single services as components, which contain their service descriptions. Another static view is given by the equipment structure, which depicts the modularization of the hardware components. The last part illustrates the dynamic behavior of each service as control logics. Relationships of planning objects are represented via links between the respective objects.

- **Concrete Design:** After the abstract design is completed, the service design is detailed according to the concrete specification of the services. Thereby, the services are specified by committing on concrete devices of the production equipment. The service structure, equipment structure, and control logic are modified in case if the chosen hardware requires any changes or detailing of the results from the abstract design.

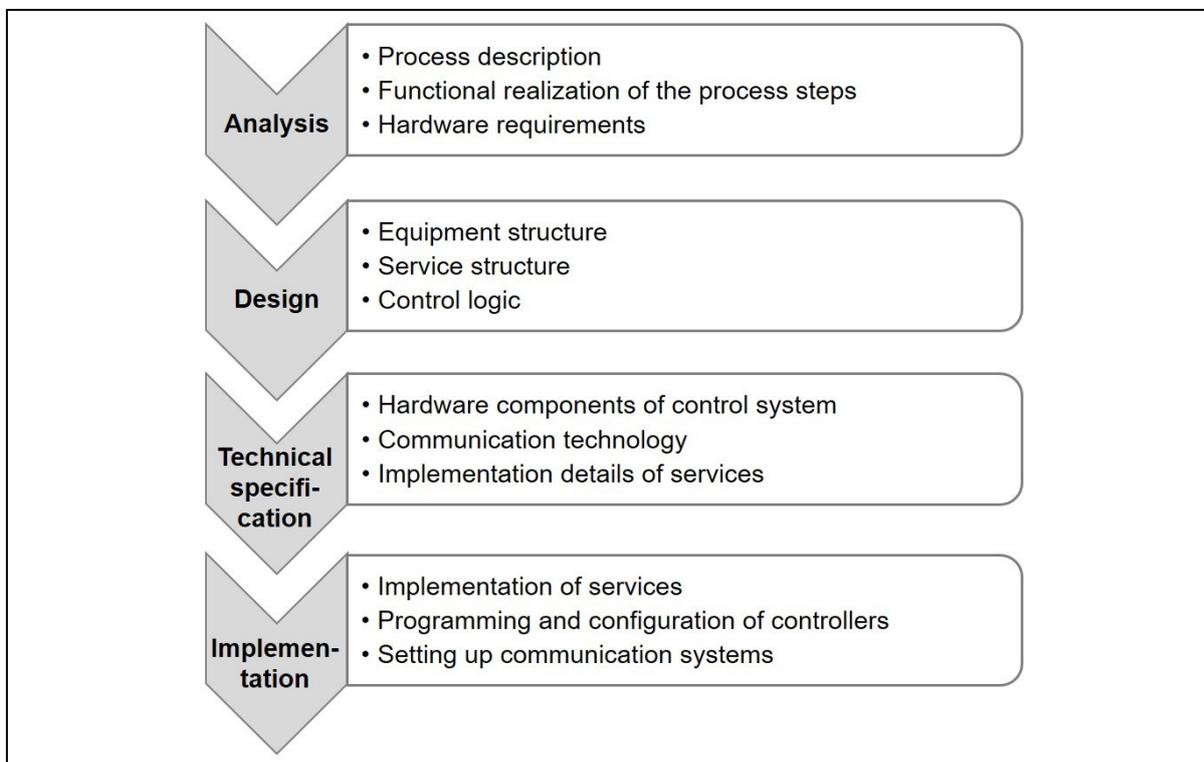


Figure 6-20: Phases and results of the PES COP process

- **Technical Specification:** In terms of distributed systems the design phase defines the software architecture, whereas the technical specification phase determines the system architecture of the control system (see Chapter 3.2). This happens by deciding on all technical details that are needed to implement the services with respect to the non-functional requirements on the control system. This includes among other things the selection and configuration of the hardware components where the services are running on, the communication technology, and an implementation concept of the services.

- **Implementation:** Finally, the planning results are transferred into realization. The individual services are implemented and the controllers and communication systems are set up. After that, the service bindings can be established (step 4 & 7 of Chapter 6.4.2) so that the control procedure is executable.

6.5.2 Placement within the Production Engineering Process

So far, the development of service-oriented control procedures has been regarded autonomously. However, control engineering is typically not executed standalone but in association with the planning of a production system. Hence, the applicability of the previously defined PES COP process can be granted by embedding it into the production engineering process (see Chapter 2.3.2). Thereby, the planning phases of today's common planning process are adapted as much as necessary to integrate the new concepts provided by PES COP as optimal as possible (see Figure 6-21). Furthermore, potential effects and opportunities for improvement regarding the planning phases and other engineering disciplines are analyzed.

The starting point to link the new engineering method to today's common production planning process constitutes the rough planning phase. Process plans and the formulation of requirements on the production equipment are already common deliverables based on the input from product design and factory layout. Hence, these process plans are further developed to the more detailed process description. Additionally, the functional realization is determined and possibly existing hardware requirements of each process step are reflected.

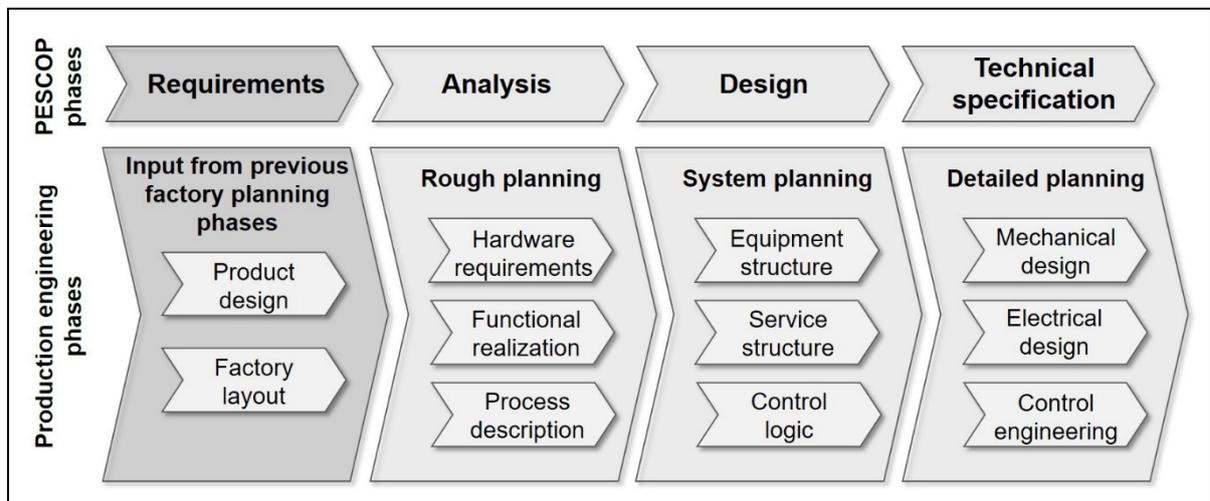


Figure 6-21: Adapted planning phases of the production engineering process

Between the rough planning and detailed planning an additional planning phase is introduced called *system planning* which covers the tasks of the PES COP design phase. Inspired by the principles of systems engineering, it comprises a holistic planning of the production system based on existing requirements (see Chapter 4.3.5). Therefore, the

planning results are divided into the three views on the production system according to the planning aspects of services (see Chapter 6.3.2 & 6.5.1): the hardware view with the equipment structure, the functional view with the service structure, and the dynamic view with the control logics. These planning results are worked out in two concretization steps following the sub-phases of the PES COP design phase. During each concretization step the three views can be developed in parallel whereby dependencies between planning objects imply planning sequences decided on a case-by-case basis.

The last planning task constitutes the detailed planning phase where all technical details of the individual engineering disciplines are specified based on the results from system planning. For control engineering this signifies the tasks of the PES COP technical specification phase. The preliminary work of the system planning provides the prerequisite for the engineering disciplines to start on deeper detail level in contrast to today's conventional planning procedure. Since the specific details of the different engineering disciplines are mainly independent of each other, the disciplines can be executed in parallel by experts of the respective domain. Nevertheless, additional dependencies can come up and need to be reflected accordingly.

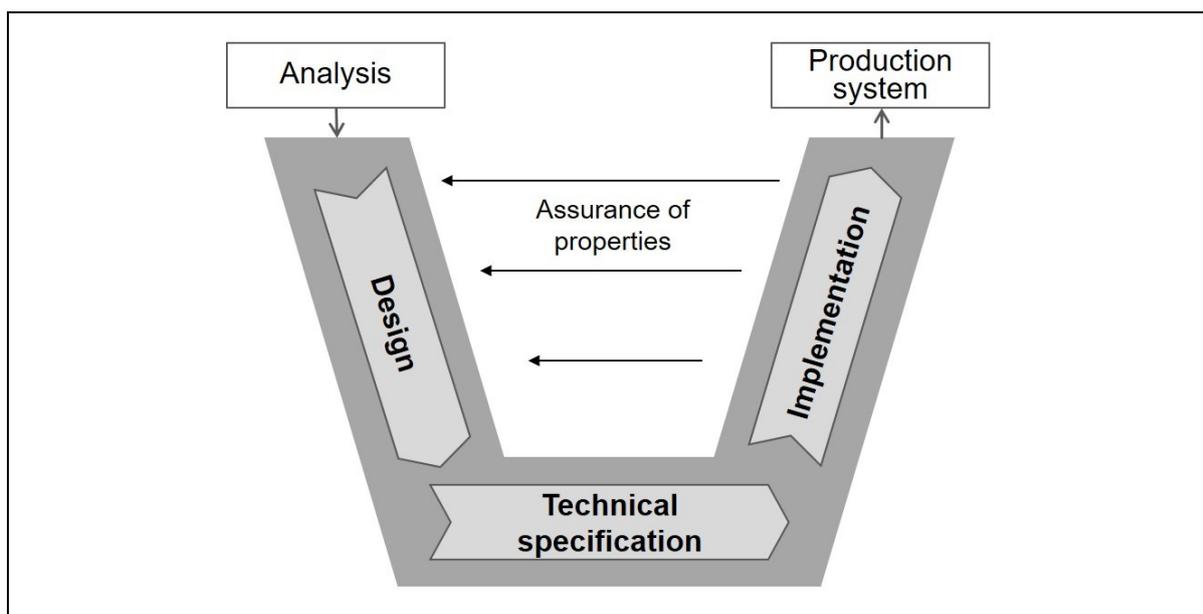


Figure 6-22: The PES COP process as V-model

The PES COP process constitutes an enabler to integrate innovative concepts from systems engineering (see Chapter 4.3.5) and mechatronic systems (see Chapter 4.3.2) into the production engineering process. Thus, the pure sequential arrangement of the production engineering phases can be easily transferred to the shape of a V-model for mechatronic systems (see Chapter 4.1.2). The analysis phase acts then as input for the left leg of the V which covers the design phase. The technical details of the single engineering domains are detailed during the technical specification phase, which constitutes the bottom of the V. The

right leg of the V supplements the previously defined phases with the implementation phase where the system is realized from bottom to top and continuously tested against its specification. The final result is the production system.

This proposed production engineering process shows significant differences to today's typical engineering procedure. Usually, the planning of a production system is strongly directed by the mechanical design. In contrast to this, the integration of PESCOMP implies a strong functional-driven instead of a hardware-driven design with the service structure as a central delivery. Together with the equipment structure and control logic, it represents a holistic system design on an abstract level. Obviously, these additional or extended deliverables necessitate the spending of increased efforts for the pronounced process planning and the new system planning phase. However, these additional efforts pay off by numerous potential benefits:

- **Time savings:** The major part of the additional efforts is anticipated from the detailed planning phase where efforts can be reduced. Simultaneously, these efforts constitute the basis for the parallelization of planning tasks during detailed planning which allows a shorter time-to-market at the end.
- **Seamless design:** The planning process describes an integrated procedure from a detailed process planning to a holistic system design, which finally merges into the domain-specific design. Explicit instructions to link the individual planning phases permit a seamless planning procedure. As a result, the current gap between the production process planning and the implementation of control procedures can be eliminated by the mapping between process description and service structure.
- **Dealing with complexity:** The step-by-step concretization of the production system as a whole enables an engineering in an easy to follow top-down manner. Moreover, the separation of planning results according to the individual perspectives on the production system provides an improved comprehensibility. Both aspects permit a systematic structuring of planning tasks and help to deal with the growing complexity of production engineering.
- **Higher flexibility and reusability:** The new rough planning phase and the system planning phase allow the reduction of the dependencies on hardware-specific details in the early planning phases. For this reason, a higher flexibility for possibilities of the concrete realization is given. Moreover, the planning results from the new rough planning phase and system planning phase provide a considerably higher reusability so that reconfigurations of an existing production system or similar planning tasks for a new one can be executed with less effort.

6.6 Model-driven Engineering Methodology

The concepts about SOA-AT, the design of equipment and control services, and the PESCOMP process are now put together to form an integrated model-driven engineering

methodology for service-oriented control procedures. In the following, it is abbreviated by the term MDE for SOA-AT. This thesis explicitly concentrates on the functional specification of the control system so that the methodology covers the analysis and design phases of the PES COP process.

A reference model determines the modeling workflow in accordance with the engineering process (see Chapter 6.5.1). The planning models are formally specified by meta-models and their development within the engineering process is examined in detail. Thereby, the models are specified in a general way so that they neither depend on specific modeling tools nor on certain technologies for the implementation. Moreover, structural blueprints are defined for the equipment model and the service model as reference architectures.

6.6.1 MDE for SOA-AT Reference Model

The general methodology of MDE for SOA-AT is represented by a reference model (see Chapter 4.2.1). It defines which models are generated in which development steps to enable an integrated proceeding and a seamless information flow (see Figure 6-23). It is separated into seven sections that are arranged in the vertical dimension and the horizontal dimension. The workflow is horizontally arranged in three sequential planning phases: analysis, abstract design, and concrete design. The two latter phases are additionally split up in vertical orientation according to the three different design aspects (see Chapter 6.3.2) that can be simultaneously developed. As a result, four planning models are introduced (see Figure 6-23):

- Process model (1)
- Equipment model (2a + 2b)
- Service model (3a + 3b)
- Control logic model (4a + b)

Since planning results of adjacent sections depend on each other, information needs to be shared between the models. During the development, the information exchange between two sections happens either by a direct information flow indicated with an arrow or by a mapping task between model elements of different models depicted as diamonds. In the latter case the mapping describes an additional planning step which is necessary to develop one model based on the results of a model from another section. The model-driven methodology comprises three mapping tasks (see Figure 6-23):

- Functions to services (A)
- Services to hardware components (B)
- Abstract hardware templates to concrete hardware templates (C)

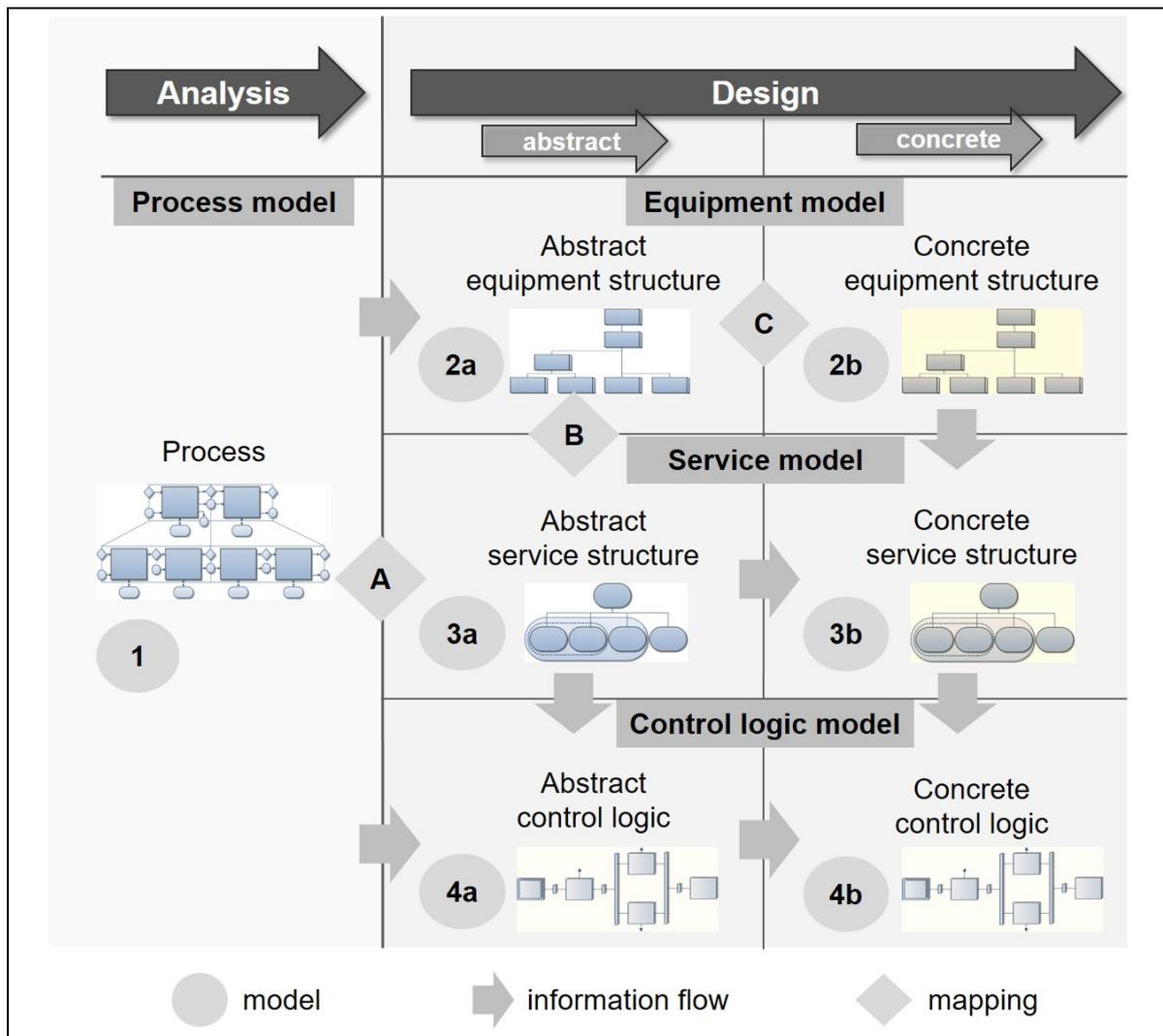


Figure 6-23: Reference model for MDE for SOA-AT

6.6.2 Modeling Concepts

Already today, modeling is a common tool to depict planning results in a user-friendly and comprehensible way. Thereby, the rules and features for modeling are usually defined by the modeling tool that is used for a specific engineering discipline, for example, a CAD application provides a set of modeling features for the mechanical design. To make the model-driven engineering methodology generally applicable without the dependence on a certain modeling language or modeling tool, metamodels as an abstract syntax define the set of modeling concepts, their attributes, and their relationships (see Chapter 4.2.1). Before these are determined in the upcoming sub chapters, a general framework with modeling concepts is given. It is based on OO and basic MDD concepts (see Chapter 4.1.1 & Chapter 4.2.1).

Four levels of modeling are defined according to the MOF levels M0, M1, M2 and M3 (see Figure 6-24). The planning model itself is built up by elements from level M0 and level M1

whereby the actual planning objects are represented by M0 elements being instances of libraries or catalogs elements from level M1. To define the rules for building the individual planning models, metamodels are defined that are represented by level M2. The general rules to build the metamodels are defined within the concept model which expresses the meta concepts of level M3 (see Figure 6-25).

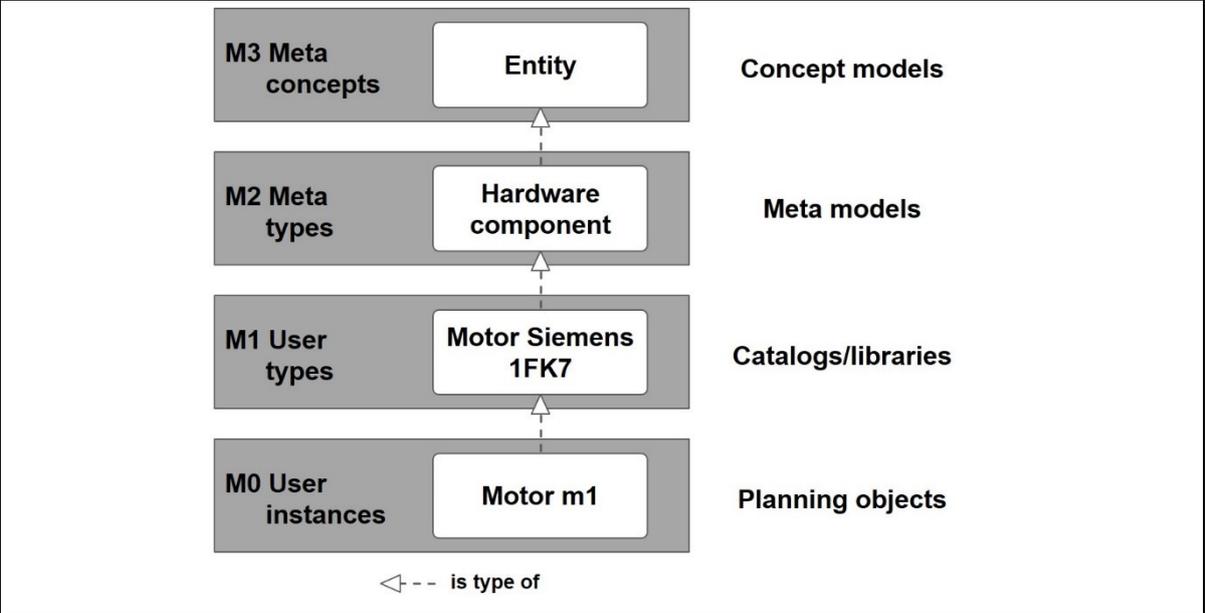


Figure 6-24: The four modeling levels with exemplary elements

The concept model comprises the element *entity metatype* which represents certain classes of entities according to the respective model like “service” or “hardware component”. The basic concept to depict elements of level M0 and level M1 is an *entity* which can either represent a planning object as an *entity instance* or a class of objects as an *entity templates* (see Figure 6-25). The actual planning objects are depicted on level M0 as *entity instances*. During the PESCOP process *entity templates* are used as elements of the service library that are instantiated to services of the planning model. The same holds true for concrete hardware components that are determined by assigning a device from the device catalog. Entities can have one or more *attributes* to describe certain characteristics.

Furthermore, entities can have dependencies to other entities which is expressed via a *relationship* between two entities. Relationships are unidirectional so that a target and a source needs to be determined. General relationships between entities are defined as *relationship metatypes* that are connected between two entity metatypes. Three relationship metatypes are already used within the concept model to express object-oriented modeling concepts. The first one is particularly needed to create instances of classes. If an element of a certain modeling level is a sample of a class on a higher modeling level, the sample element is linked to the class element via the *is type of* relationship. The second one indicates when a class is a generalization of another class. Therefore, the specialized class

is connected to the class whose properties it adopts via the *inherits from* relationship (see Chapter 4.1.1). Strongly related is the third relationship metatype which applies the inheritance principle in a looser way as *is concretization of* relationship. It is used between elements on the same modeling level and creates several categories and sub-categories of elements in the service library or device catalog (see Chapter 6.3.3). Another common relationship metatype indicates when an element *is part of* another element to present composition or modularization of elements. Beyond this, any other type relationship of between two elements can be introduced to express certain dependencies of model elements for the particular meta models.

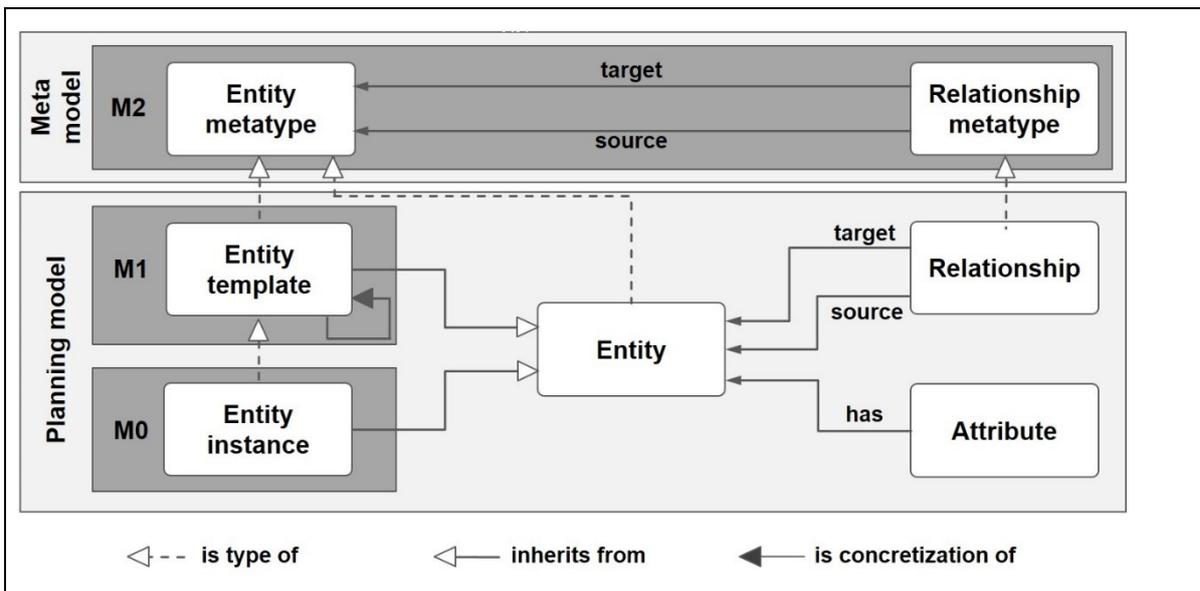


Figure 6-25: Concept model

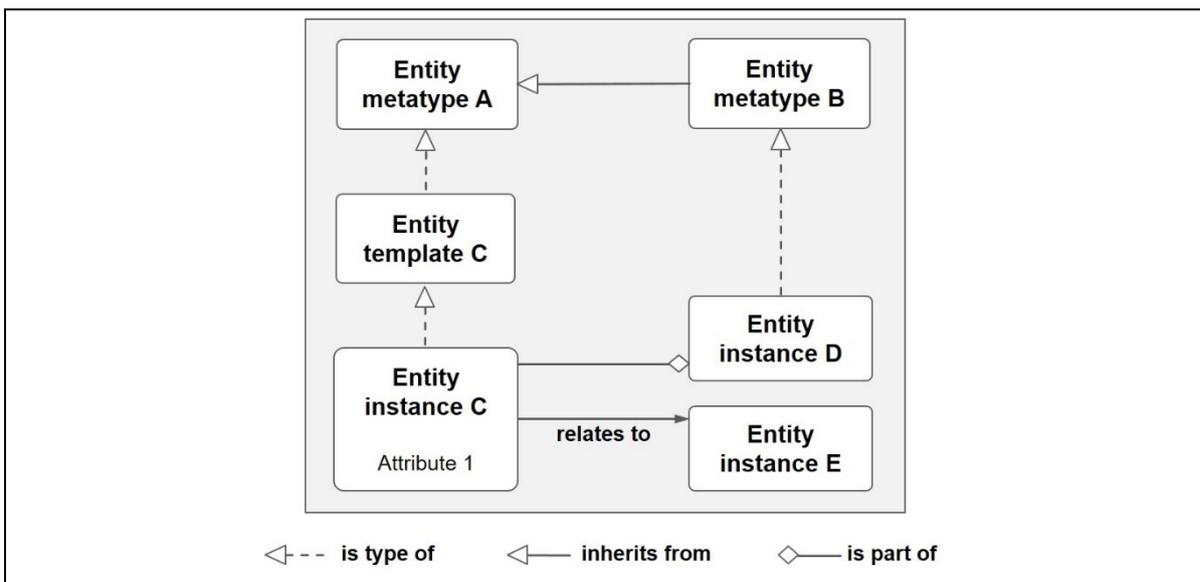


Figure 6-26: Graphical notation of the metamodels and models

The graphical representation of the metamodels and planning models are put in a similar way as the modeling concepts with some deviations (see Figure 6-26). All entities and entity metatypes are represented as rectangles. To provide a better overview and to make the graphical representation more comprehensive all attributes are depicted inside the rectangle and the relationships are presented as arrows. The most common relationships are indicated with a certain format; others can be introduced by normal arrows with labels to indicate their type.

6.6.3 Process Model

The process model depicts the results from the analysis phase. This primarily includes the description of the production process as production steps and the functional realization of the elementary process steps. Furthermore, it should be possible to depict potential requirements on the production equipment and information about the product states for each process step in order to adequately consider results from earlier planning phases.

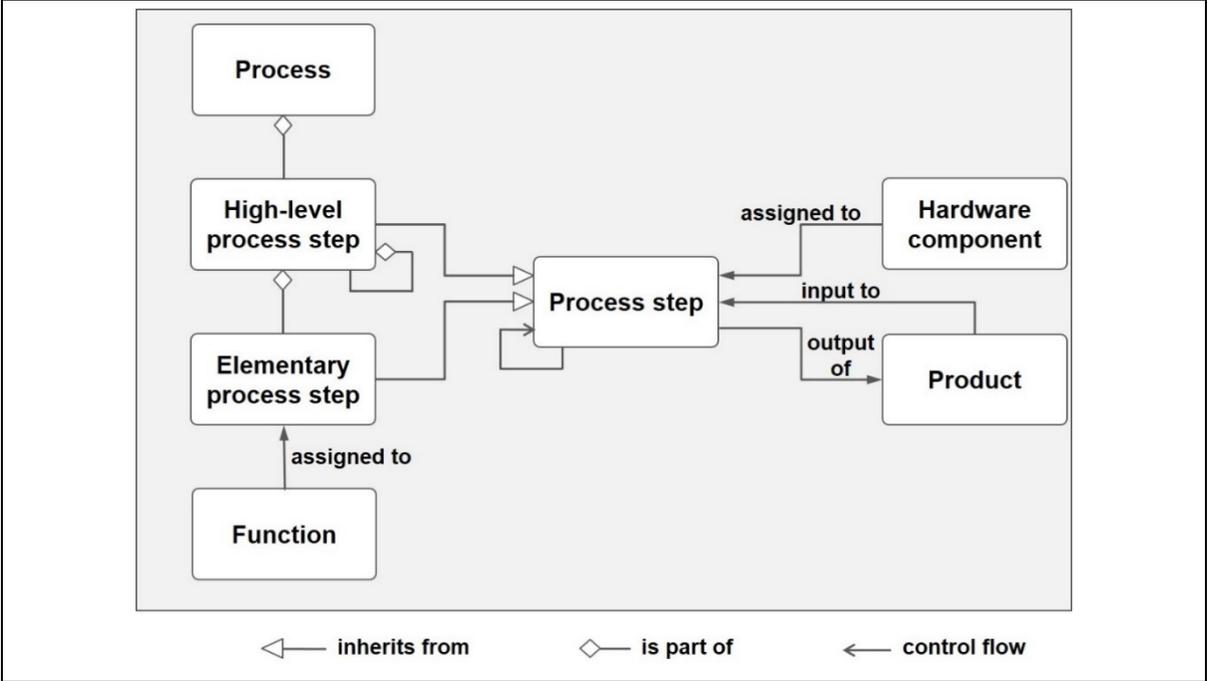


Figure 6-27: Metamodel of the process model (M2 level)

All concepts to create a process model of the PES COP process are defined within a metamodel (see Figure 6-27). The overall production process is represented by the element *process*. Each process model is depicted by a number of *process steps* that can be arranged in several granularity levels. Thus, *high-level process steps* are decomposed to sub processes that contain process steps again (see Figure 6-28). The decomposition is continued until *elementary process steps* are reached that are assigned to *function* elements which define their functional realization.

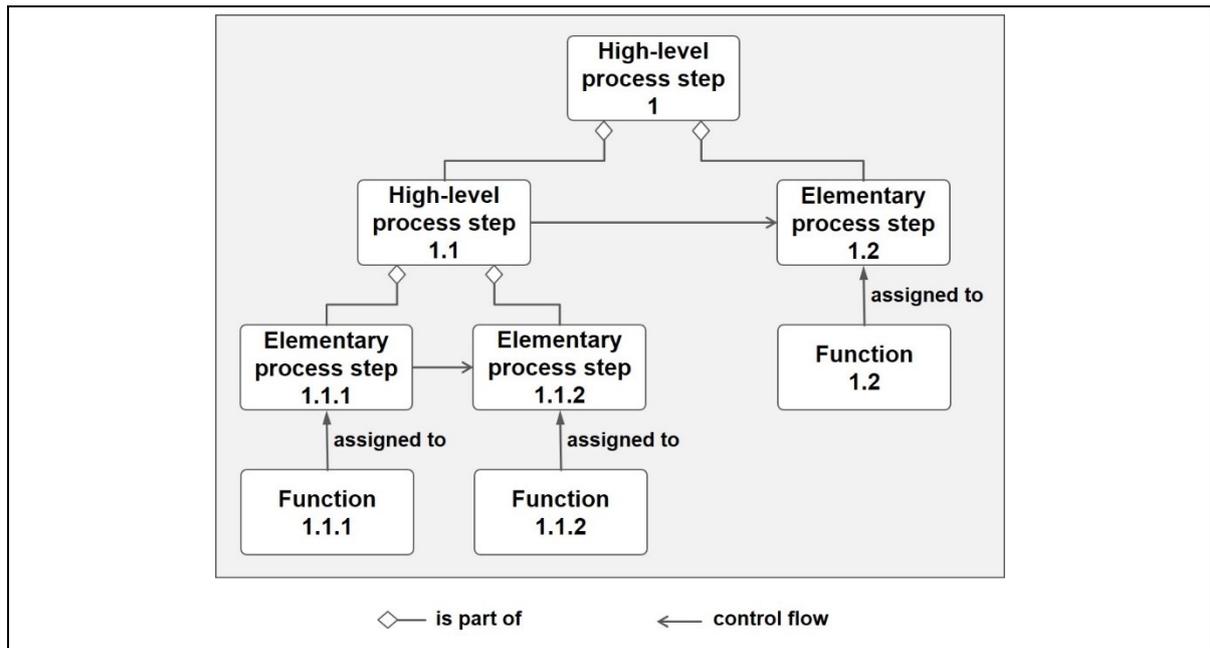


Figure 6-28: Exemplary instance of a process model (M0 & M1 level)

To determine the process flow order, the process steps are connected by *control flow* relationships that indicate which process step is followed as soon as another process step has been completed. Two further elements that are inspired by concepts from the VDI 3682 (see Chapter 4.2.2) help to enrich the process steps with further information. Since the product is formed by the production process, several states of the product can be defined as *product* elements that are *input to* or *output of* certain production steps. This additional information helps to connect the process planning with the previous product design. If certain types of machines or field devices are already determined to execute actions of the production process, this information is represented as *hardware component* elements which are assigned to the respective process steps.

6.6.4 Equipment Model

The abstract and concrete equipment structure are displayed within the equipment model. It illustrates the composition structure of the physical parts as individual *hardware components* (see Figure 6-29). Each hardware component can consist of a number of other hardware components according to the modularization principle. An additional relationship between hardware components shows which component is needed to *operate* another component. This illustrates functional dependencies in terms of a functional group (see Chapter 6.2.1) and is required for the correct implementation of equipment services. A component *extends* another component when its purpose is to exclusively serve the component. This would be the case when sensors are added in order to check certain states of a field device, for example, a cylinder with sensors that detect whether it is extended or not. The entity instances of the equipment model are *hardware component instances*. They

can be specified directly or created by instantiation of *hardware component templates* which represent entity templates and are collected within the device catalog.

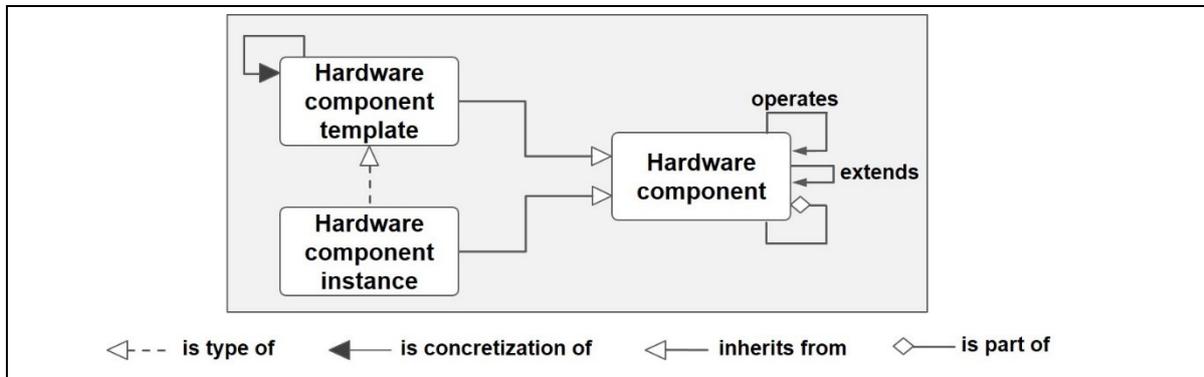


Figure 6-29: Metamodel of the equipment model

A reference architecture defines categories of hardware device templates that represent four granularity levels (see Figure 6-30). It specifies four categories of hardware components that are inspired by the ISA-88 (see Chapter 4.4.1). For manufacturing control the highest interesting equipment level concerns the scope of a *production line*. Each production line is separated into *work cells* with defined physical dimensions that execute sequential production steps. Work cells comprise a number of *field devices* that are the smallest self-contained hardware components. In many cases, certain field devices within one working cell strongly relate to each other when they build a functional group or a reusable hardware unit. To depict their coherence, the hardware component template *module* is used.

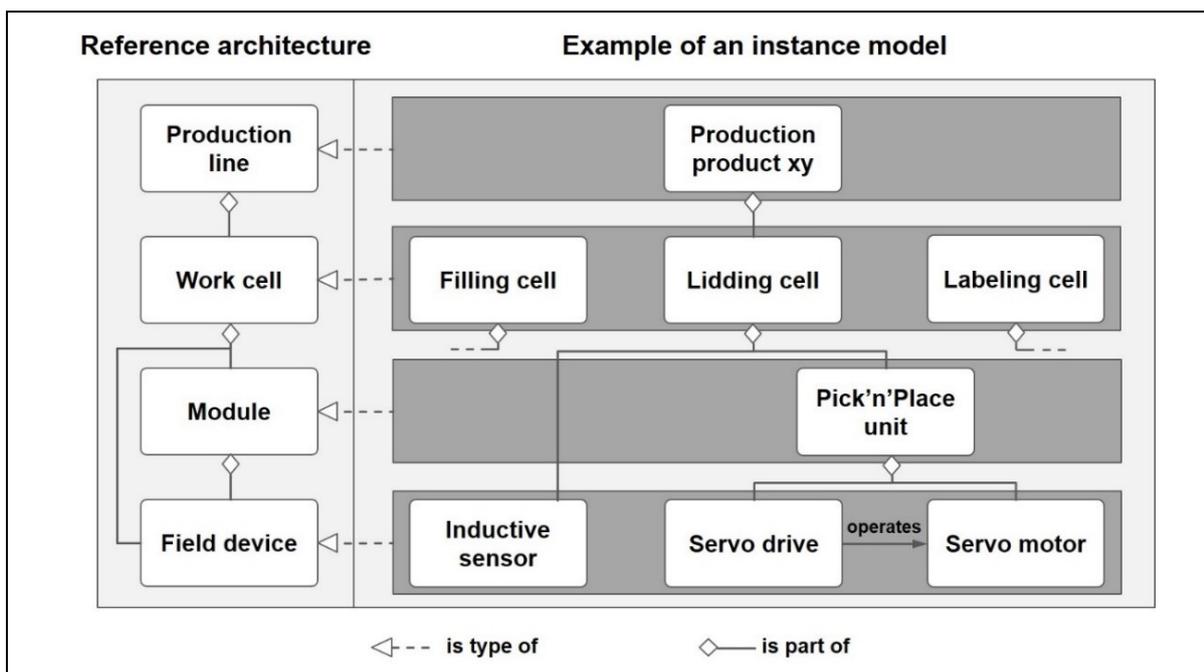


Figure 6-31: Reference architecture and example of an equipment model

To determine the individual elements of an equipment model, the usage of a device catalog is recommended. Library elements can represent elements of each level of the reference architecture and are particularly beneficial on the lower levels where the reusability is high. To pick suitable and available field devices for realizing the functionality of a work cell, commercial parts from an internal or external device catalog can be picked (see Chapter 6.3.3). On the higher levels a device catalog is rather applied to capture already defined work cells or modules in order to reuse them for other applications.

6.6.5 Service Model

The service model depicts the composition structure of the *services*, which define the functional structure of the production line (see Figure 6-32). Since the composition is realized through service orchestration, the service model indicates whenever a service *uses* other services to execute its functionality. Furthermore, the model defines the *service operations* as attributes of the respective services. The entity instances of the model are *service instance* elements, which represent the services are to be implemented according to a defined implementation concept to make the control program executable (see Chapter 7.2). They can be built directly or by instantiating a *service template* which is listed in the service library (see Chapter 6.3.3).

Besides the specification of the services, the service model also presents the dependencies between services and hardware components. Therefore, it defines two relationships that connect services with hardware components of the equipment model. The relationships indicate which hardware component *executes* the functionality of a service and if another hardware component is required to *implement* its functionality (see Chapter 6.2.1).

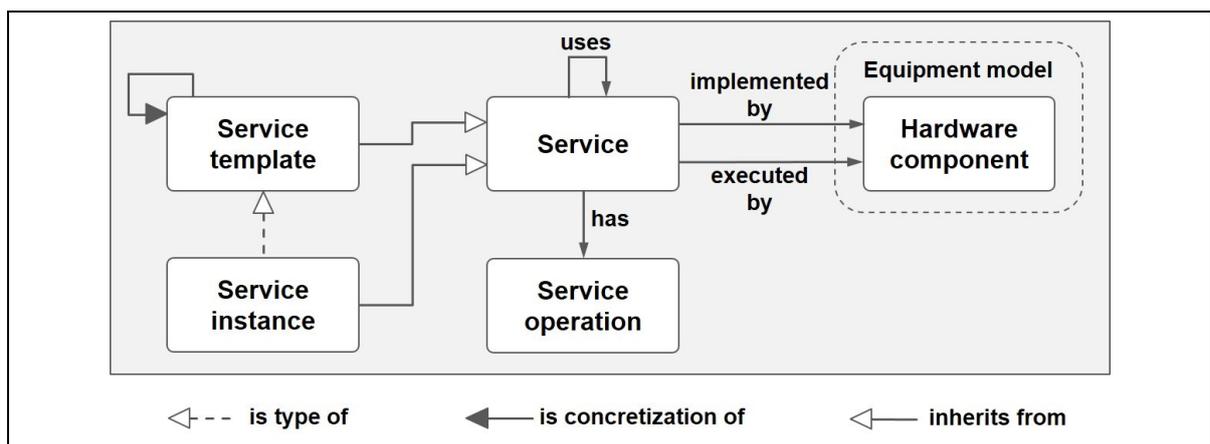


Figure 6-32: Metamodel of the service model

Similar to the equipment model, the entity metatype *service template* can have sub-categories to define a reference architecture for the service model (see Figure 6-33). Its structure can be directly derived from the reference architecture for service-oriented manufacturing control (see Chapter 6.2). Thus, a first division takes place in accordance

with the two service layers *equipment services* and *control services*. Equipment services can be further distinguished into *basic services* and *composed services*. For control services the sub-categories *process service*, *product service*, and *supporting service* exist.

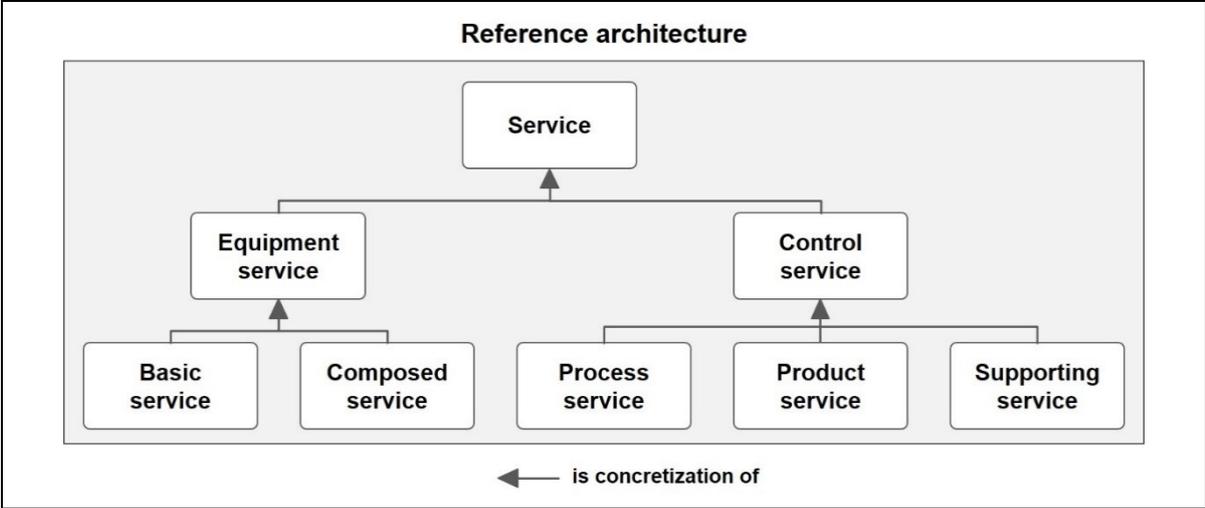


Figure 6-33: Reference architecture of the service model

6.6.6 Control Logic Model

The dynamic behavior of the services is specified within the control logic model. The element *control logic* symbolizes the complete logic that implements the functionality of a certain service operation which is executed as soon as the operation is called (see Figure 6-34). Thus, each control logic element is connected to a service operation entity of the service model that *represents* the control logic externally. Generally, the control logic is realized with logical instructions which can be defined by using pseudocode in an informal way independently of the technical realization. Later on, the instructions need to be implemented conforming to a respective implementation language.

In case of service operations making use of other services, the control logic can comprise an *orchestration logic* which can be modeled graphically in more detail than pure logical instructions. An orchestration logic is composed of *process steps* similar to the process model, which constitutes a direct input for the control logic model by providing a rough process flow. For the development of a control logic based on the process model, the process steps have to be detailed and alternately be connected to *control structure elements*. The control structures define in which logical order the process steps are executed, for example, as sequential process flow, conditional branches, or loops. When a process step is active the execution of associated logical instructions and external services can be triggered. In the latter case, another link to the service model is required to indicate which service operation is *called* by the process step.

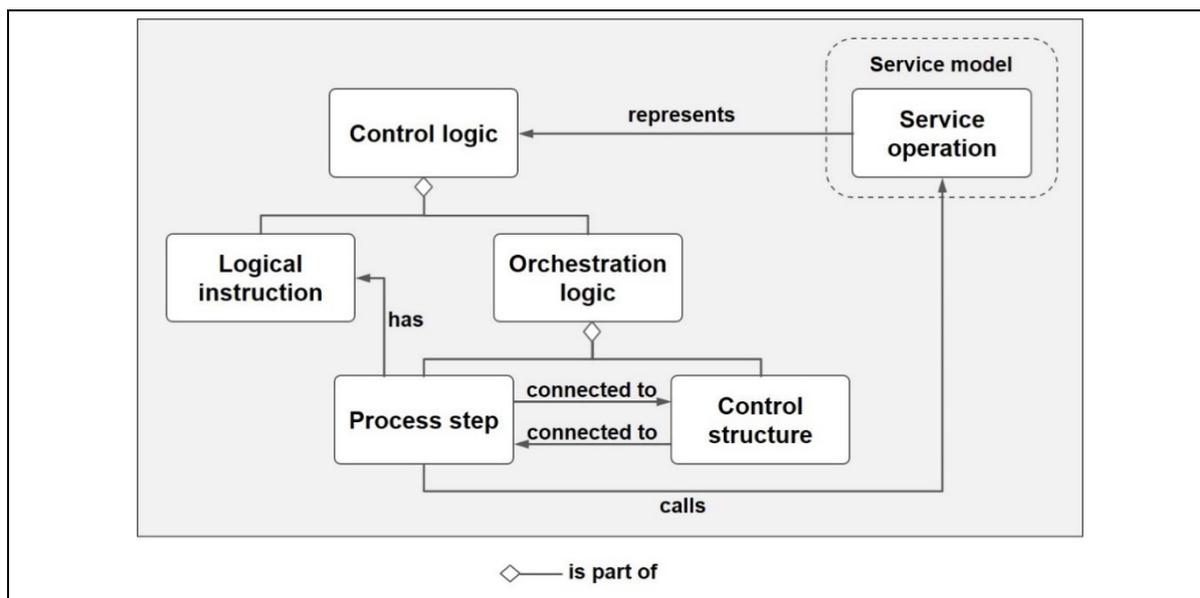


Figure 6-34: Metamodel of the control logic model

6.6.7 Model-driven Engineering Workflow

Now that the individual models for the different kinds of information have been defined, the detailed workflow for MDE for SOA-AT can be described (see Chapter 6.5.1). Therefore, the individual execution steps are examined with emphasis on the information flow and the mappings between the individual models (see Appendix A).

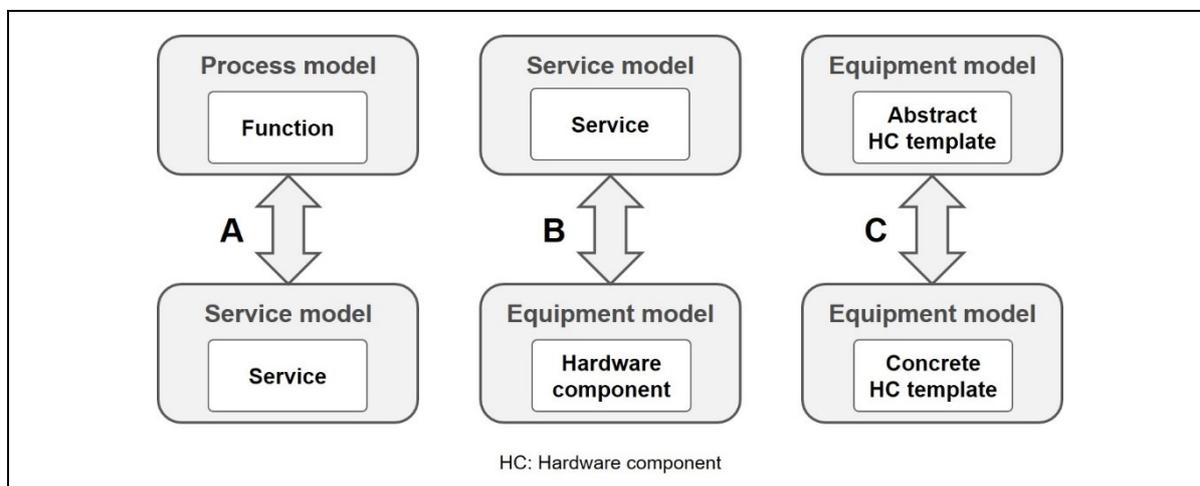


Figure 6-35: Mappings

During the analysis phase the process model is developed. The model represents the process description consisting of a number of process steps. The process steps are detailed in several levels until the functional realization can be determined for each step. They can be optionally enriched with information about the related product state and the required hardware component. The design phase comprises the parallel development of the equipment model, the service model, and the control logic model. These models are

developed in two concretization steps according to the degree of dependence on the production equipment. First of all, the development of the service model is initiated with the first mapping where services are determined that execute the functional realizations of the process steps (mapping A in Figure 6-35). After that, the service structure is detailed which includes the specification of the control strategy by defining all required equipment services and control services.

For the equipment model the first input about required hardware components is directly transferred from the process model. During the second mapping task the remaining hardware components are determined that are required to execute the services (mapping B in Figure 6-35). If the picked hardware components provide further services to those that are needed for the current process, these services are also included into the service model. The control logic model also receives direct input from the process model in terms of the process steps and the process flow order which serves as a first rough control logic. The control logics need to be detailed according to the desired control behavior of the system and the services that have been defined in the service model. The concrete design phase begins with the third mapping which refines the abstract hardware components to concrete hardware components wherefore commercial parts have to be chosen (mapping C in Figure 6-35). Based on this refinement, the service structure and the control logic need to be adjusted as well. As a last step, the final details of all models are worked out.

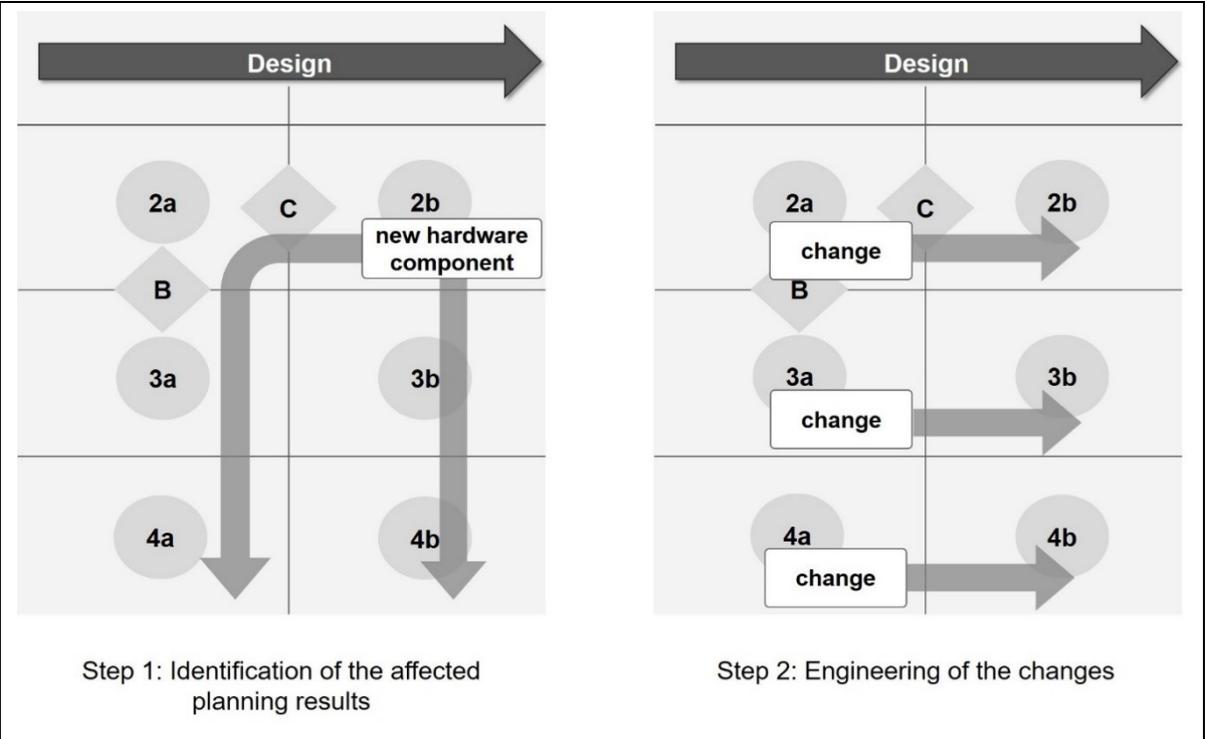


Figure 6-36: Reconfiguration process

So far, the general engineering workflow (see Chapter 2.3.1) and the above described workflow for MDE for SOA-AT focus on development planning tasks where the production system and its control system are defined from scratch (see Chapter 2.3.1). This is due to the fact that this scenario represents the most general case where the models are developed from the bottom up. However, reconfiguration tasks of existing systems occur much more often than complete replanning. Since the type and scope of the reconfiguration can vary within a wide range, the respective workflow for reconfigurations depends on the specific case. Nevertheless, the existing planning models provide an ideal basis to support the execution of any reconfiguration task because the dependencies between planning results are explicitly defined as relationships or mappings.

The starting point constitutes the changes that call for a reconfiguration, for example, new or exchanged hardware components or a modified control logic (see Figure 6-36). First, the affected planning objects are identified and the impact on other planning objects are analyzed by checking step by step the effects of the change on related planning objects in a bottom-up manner. After that, the consequences for all affected planning results need to be defined so that the change can be realized in the desired way. This happens like the general engineering process in a top-down manner starting with the planning objects on the highest granularity level. By these two steps, the scope of the reconfiguration is precisely defined and required modifications are specified.

7 Application Concepts

MDE for SOA-AT is defined as a general concept (see Chapter 6) detached from special development tools, IT technologies, programming languages, etc. In order to lift its relevance and expediency, this chapter describes application concepts for the standardized naming of planning objects and an implementation concept to transfer the concept into practical applications.

7.1 Standardized Naming of Planning Objects

During the execution of MDE for SOA-AT numerous planning objects are determined that are indicated by their name. Thus, the name stands for the meaning of the individual planning object and is directly connected to a set of information. This supports the cooperation of different peer groups that are involved in the engineering process by reducing information gaps through a common understanding.

Furthermore, naming standards are the fundament to effectively apply library concepts and thus, to fully support reusability of planning results (see Chapter 6.3.3). Consequently, the mapping tasks (see Chapter 6.6.7) benefit from standardized names by defining certain rules how an object A from a library would be mapped to an object B from another library. These rules could be stored and reapplied to support the engineer in finding suitable mapping partners.

However, the establishment of standardized names of planning results is complicated due to different interests and perspectives of various peer groups (e.g., OEMs, equipment suppliers, device manufacturers) that potentially are economic competitors. Additionally, the broad variety of production processes with different characteristics would lead to extensive efforts for the definition of generally applicable naming standards. Nevertheless, there already exist engineering standards that can help to establish company standards or even domain-specific standards. In the following, concrete proposals are given how existing naming standards can be used and how the mapping procedures can be executed based on them.

7.1.1 Naming of Functions

The determination of the functional realization of process steps constitutes a tool to link the elementary process steps of the process description with services for their execution (see Chapter 6.4.2). The possibility to choose from a certain pre-defined set of functions supports the development of the process description. As soon as a process step can be realized by

one or more functions from the standardized library, it can be ensured that an elementary process step is reached and the decomposition is finished.

A suitable source to fill such a library for manufacturing processes are the functions that are defined within the standards VDI 2860 and DIN 8580 (see Chapter 4.4.2). Beyond this, the library can be extended by special functions that are repeatedly used. For this thesis, two elements are added: an element for reading and writing information via RFID and a placeholder element which can be chosen if no other library element fits. As a result, the library with functions of manufacturing processes comprises 43 elements (see Appendix B). The graphical representation of the functions supports the design with graphical models for a high comprehensibility so that the symbols can be directly attached to the respective process steps within the process description (see Figure 7-1).

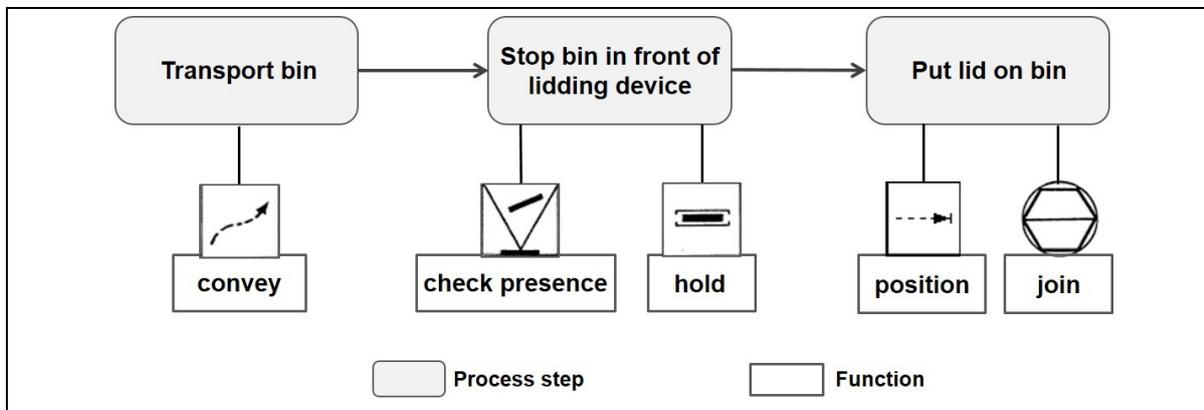


Figure 7-1: Exemplary functional realization of a process description by using the function library

7.1.2 Naming of Services and Service Operations

Standardized naming of services is highly important to leverage the full potential of SOA; particularly to permit reuse of services and support the binding of services (see Chapter 3.4.1 & Chapter 6.2.2). To apply the naming scheme for services (see Chapter 6.3.1) the verb/noun and adjective/noun combinations have to be chosen to express the function of the service and its form of energy or operating principle. Therefore, the eCI@ss standard is used to serve as a source to deliver standardized terminology (see Chapter 4.4.2).

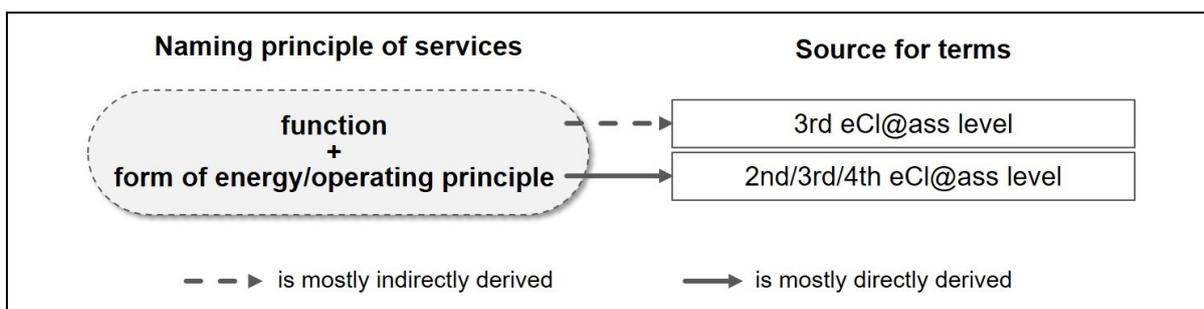


Figure 7-2: Service naming based on eCI@ss

For defining a term for the function the third eCl@ss level “group” provides the input to determine a standard term for the function of a device. For instance, for the “synchronous motor” the function “rotate” and for the “optoelectric sensor” the function “detect” are assigned. The second level “main group”, the third level “group”, and the fourth level “sub group” often contain terms that describe the operating principles for sensors or form of energy for actuators. The examples above would then be extended to “rotateElectric” and “detectOptic” (see Figure 7-2). Although a direct derivation of appropriate terms is not always possible, it provides a viable basis on which a service library with consistent naming of the services can be created.

Each service comprises a number of service operations, which also needs to be determined. For simple services, the operations can be directly derived from the term of the function of the service, for example, the services “detectOptic” and “detectMagnetic” have the operation “detect”. More complex services are characterized by a broader set of operations with a higher amount of input and output data. This set of operations with certain inputs and outputs needs to match with the control interface of the device. Generally, the more complex the functionality of a field device is, the more specific is its control interface. Consequently, deviations between the general service of an abstract hardware component and the service of the concrete hardware component become more likely with rising complexity of the functionality of the hardware component (see Chapter 6.3.4). This requires the adaption of the service after executing the mapping task C (see Chapter 6.6.7). In accordance with the service hierarchy defined for the service library (see Chapter 6.3.3) a further subdivision of a hardware-independent service into hardware-specific services would be needed in this case.

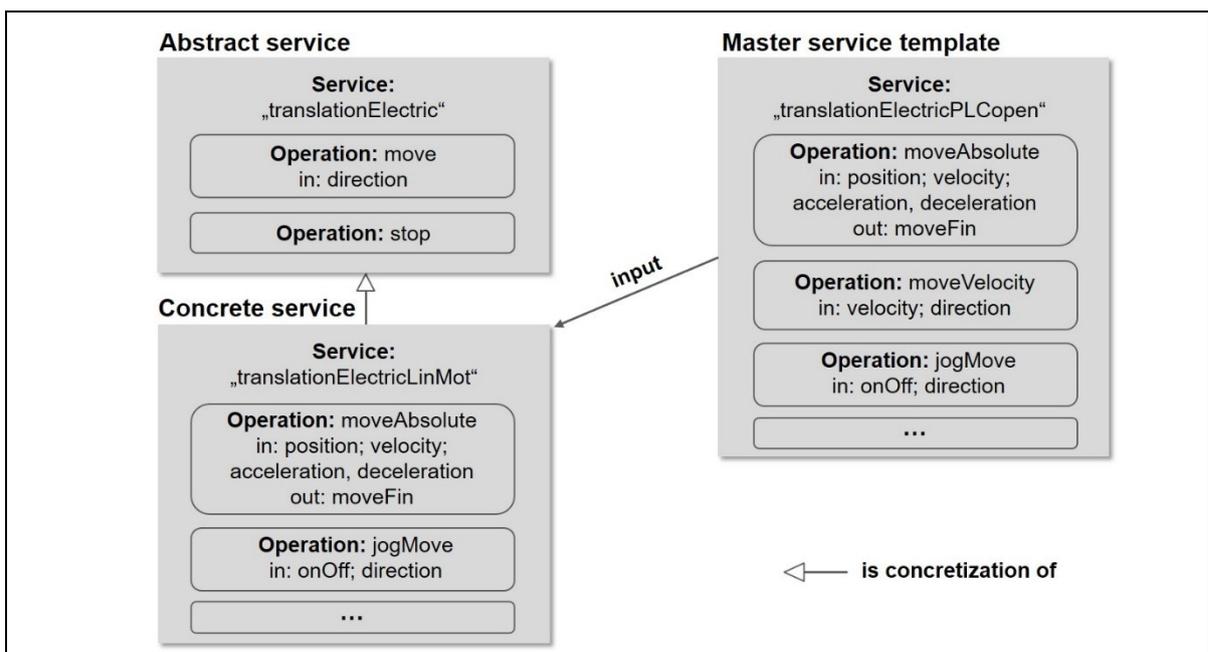


Figure 7-3: Refinement of a service with master service templates

Although hardware-specific services make the standardization and reuse of services more difficult, the situation can be improved by the concept of *master service templates* (see Figure 7-3). Such a master service template can be applied for certain device categories with complex functionality and comprises all possible operations. In contrast to this, the abstract service template comprises just a minimal set of general service operations. For defining the concrete service, the abstract service is extended by device-specific operations taken from the master template. This enables a compromise between standardized service templates, the handling of complexity, and an application of the SOA concept to all possible device functions. In the ideal case the master services are also derived from automation standards. For example, the PLCopen “Motion Control” standard (see Chapter 4.4.2) is suitable to define a master service template for motion control applications.

7.2 Implementation Concept

In the following, one possible implementation concept is presented, which describes how existing modeling languages, modeling tools, and technologies can be combined to turn the MDE for SOA-AT method into a real control application (see Figure 7-4). The focus is laid on the optimal support of the design phase of the PESCO process with a suitable representation of the sub models with all cross-model dependencies. Moreover, a suitable representation of the process model and a proposal for the realization of the services is given.

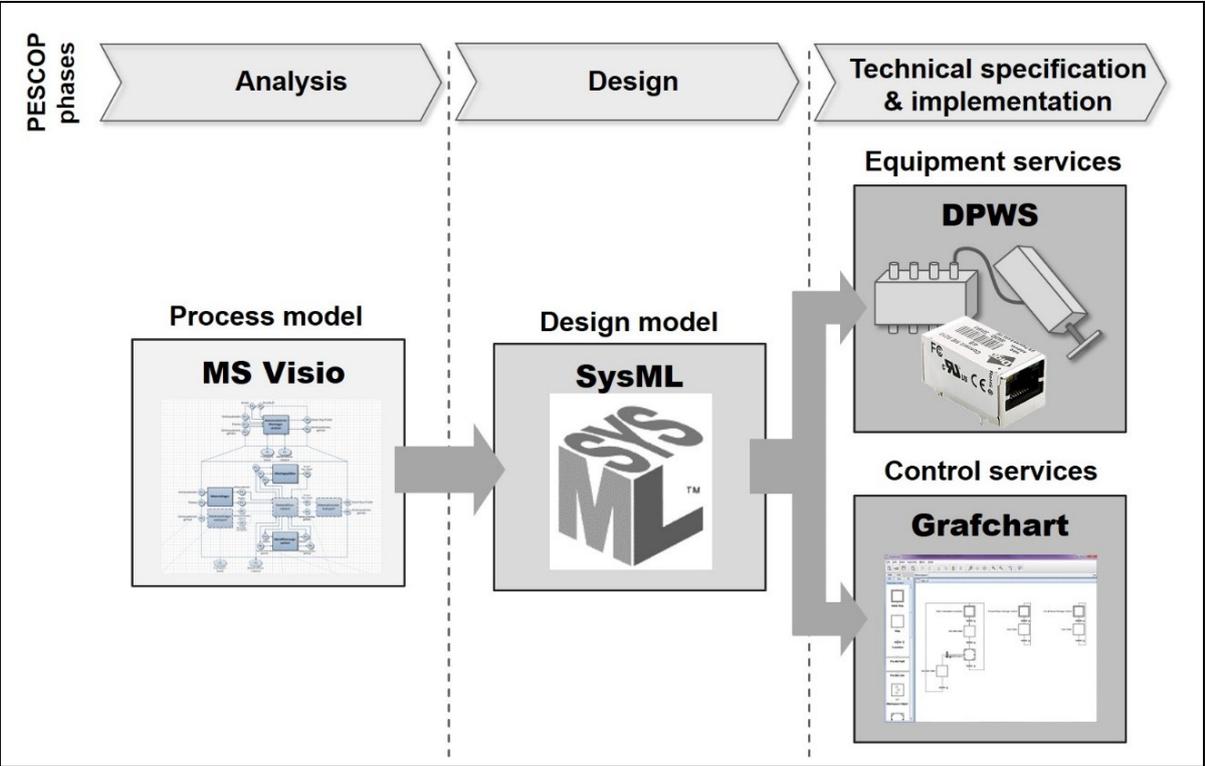


Figure 7-4: Implementation concept

7.2.1 Representation of the Process Model

The analysis phase is executed with a general-purpose process modeling tool to gather the input information for the design phase in a graphical way. The types of modeling objects like process steps, hardware components, relationships, etc. are distinguished by different shapes (see Figure 7-5). The format of the shapes is inspired by the graphical modeling language defined in the VDI 3682 guideline (see Chapter 4.2.2). For this thesis, Microsoft Visio is used to execute this task, but similar modeling programs can be used to achieve the same representation.

For splitting up high-level process steps into sub processes, several levels of the process model can be developed. The highest granularity level is level 0 which comprises high-level process steps that are detailed in level 1 and so on. The number of levels should be chosen depending on the complexity of the overall production process. Each process step gets a process step ID as a number so that it can be clearly identified. The numbering is applied according to the granularity level so that the process steps within a sub process start with the number of its high-level process step which is extended by a consecutive number.

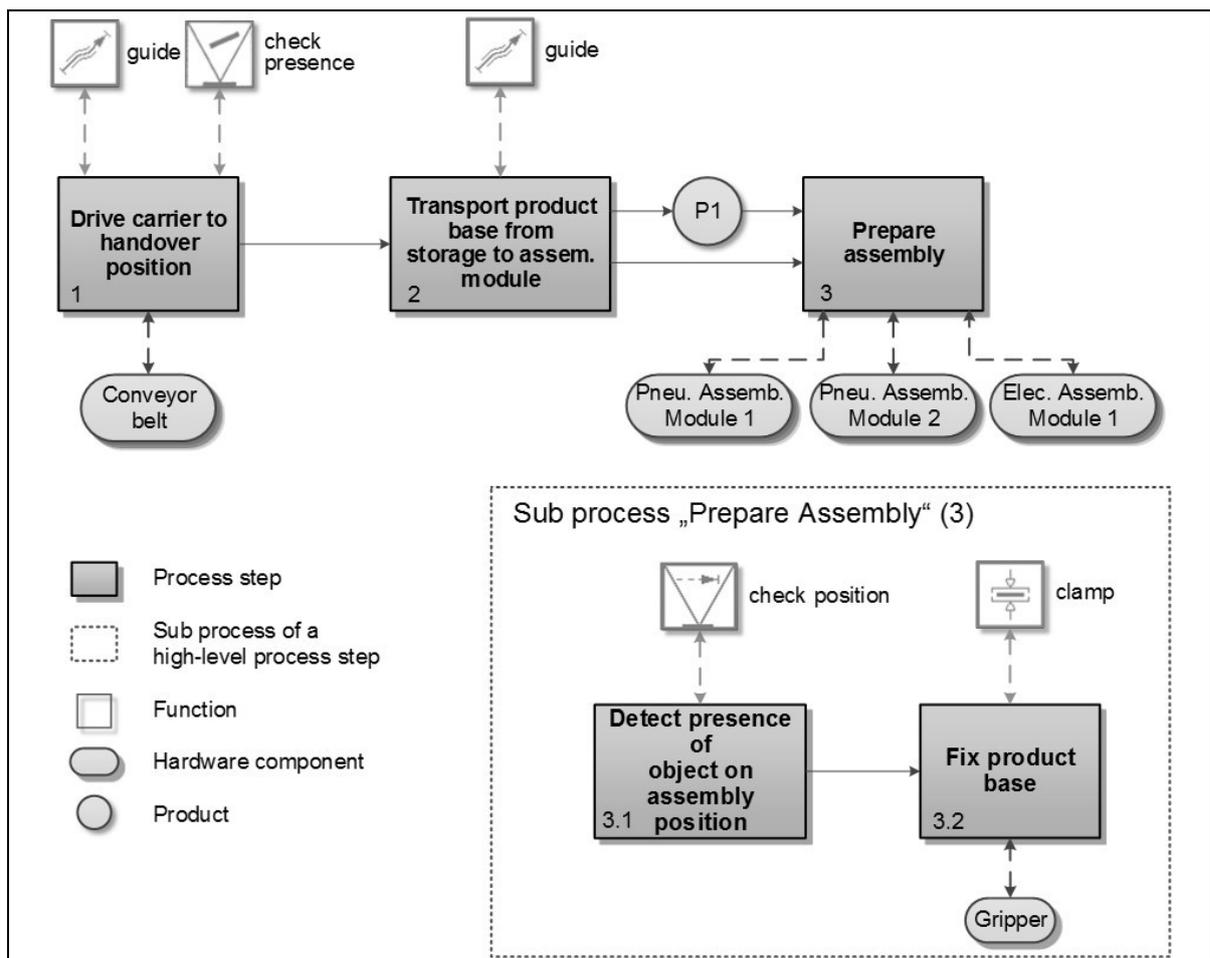


Figure 7-5: Process model example with MS Visio

7.2.2 Representation of the Design Phase Models

Since the planning models of the design phase are the core result of MDE for SOA-AT, a formal representation of the equipment model, the service model, and the control logic model is chosen. A proper modeling language for this constitutes SysML/UML which is applied in this thesis by means of the UML modeling tool Altova UModel (see Chapter 4.2.2). In the following, a definition is given how SysML/UML modeling elements are used to create the modes in accordance with the metamodels (see Chapter 6.6).

Generally, the overall system is designed as one SysML/UML model, which comprises several diagrams to implement the sub models (equipment, service, control logic). Having all information within one model allows to link modeling elements from different sub models to show their dependencies (e.g., services to hardware components). The entity instances representing the individual objects of the system are designed with SysML blocks. Entity template elements taken from the device catalog or the service library are depicted as UML classes. The relationships between entities can be partially represented by pre-defined SysML/UML relationships that are provided by UModel. Other relationships are realized by using the general *dependency* relationship which is then enhanced by a certain stereotype. The categories of hardware devices and services of both reference architectures are also implemented as stereotypes that can be assigned to the respective blocks and classes.

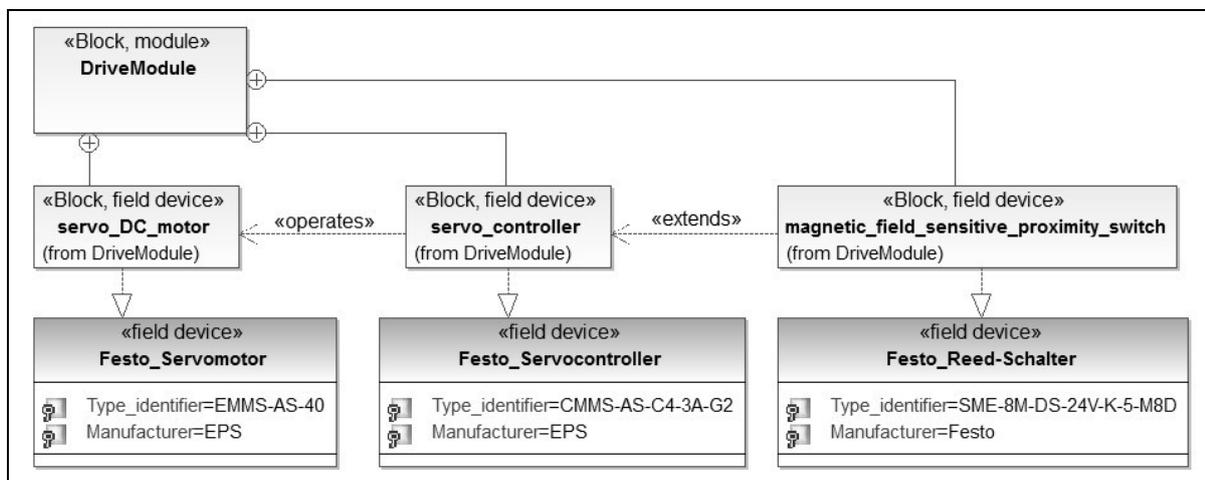


Figure 7-6: Presentation of the equipment model as Block Definition diagram

The equipment model is realized as Block Definition Diagram with the following elements (see Figure 7-6):

- **Blocks** represent hardware component instances.
- **Classes** represent hardware component templates.
- **Class property “Manufacturer”** indicates the name of the manufacturing company of the field device.

- **Class property “Type identifier”** indicates the type identifier under which the device can be found in the device catalog of the manufacturer.
- **Relationship “containment”** (solid line with a “plus” sign at the end) represents the “is part of” relationship between two hardware components.
- **Relationship “realization”** (dashed line with a hollow triangle) between blocks and classes represents the “is type of” relationship between an instance and template of a hardware component.
- **Relationship “realization”** (dashed line with a hollow triangle) between two classes depicts the “is concretization of” relationship between two hardware component templates within the device catalog.
- **Stereotypes “operates” and “extends”** refine the “dependency” relationship between two hardware components.
- **Stereotypes “field device”, “module”, “work cell”, and “production line”** indicate the category of the hardware component according to the reference architecture (see Chapter 6.6.4).

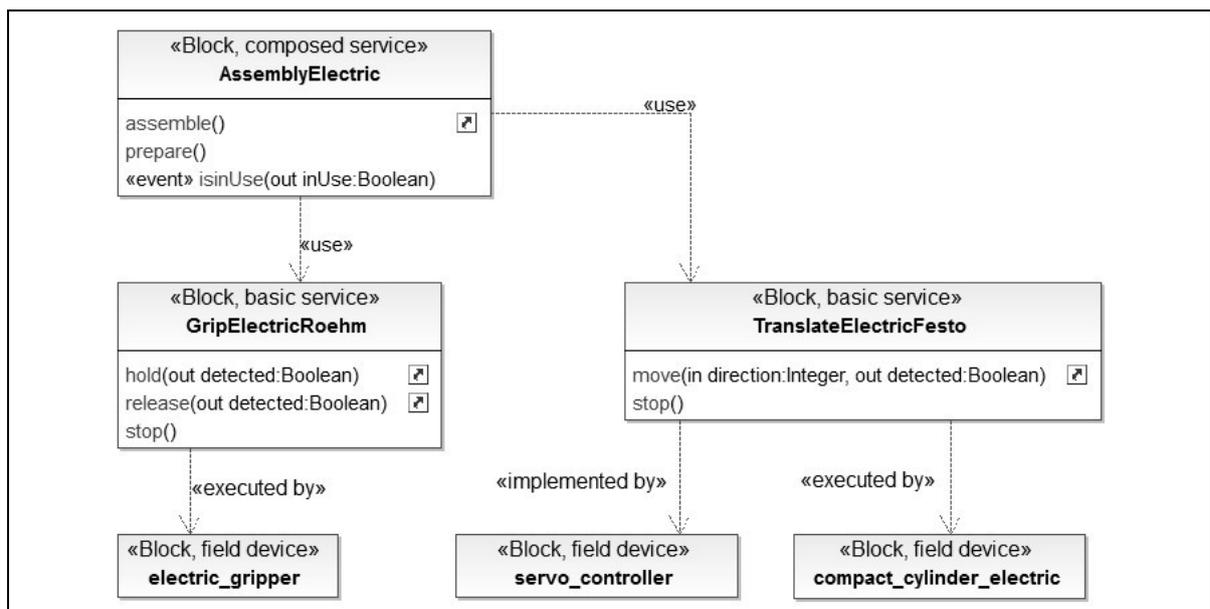


Figure 7-7: Presentation of the service model as Block Definition diagram

The service model is also depicted as a Block Definition Diagram (see Figure 7-7):

- **Blocks** represent service instances.
- **Classes** represent service templates.
- **Operations of blocks and classes** represent the service operations that belong to a service.
- **Relationship “use”** implements the “uses” relationship between two services.

- **Relationship “realization”** (dashed line with a hollow triangle) between blocks and classes represents the “is type of” relationship between an instance and the template of a service.
- **Relationship “realization”** (dashed line with a hollow triangle) between two classes depicts the “is concretization of” relationship between two service templates within the service library.
- **Stereotypes “executed by” and “implemented by”** refine the “dependency” relationship between services and the hardware components to indicate the “executed by” and “implemented by” relationships.
- **Stereotype “event”** tags a service operation in case it works according to the evening principle (see Chapter 6.3.1).
- **Stereotypes “basic service”, “composed service”, “process service”, “product service”, and “supporting service”** tags the service blocks with the respective category of the service reference architecture (see Chapter 6.2.2).

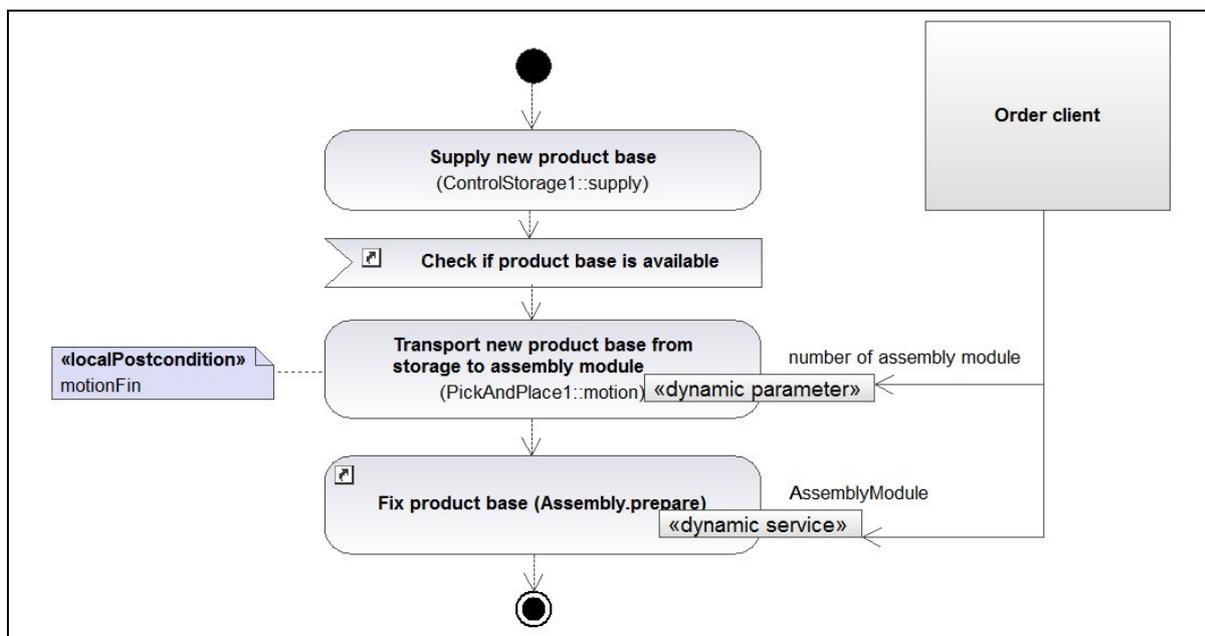


Figure 7-8: Presentation of the control logic model as Activity Diagram

The control logic models, which define the control logic of services, are depicted as Activity Diagrams (see Figure 7-8):

- **Activities** encapsulate the control logic of a service operation that is modeled as an Activity Diagram. The “is part of” relationship of process steps and control structures to the control logic is indirectly realized by including the elements into the diagram.
- **Hyperlinks from service operations to activities** express the “represents” relationship to connect a service operation with its control logic.
- **Actions** represent process steps that can have pre/post conditions to trigger or end their activation. Three different kind of actions are used:

- **CallOperationActions** represent the process steps that have a “calls” relationship to another service operation which is linked by the “operation” property of the action.
- **CallBehaviorActions** comprise the property “behavior” which links to another activity. This allows to split one control logic into several parts.
- **AcceptEventActions** (rectangular with arrow shape on the left side) wait for a certain event from a service operation for continuing the control flow. The respective service operation is connected via a hyperlink.
- **Control/object flow, initial/final/decision/fork/join nodes, exception handler, etc.** realize the control structures to design the flow of control within the Activity Diagram.
- **Object nodes** represent sources or pieces of information that are needed to execute the control logic (e.g., “order client” in Figure 7-8).
- **Input pins** of actions illustrate that the action receives information from outside that is needed to execute the action.
- **Stereotype “dynamic service”** of an input pin indicates an abstract service allocation (see Chapter 6.2.2) so that the service which is called by the action is dynamically determined.
- **Stereotype “dynamic parameter”** of an input pin expresses that the input of a service operation which is called by the action is dynamically determined.

7.2.3 Realization of Services-oriented Control Procedures

To apply and evaluate the MDE for SOA-AT methodology on real use cases the execution of the subsequent PESCOP phases technical specification and implementation are necessary to develop an executable control procedure from the design model (see Chapter 6.5.1). Therefore, a control system is required where the services can be deployed to (see Chapter 3.3.3). The system architecture can be realized in different ways from implementation of the services in a conventional PLC system to more innovative solutions as distributed control system (see Chapter 3.2). In the following, a concept is presented which describes how the equipment services and control services are realized whereby non-functional requirements on the control application like response time, availability, security, etc. are neglected.

Implementation of Equipment Services

The way how equipment services as elementary building blocks of SOA-AT are deployed has an impact on the flexibility and adaptability of the production system. A common approach of innovative control concepts is to bring the intelligence as near as possible to the physical system to provide autonomous entities [Thom12]. For SOA-AT systems this matters particularly for the equipment services due to their strong dependency on the production equipment. This can be obtained in terms of mechatronic components (see

Chapter 4.3.2) that are built by modularized hardware components with embedded controllers that deploy the services. The embedded controller serves as a service gateway, since it executes the software for implementing the service and provides the interface between the production equipment and the control system.

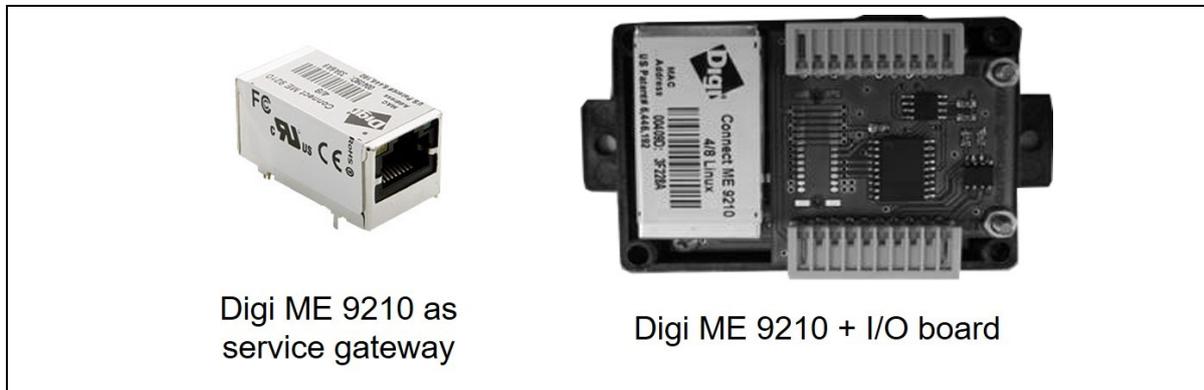


Figure 7-9: Service gateway [FigDigi01]

The “Digi ME 9210” embedded system is selected as a suitable technology for service gateways (see Figure 7-9). It comprises a 32-bit processor with 75MHz and 8MB SDRAM, has small dimensions (36.7mm x 19.05mm x 18.67mm), and supports a range of communication interfaces like Ethernet, SPI, and I²C, and general-purpose inputs and outputs [Digi16]. To make the Digi ME 9210 universally applicable, an additional I/O board has been developed which provides eight digital inputs, eight digital outputs, and a RS232 communication interface to connect the field devices and the power supply. The Ethernet interface constitutes the connection to the control system.

For the implementation of the services, the SOA technology DPWS is chosen (see Chapter 3.4.5) because it already provides a package with some extended functions which make the usability for equipment services convenient. Especially the functions to discover the services, to realize eventing operations (see Chapter 6.3.2), and the transport-neutral addressing are important features. Furthermore, the available toolkits SOA4D and WS4D can be used to develop the DPWS services as C programs for the embedded controllers.

Implementation of Control Services

The main emphasis of the services on the control layer is the implementation of a certain control logic making use of services on the equipment layer. Thus, the main requirements for their realization are the representation of the logic in an easily comprehensible and well-structured graphical modeling language and an available tool for design and execution. Both are fulfilled by the process modeling language Grafchart in combination with the tool JGrafchart (see Chapter 4.2.2). Furthermore, the semantics of SysML/UML Activity Diagrams and Grafchart are similar because both modeling languages are inspired by Petri Nets and State Charts. Thus, the control logic models can be transferred straightforward

from the Activity Diagrams into a JGrafchart program with some simple mapping rules (see Figure 7-10).

The basis for a seamless data transfer constitutes the XMI representation of the design model in SysML/UML (see Chapter 4.2.2). Since JGrafchart also uses a XML-based data format, a transformation tool *SysML2JG* has been developed that executes the transformation automatically without loss of information (see Appendix J) [Ade13]. After importing the control logic, some further concretization of the JGrafchart program is required in order to make the control logic executable and robust in terms of exception handling, the connection to uplink control systems, etc.

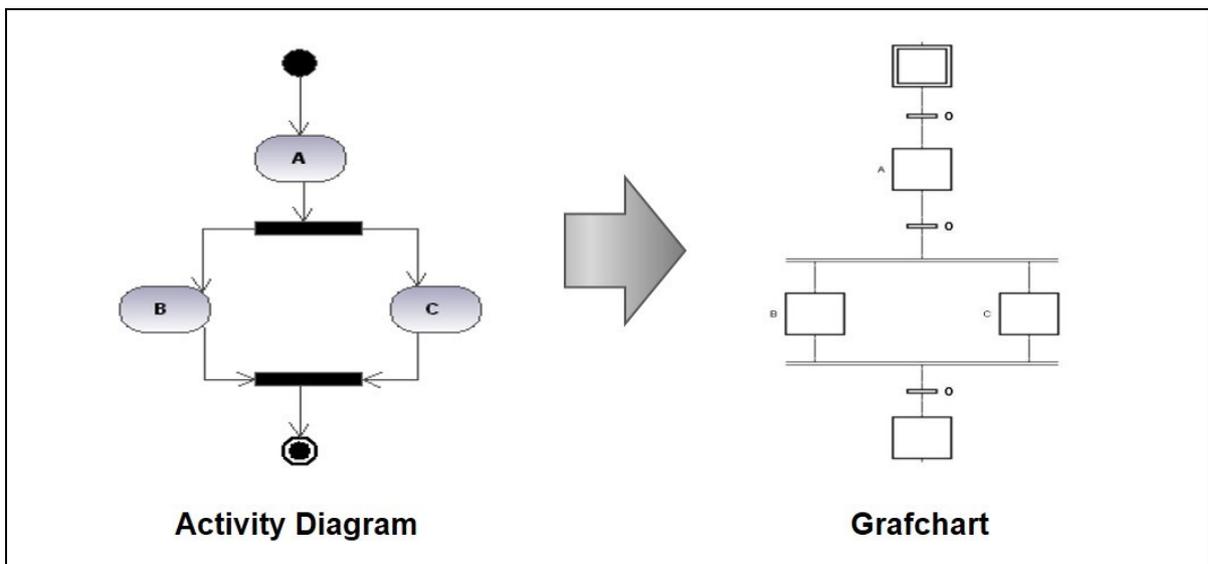


Figure 7-10: Transformation of an Activity Diagram to a Grafchart

The identification of the potential of JGrafchart to serve as an engineering tool and runtime environment for service-oriented control procedures led to a cooperation between the *SmartFactory*^{KL} and the Lund Institute of Technology. The result of this cooperation are new functions to connect and operate external DPWS services that have been introduced to JGrafchart Version 2.1.0 [Theo13]. With these extensions, a direct connection between equipment services implemented in DPWS on embedded devices and the control logic realized by JGrafchart can be established during runtime.

8 Proof of Concept

The last step of this thesis constitutes the proof of concept of the elaborated concepts from the previous chapters. Therefore, a use case is described where the MDE for SOA-AT methodology and its application concepts are applied for several application scenarios. Besides the illustration of the practical relevance of the new methodology, the scenarios provide the base to assess the new methodology against the state of the art.

8.1 Description of the Use Case

8.1.1 Demonstration System

The *SmartFactory*^{KL} is a legal non-profit association and acts as a multi-vendor research, development, and demonstration center for innovative, industrial automation technologies [Zühl10]. By means of several demonstration systems, the practical applicability of new ICT concepts and technologies is shown and proven at installations with industrial production equipment. One demonstration system represents a production line that produces an electronic key finder. It can be connected from a smartphone via Bluetooth and be triggered to flash a light or generate a signal tone. The individualization of the product is possible by a customized product cover with an individual engraving. The line comprises four work cells (see Figure 8-1 from right side to left side): milling unit, commissioning unit, assembly unit, and manual work unit.

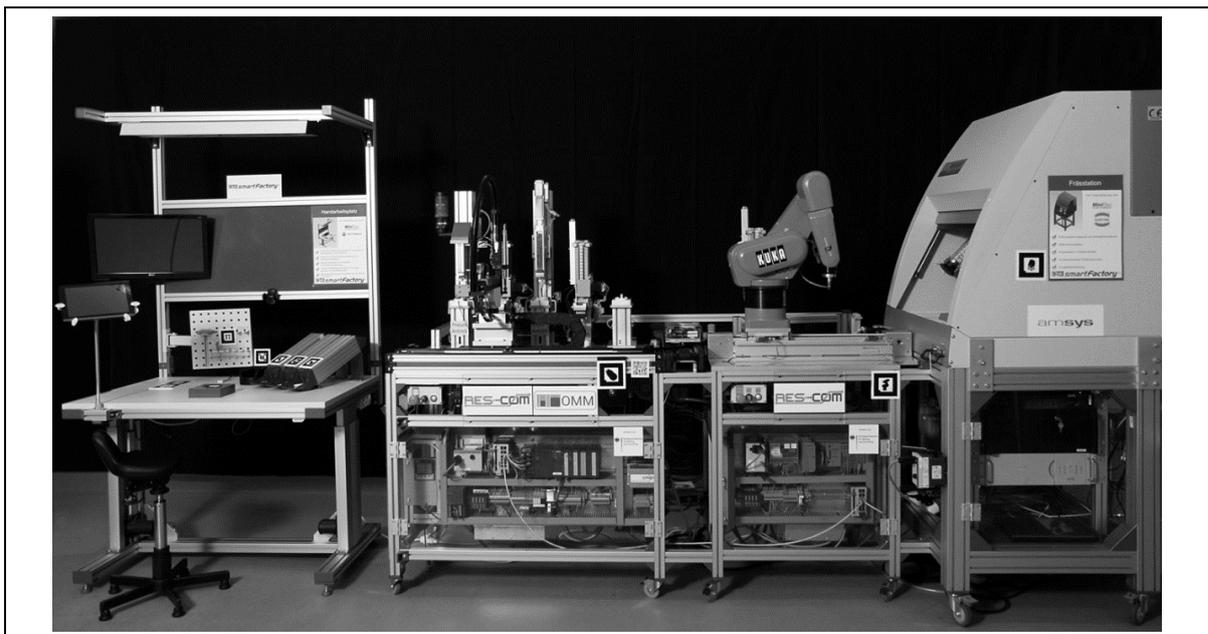


Figure 8-1: The *SmartFactory*^{KL} demonstration system

The product consists of a product base, a product cover, and a circuit board which is placed inside (see Figure 8-2). The product cover contains a RFID chip that acts as a product memory. It contains information about the current status of the product during the production process and product-specific information like the details about the customized engraving. Moreover, a data matrix code with the Bluetooth address is attached to the circuit board, which is read and afterwards written into the product memory before the product is assembled.

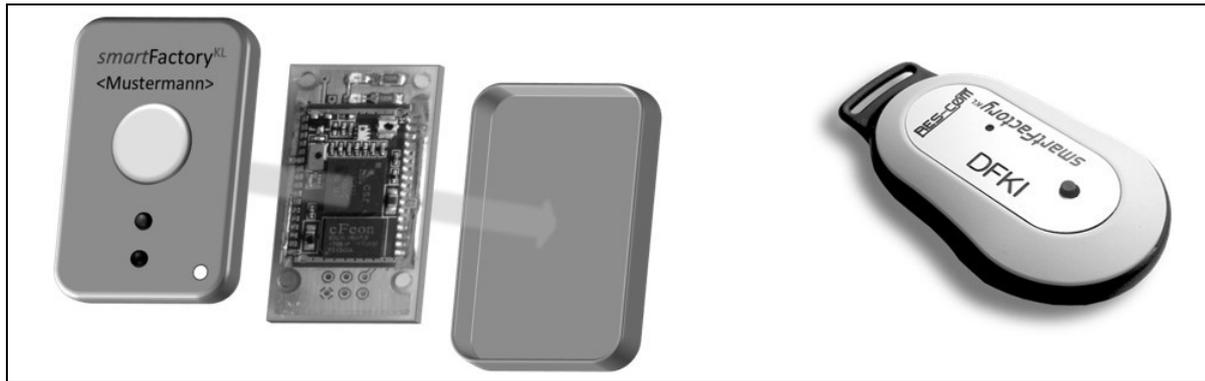


Figure 8-2: Product parts (left) and final product (right)

8.1.2 Assembly Unit

The concepts of this thesis have been applied at the key finder production line with focus on the assembly unit, which comprises more than 50 field devices with actuators, sensors, and controllers. The purpose of this work cell is the assembly of the three product parts to the final product. The process taking place in the assembly unit starts with the handover of the product cover from the commissioning unit to the assembly unit via a product carrier transported on a conveyor belt. Previously, the product cover is processed within the milling unit. The other two product parts are held in storages within the assembly unit (storage product base and storage circuit board). The unit comprises three assembly modules that assemble the final product by joining the product parts. One uses electrical energy to operate the actuators whereas the other two have pneumatic devices. A pick-and-place module realizes the main material flow to transport the product parts within the unit and the final product to the slide which hands it over to the manual work unit. Additionally, some safety equipment, a valve terminal, a RFID reading/writing (R/W) device, and a data matrix reader are part of the work cell.

The production process within the assembly module can be executed in several varieties. Since the assembly modules differ in the form of energy they are using, they have a different set of field devices leading to individual properties. Furthermore, the product carrier can host up to three covers at once. Depending on the current process order, the process in the assembly unit is started with one, two, or three product covers in the queue and a certain assembly station is picked for each product. These decisions are taken by a supervisory

application called order client which initiates the orders and transmits the information to the PLCs of the work cells.

In the initial state the assembly unit was controlled by a Siemens Simatic S7 PLC which executes the program to implement the assembly process and connects the work cell to the order client (see Figure 8-3). The field devices are connected via a fieldbus or conventional I/O wiring to the PLC.

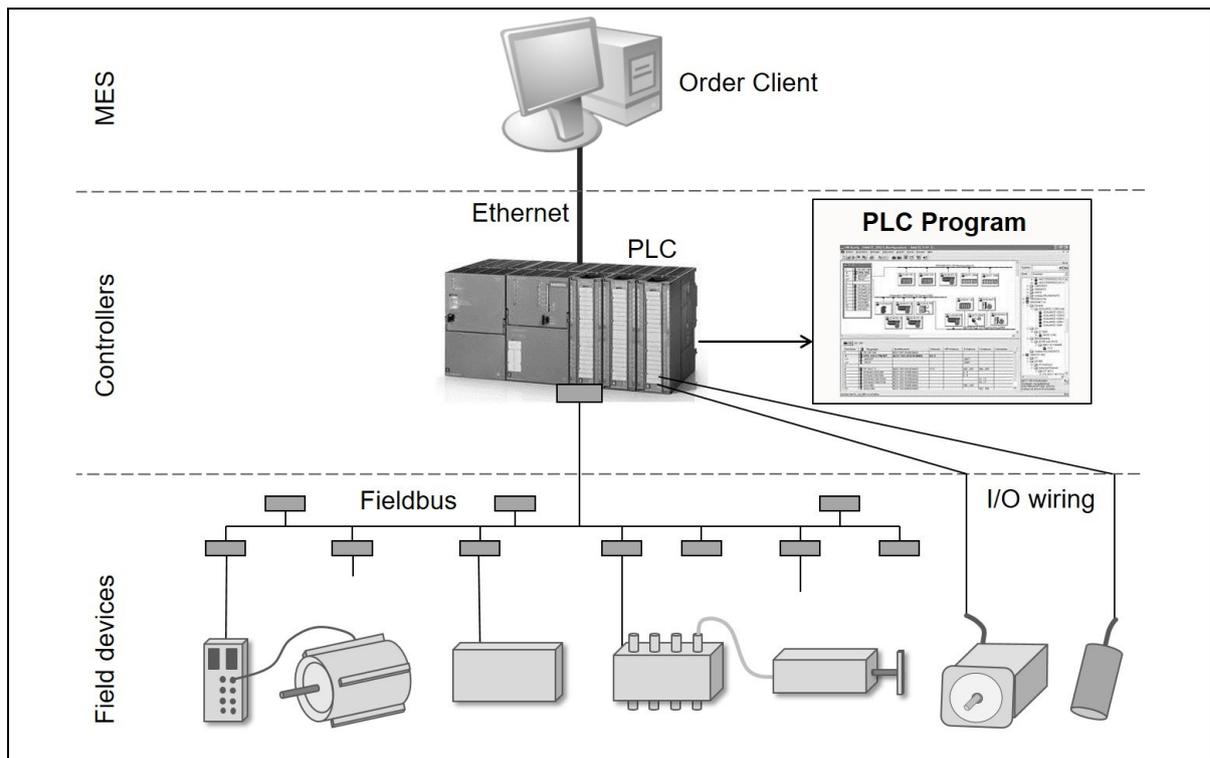


Figure 8-3: Conventional control architecture [FigSim01]

8.2 Application Scenarios

In the following, the MDE for SOA-AT methodology (see Chapter 6.6) is applied to the assembly work cell of the *SmartFactory*^{KL} demonstration system by following the application concepts (see Chapter 7). According to the types of factory planning (see Chapter 2.3.1) the two scenarios development planning and reconfiguration will be investigated because they comprise the planning tasks that are covered by the engineering methodology.

8.2.1 Execution of the Development Planning

The development planning scenario covers the design of the assembly unit from scratch. Since the mechanical and the electrical equipment of the work cell is already there, it is assumed that the design of the hardware is executed in similar concretization steps that have been passed through during the actual development of the demonstration system.

This concerns particularly the fact that the detailing of the pick-and-place module takes place in a later point in time after all other hardware devices have been defined.

Analysis

The basis for the analysis phase are the product specification (see Chapter 8.1) and a rough layout of the production line. The rough layout depicts the four work cells contained by the production system and the stages of the product (see Appendix C.1).

The process model is developed with three granularity levels. The superior level 0 describes the overall production process of the production line (see Figure 8-4 and Appendix C.2). The process step “Automated assembly” is then detailed on level 1 (see Appendix C.3). Three process steps require another detailing level on level 2 to decompose all high-level process steps into elementary process steps (see Appendix C.4). Afterwards, functional realizations are determined by assigning functions to each elementary process step. As far as possible, hardware components are added to the process steps to reflect the requirements on the production equipment. This makes it apparent that the pick-and-place module is a very central component because it is used by many process steps.

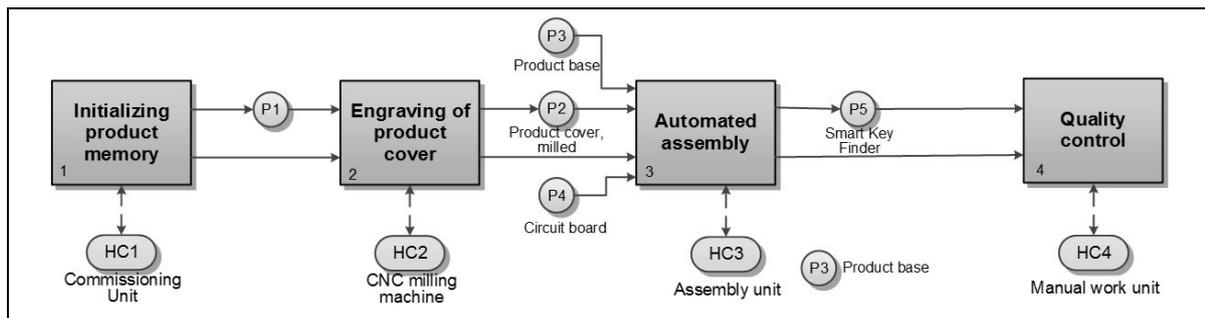


Figure 8-4: Top level of the process model

Design

First, the abstract design models are developed based on the analysis phase. By executing the mapping A, the services for the single process steps are derived from the process model (see Appendix D.1). Based on this the service model is developed (see Appendix G). One supervisory process service “AutomatedAssembly” is defined which accesses four basic services and seven composed services which represent the functions of several modules of the equipment model (see Appendix G.1). Each of these composed services consists of several basic services and in some cases also another level of composed services. For example, the composed service “AssemblyPneumatic1” uses the basic service “DetectOptic5” and two other composed services “VerifyingTranslatePneumatic1” and “VerifyingGripPneumatic1” (see Appendix G.4). In total nine composed services and 39 basic services are specified within the service model. The basic services are enriched with

the relationships “executed by” and “implemented by” to the respective hardware components.

In parallel, the hardware components defined in the process model are transferred to hardware component instances of the equipment model. It depicts the “AutomatedAssemblyUnit” as work cell (see Appendix E) and comprises a valve terminal as a single field device and eight modules: the three assembly modules, both storages, the pick-and-place module, the conveyor belt module, and a safety equipment module. Each of the modules consists of a number of field devices so that in sum 54 field devices are defined for the complete work cell. The individual field devices are determined based on the service design by means of mapping B (see Appendix D.2). Besides the “is part of” relationship also the “operates” and “extends” relationships are determined between the field devices.

The control logic model details the control flow of the process service “AutomatedAssembly” (see Appendix H). Due to its complexity it is divided into a high-level control part (see Appendix H.1) and a sub process (see Appendix H.2) which comprises the execution of the actual process steps. Thus, the process model is transferred to a separate SysML activity “AutomaticAssembly” where the control logic is detailed with control structures and the service calls. Since composed services also implement their service orchestration by a certain logic, it might be worth to model their control logic, too. In this case, the logic of the storages is considerably more complex compared to the other composed services so that it is designed in detail in a separate model (see Appendix H.3).

The concrete design starts with mapping C where hardware component templates are assigned to the hardware component instances within the equipment model (see Appendix F). Therefore, templates of commercial parts are picked that can be bought from certain vendors. The detailing of the equipment model also comprises the concrete definition of how the pick-and-place module is realized (see Appendix D.3). Based on the concrete equipment model the other models are detailed accordingly. For the service model the services of the pick-and-place module are added or modified (see Appendix D.3). Moreover, the abstract services “translateElectric” and “gripPneumatic” are split up in several device-specific services, for example, “translateElectricStep” of the stepping motor and “translateElectricFesto” for the Festo cylinder. These changes have to be taken over to the control logic model. Additionally, the high-level control part of the “AutomaticAssembly” process is specified with start-up and exception routines and the communication to the order client.

Technical Specification and Implementation

For the realization of the service-oriented control system the assembly unit is first extended by a number of service gateways (see Chapter 7.2.3). Each of the modules receive one service gateway with the exception of the RFID R/W device and the matrix code reader that get their own service gateway. This makes in total 10 embedded devices which are installed with respective I/O boards and connected to the field devices (see Figure 8-5 showing the pneumatic assembly module as an example).

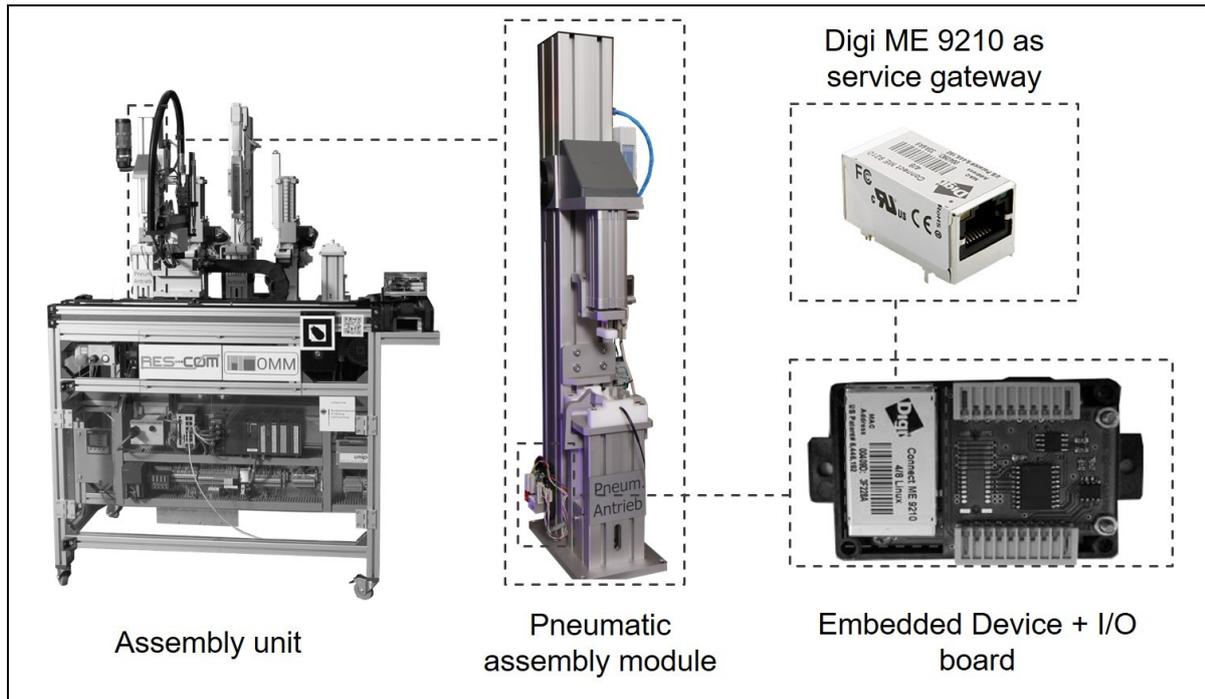


Figure 8-5: Hardware extensions with embedded devices [FigDigi01]

The equipment services are then implemented on these service gateways as DPWS services. In sum 48 DPWS hosted services are programmed and deployed on the embedded devices. A conventional Windows PC serves as the execution platform of the control services that are implemented as a JGrafchart program (see Chapter 7.2.3). Therefore, the control logic model is transferred to a Grafchart and the DPWS interfaces to the equipment services are established. Additionally, the connection to the order client, which is running on the same PC, is realized via a socket connection. After a last detailing and testing phase for the services the complete assembly unit is integrated into the entire production line with its new service-oriented control architecture (see Figure 8-6).

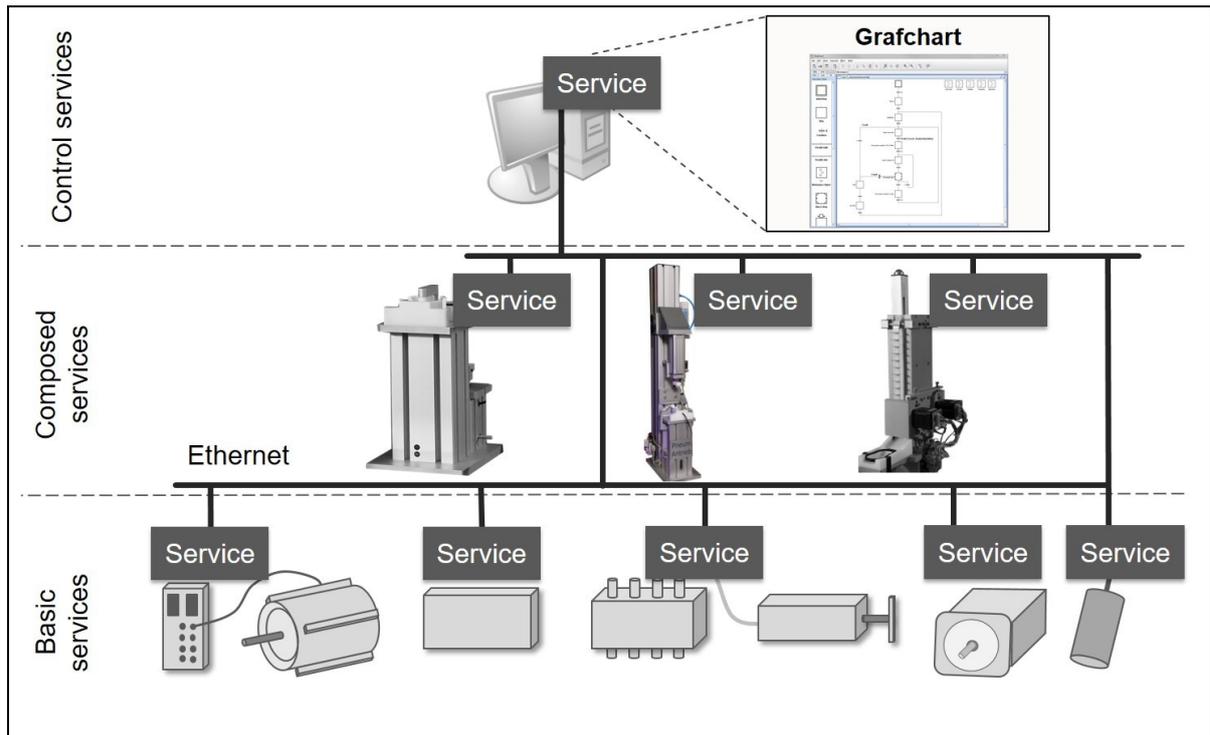


Figure 8-6: Service-oriented control architecture

8.2.2 Execution of Reconfiguration Tasks

Since the scenario of reconfiguration can happen in many different ways, three sub scenarios are investigated in the following.

Reconfiguration 1: New process variant

A new process variant is introduced where the assembly of the product is executed within the manual work unit instead of the assembly unit. This scenario could occur when the assembly modules are not available due to maintenance tasks or similar events. To understand the required changes, the starting point of the change is reflected within the planning models (see Figure 8-7). In this case, the initial change is made in the process model which gets extended by a branch in parallel to the process step “Automated assembly” with the process steps “Prepare manual assembly” and “Manual assembly” (see Figure 8-8).

For this process variant, the assembly unit forwards the single product parts directly to the manual work unit which doesn’t require any additional functionality of the production equipment. Thus, the equipment model and service model remain the same. The only modification is executed within the control logic model where a second sub process “Manual Assembly” is added as another SysML activity “PrepareManualAssembly” (see Appendix I.1). For the implementation, the logic of the process service “AutomatedAssembly” has to be extended in JGrafchart accordingly.

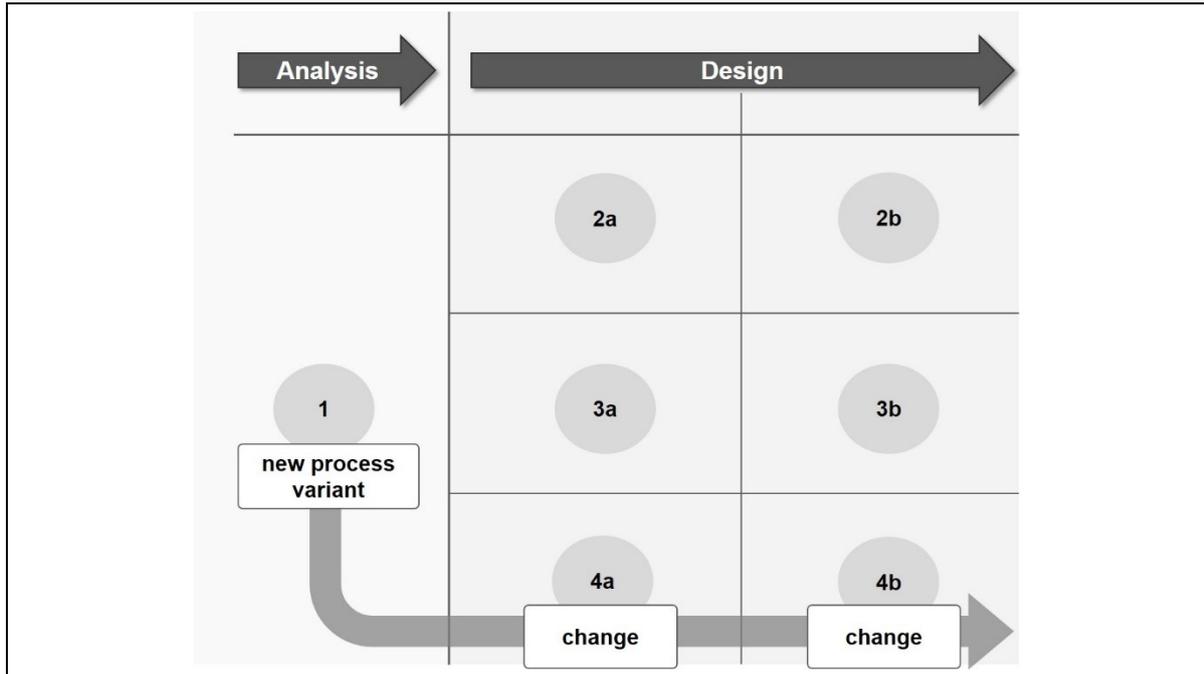


Figure 8-7: Reconfiguration process for scenario 1

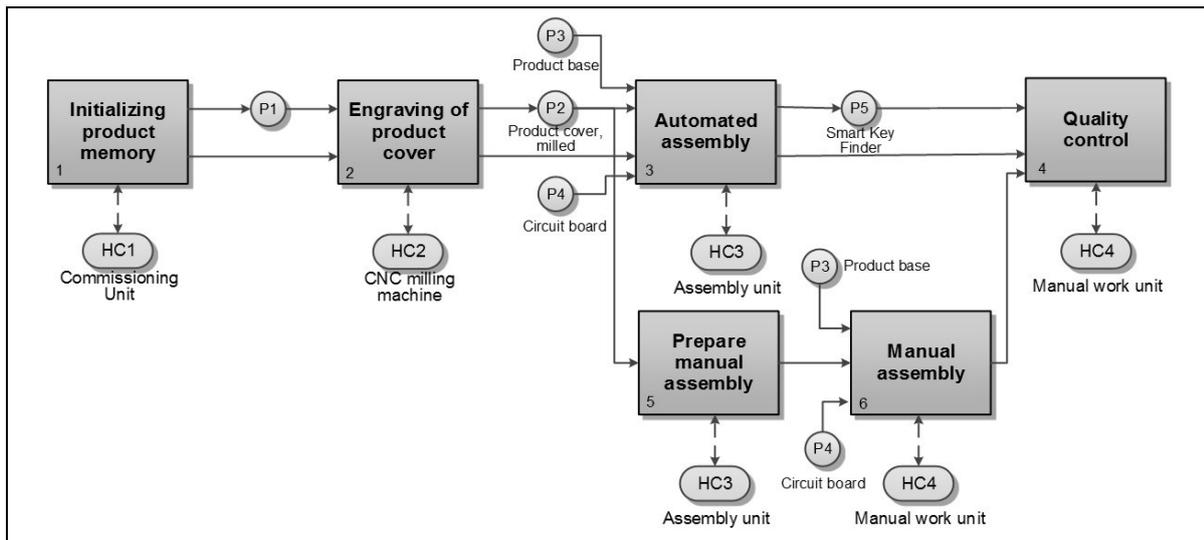


Figure 8-8: Extended process model for process variant

Reconfiguration 2: Introduction of new functionality

One of the assembly modules is extended by an additional function to verify that the assembly of the product is executed correctly. Therefore, a new operation “qualityCheck” is added to the existing composed service “AssemblyPneumatic2” (see Appendix I.2). Since the set of operations then differs from the “AssemblyPneumatic1” service, the service is renamed to “AssemblyPneumaticQC1”. First, the new functionality needs to be reflected within the service model and then the impact on the process model is determined (see Figure 8-9). For that reason, the sub process “Automated Assembly” is extended by a

process step “Check quality” (see Figure 8-10). It is decided that the required check can be executed after the process step “Assemble” by verifying the height of the assembled product. If the assembly is not executed successfully the detected height of the product cover would be either too low or too high.

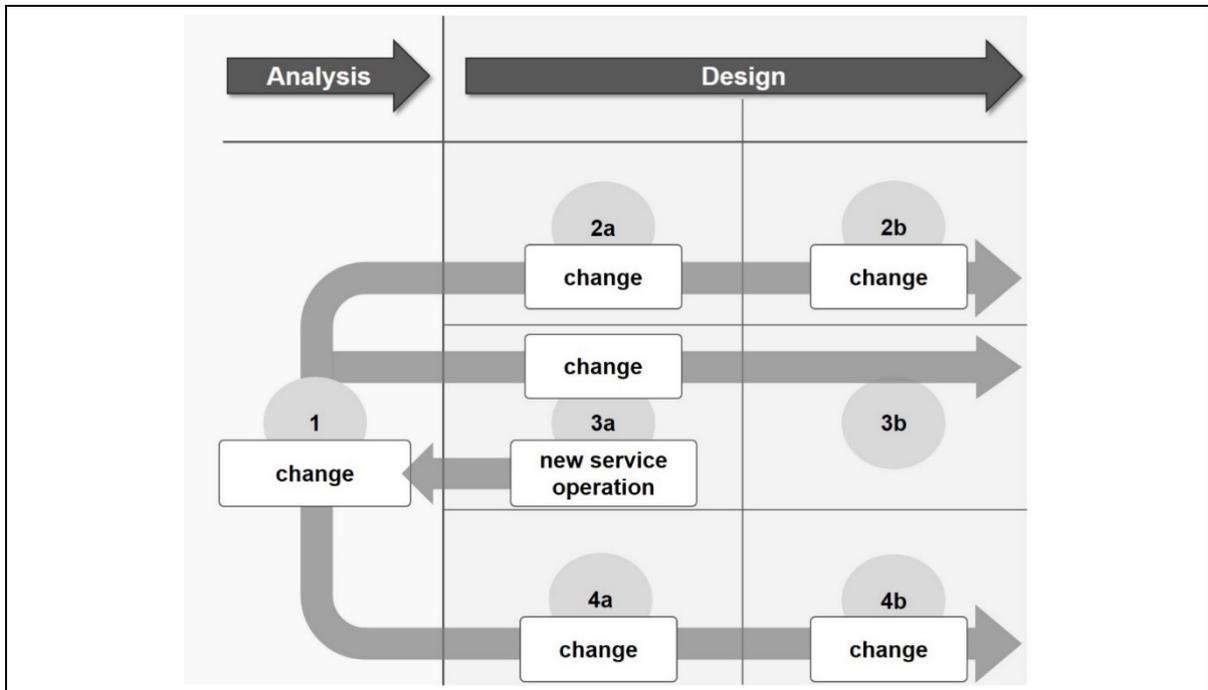


Figure 8-9: Reconfiguration process for scenario 2

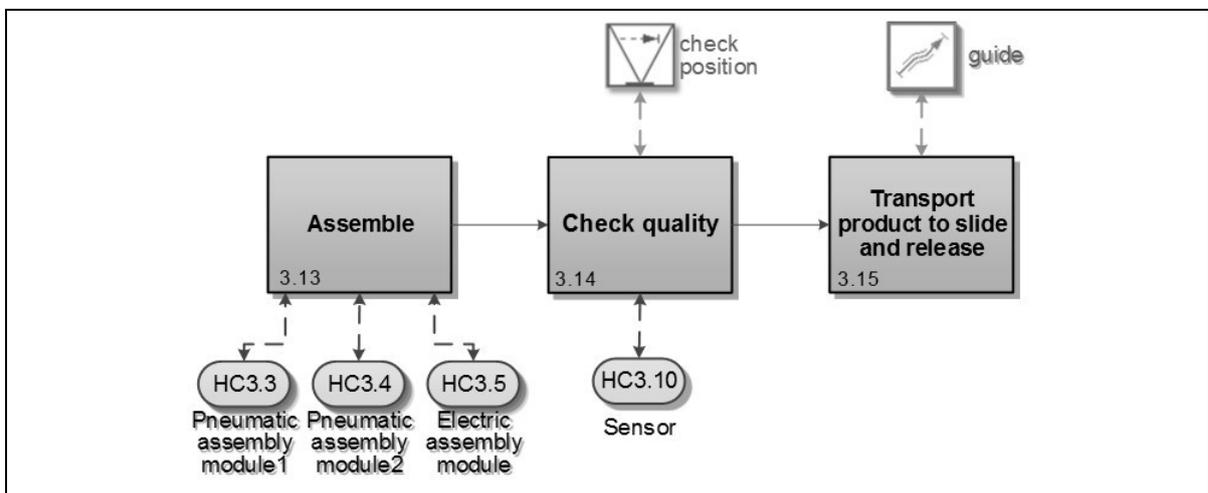


Figure 8-10: Additional process step in sub process “Automatic Assembly”

After that, the required changes to the other models are considered (see Figure 8-9). Since the existing equipment is not capable of this particular function, a new basic service and a new field device need to be introduced. For the equipment model the new field device “InductiveProximitySensor” is added to the module “PneumaticAssemblyModulePnP” which comes from the equipment supplier Pepperl & Fuchs (see Appendix F.4). The service

model is extended by the basic service “DetectInductive3” which is used by the modified composed service “AssemblyPneumaticQC1” (see Appendix I.2). The control logic model is changed in accordance with the process model by adding a process step where the new service operation is called (see Appendix I.3). Within the detailing of the concrete control logic model, it is decided that the results of the quality check are send to the order client.

Reconfiguration 3: Replacement of field device

The last scenario covers the replacement of a field device in case of a failure and the exact type of field device is not available any more. Thus, the initial change is made in the concrete equipment model where the respective hardware component template is updated (see Figure 8-12). Since the internal control logic of an equipment service depends on the respective device, the service implementation has to be exchanged in any case. This needs to be adequately considered within the technical specification and implementation phases.

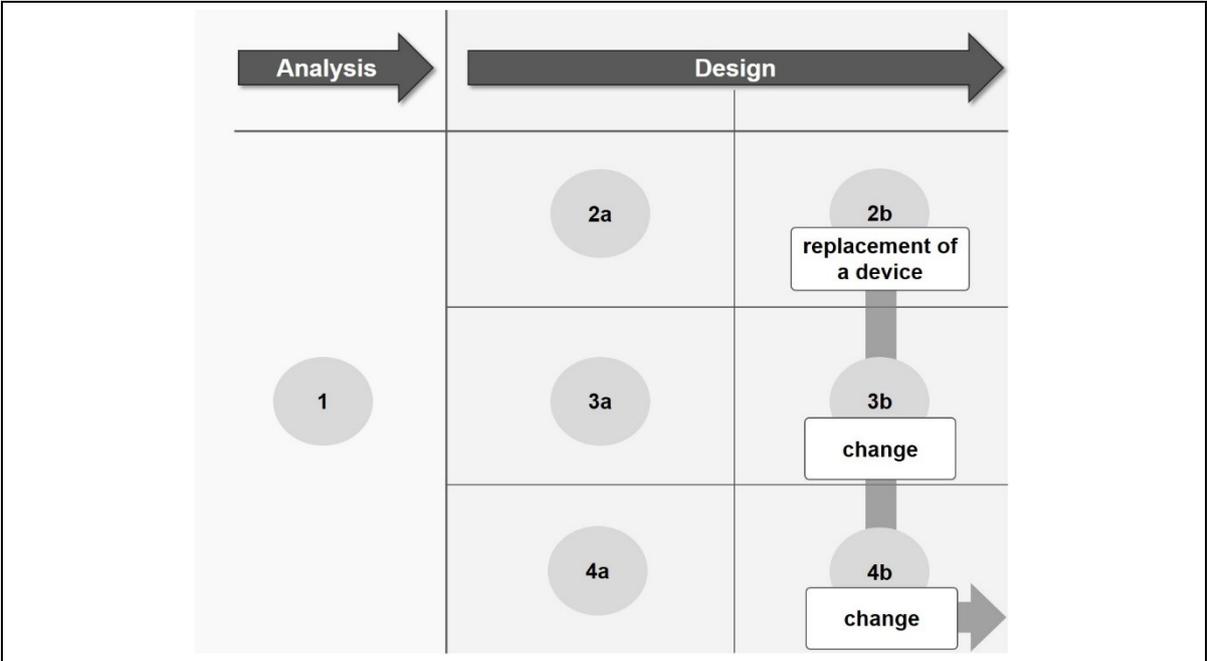


Figure 8-11: Reconfiguration process for scenario 3

If other changes within the design models are required, depends on the service of the obsolete device. In case the service can be reused by the new device with the same service operations, no further changes are required within the models. For example, both devices are inductive sensors and have the service “detectInductive”. The service description then remains the same and thus, also the service interface.

If the service of the obsolete device is device-specific, for example as “detectInductiveP+F”, then the service needs to be updated within the service model (see Figure 8-11). In this case the service interface is updated to another device-specific service which provides slightly different operations. For this example, the new device comes from the supplier Sick

so that the new service would be “detectInductiveSick”. If the service is not available yet in the service library, it must be created first. Moreover, the service calls to this modified service have to be updated within the control logic model. This example illustrates the disadvantage of device-specific services, which lead to higher dependencies between control programs and the field devices and thus, to higher programming and software management efforts.

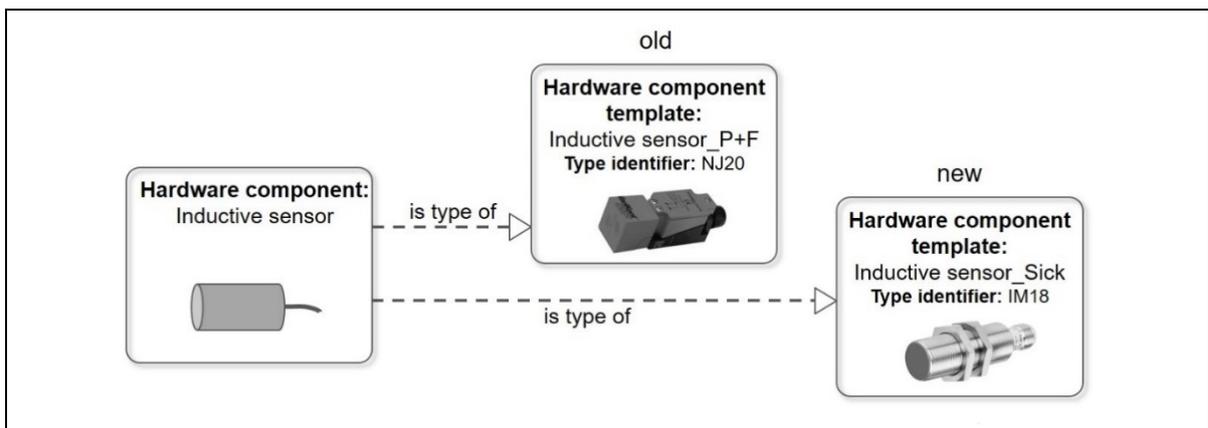


Figure 8-12: Replacement of an inductive sensor [FigPF01][FigSick01]

Similar changes are needed if the new service comes with a new operating principle or uses a different form of energy. In the given example, this would be the case if the inductive sensor is replaced by a light barrier so that the service is replaced by “detectOptic”.

8.3 Evaluation

For an evaluation of the new engineering methodology the efficiency of the control engineering is assessed based on the application scenarios (see Chapter 8.2). Therefore, the three factors for the efficiency of engineering processes (see Chapter 2.3.2) are investigated for today’s conventional control engineering (see Chapter 2.3.4) and for the new methodology MDE for SOA-AT.

Quality Assurance

Today, control procedures are usually developed from bottom-up without a concrete formulation of the requirements or high-level design of the control logic. The quality is indirectly verified during commissioning when the control programs are tested at the production equipment. Methods to check and assure the quality during the development are still rarely used.

MDE for SOA-AT supports a seamless engineering flow inspired by Systems Engineering principles. It concentrates on the left leg of the V-model and doesn’t yet include any verification methods for linking the integration of the system back to the planning phases (see Chapter 6.5.2). However, the foundation for a systematic quality assurance is already

provided by the planning models that constitute a detailed documentation of the system planning. Thus, the properties of the system can be compared with its specified design. If a requirement is not fulfilled or a property is not satisfactory, it can be tracked which requirement led to which function, service, or hardware component to specifically improve the design.

Development time

The total development time depends on the sequence of the individual planning tasks. For an assessment, the development planning of the assembly unit with both procedures is investigated. During the initial development, the time schedule has been tracked and took in total 23 weeks (see Figure 8-13). It could be observed that the engineering domains have been executed partially in parallel but generally according to the traditional sequence (see Chapter 2.3.2). The hardware design first focused completely on the mechanical design items and detailed the electrical items afterwards. Thus, the electrical realization also started two weeks after the mechanical realization tasks. After the hardware design was finished the control engineering has been executed last. Due to delays in the mechanical and electrical design phases, the control engineering has started later than planned so that the programming was still ongoing during the commissioning.

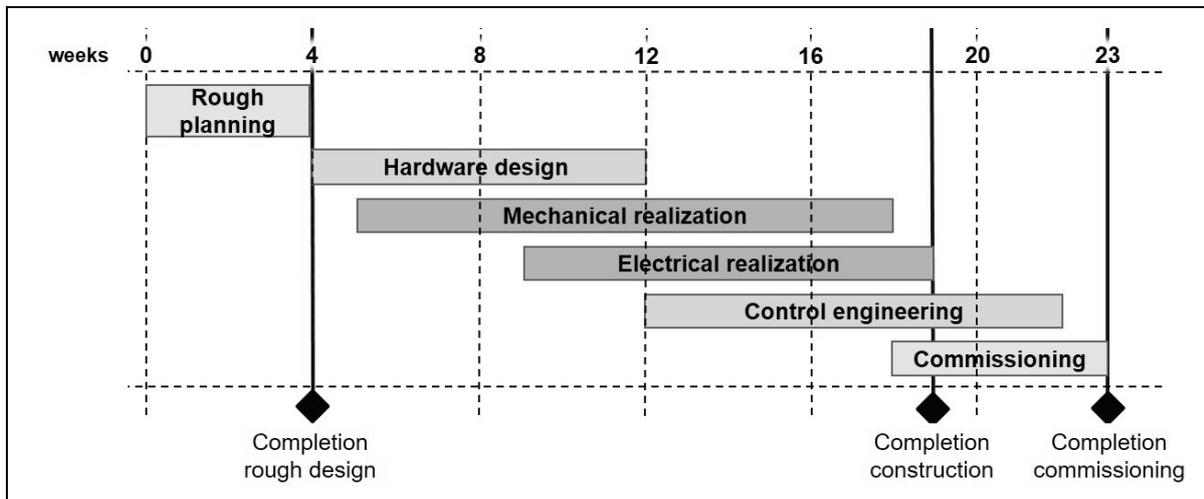


Figure 8-13: Development planning with CE

For the assessment of the duration for applying MDE for SOA-AT, the efforts for the individual tasks is assumed to be the same (see next section “Efforts” below for further investigation). The main difference here is the earlier start of the control engineering which runs in parallel with the hardware design (see Figure 8-14). The electrical design can run in parallel to the mechanical design as soon as the equipment model and service model have been developed. It is assumed that the realization phase starts a bit later due to the fact that the production system is first planned in an abstract way. However, this gets compensated by a greater parallel execution of the mechanical and electrical realization.

Since the control engineering is advanced, the commissioning including the refinement of the control software can be executed in parallel to the construction. An additional week of commissioning is assumed after all constructions tasks are done to verify the functionality of the complete system. In this case the time saving would be three weeks or 13% of the total duration of the production engineering. Consequently, MDE for SOA-AT can be considered as an enabler for shorter development timings and thus, an accelerator of time-to-market.

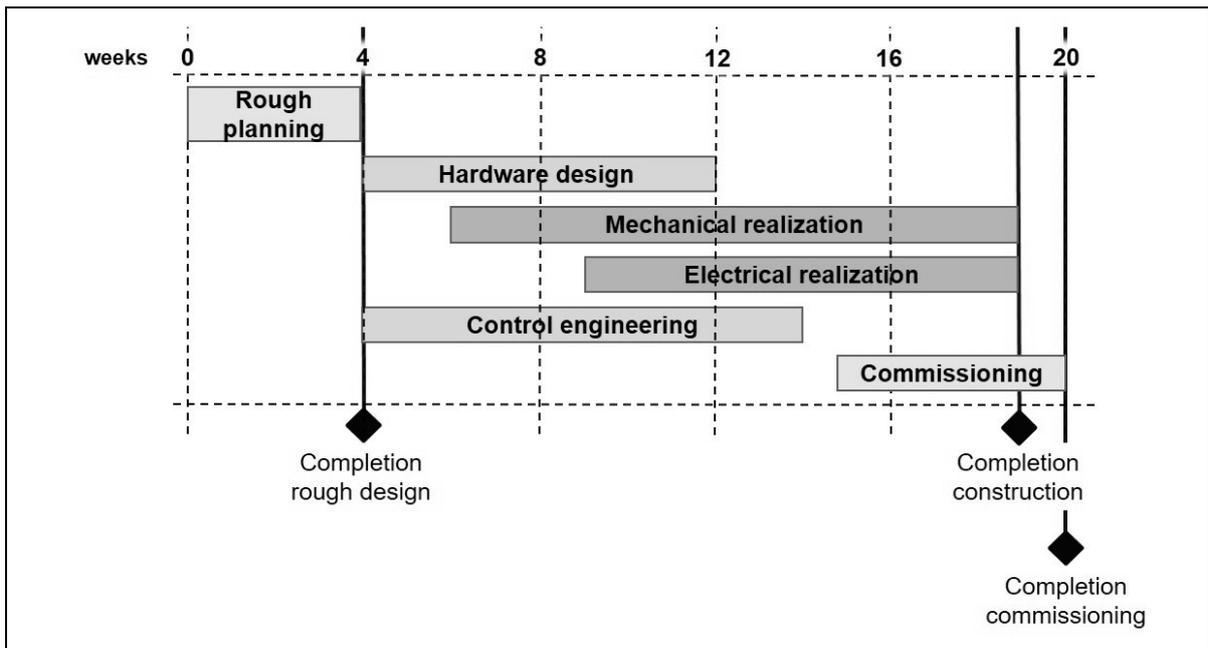


Figure 8-14: Development planning with MDE

Efforts

The exact efforts required for MDE for SOA-AT depend on several influencing factors. These can cause both higher and lower efforts compared to today's conventional control engineering procedure depending on the respective planning case.

Additional efforts:

- **Generation and maintenance of models:** The planning models need to be initially created and afterwards maintained to reflect each change that is made during reconfigurations or other tasks.
- **Development of a service library and device catalog:** The elements of the service library and device catalog need to be initially created before they can be used.
- **Definition and implementation of new services:** If a function cannot be realized by existing services of the service library, a new service needs to be created. This includes the service interface according to the standardized naming principles (see Chapter 7.1.2) and the implementation of the service in software.

Reduced efforts:

- **More efficient engineering:** The top-down planning approach enables the seamless transfer of the planning results from the early planning phases to the control engineering. Moreover, dependencies of planning objects between different engineering domains are pointed out within the models. This prevents multiple development of the same information and contradicting planning results leading to reduced engineering efforts.
- **Modular process logic:** The process logic is implemented as service-orchestration whose rough structure can be derived from early process planning results. The significant reduction of low-level programming and the derivation of a rough control logic reduces the required efforts to develop the control services. Furthermore, modular program structures have a higher degree of adaptability and makes the complexity more manageable which supports the reconfiguration of control programs.
- **Less commissioning efforts:** Since MDE for SOA-AT enables an early start with the control engineering, it is expected that the control software has a higher degree of maturity when commissioning starts. This leads to less efforts for troubleshooting or further development of the control software during commissioning.
- **Reuse of control software:** Once a service has been defined and put into the service library, it can be used for any other application. If the same technical realization is chosen, the services implementation can also be reused so that programming efforts are significantly reduced.
- **Optimized execution of reconfiguration tasks:** The required modifications for a reconfiguration task can be exactly derived from the models. Additionally, changes to the control logic can be easier implemented by means of the SOA principles.
- **Simplified testing and troubleshooting:** The comprehensibility of control software is considerably risen by the planning models providing a precise documentation. Moreover, the decomposition of control logic into several services enables a better handling of complexity. Both aspects help to apply testing procedures and to identify and fix problems.

A quantitative comparison of the efforts for a certain engineering task between both control engineering approaches is just possible when the outer circumstances are clear and all influencing factors can be considered. However, a universal assessment can be made based on the previous analysis of factors for additional and reduced efforts for MDE for SOA-AT and their classification relative to the life cycle of a production line. Generally, it can be ascertained that the factors for additional efforts are mainly tasks that occur onetime events in the beginning. This particularly concerns the generation of the models during the development planning and the introduction of the service library and the device catalog. Thus, the initial efforts for MDE for SOA-AT are estimated as being higher than for conventional engineering due the greater significance of the additional efforts (see Figure

8-15). However, if library concepts are consequently maintained and the software of the implemented services can be optimally reused from previous projects even these initial efforts can be lower.

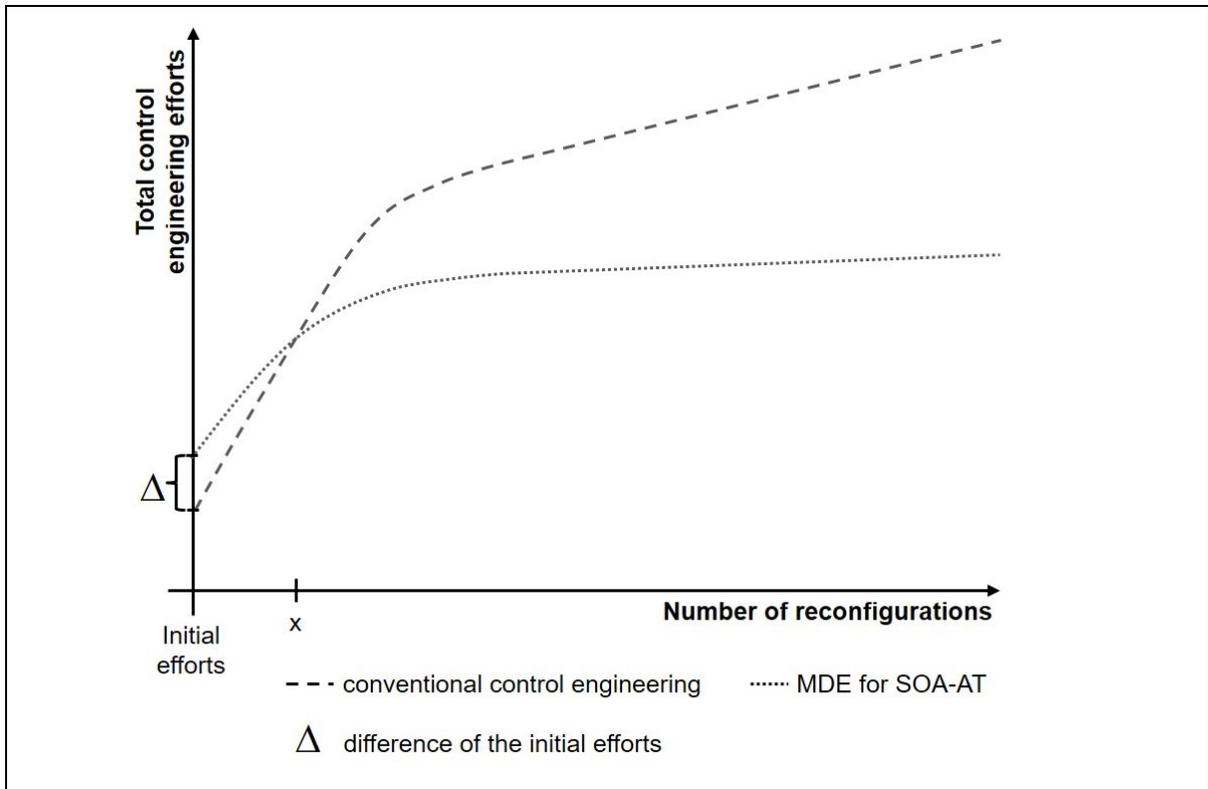


Figure 8-15: General comparison of engineering efforts

The overall engineering efforts over the complete life cycle heavily depend if and how frequently reconfigurations are executed. The effect of lower efforts by reuse is noticeable for both approaches. For MDE for SOAT-AT the additional efforts for ongoing reconfigurations is estimated as significantly lower, since reconfiguration tasks can be executed more efficiently. Thus, the curve representing the efforts over time is still increasing but it is estimated as considerably flatter than the effort curve for the conventional control engineering. In total, it is expected that in most cases the initial efforts for MDE for SOAT-AT are higher but that the total effort becomes lower than for the conventional control engineering after a number x of reconfigurations (see Figure 8-15).

9 Conclusions and Outlook

9.1 Conclusions

The core result of this thesis is the development of a model-driven engineering methodology for service-oriented control procedures to overcome certain drawbacks of today's typical control engineering process. The adoption of the SOA paradigm for manufacturing control, called SOA-AT, enables the design of control procedures as modular software architectures which generate high-level control functionality by the composition of elementary automation functions. Transferring general SOA design principles to control engineering additionally provides key principles for a systematic design process of these control architectures. This comprises first of all the process-oriented development approach which permits a top-down engineering procedure of the service structure and the control logic. A specific feature of SOA-AT is the dependency of services from the production equipment so that this particularly needs to be reflected as additional design aspect.

To develop an efficient and user-friendly methodology, the SOA design principles are combined with other proven concepts from software engineering. Object-orientation helps to define two concretization steps during the design phase and promotes reusability by libraries for planning objects. Furthermore, modeling facilitates a seamless and comprehensible design by representing planning results in a graphical way and pointing out dependencies between planning objects. All planning results are depicted as models whose modeling rules are specified within metamodels for the individual domains. SOA can be seen as enabler to effectively apply these OO and MDE principles by permitting a higher level of abstraction for the design of software.

The outcome of these considerations is the methodology MDE for SOA-AT. It is formulated as theoretical concept so that it can be applied independently on specific modeling languages, software tools, or operating systems. To show its practical relevance and to assess its effects on control engineering, an implementation concept is elaborated and applied to a real use case. Several application scenarios are investigated whose results constitute the basis to evaluate the outcome of this thesis against the initial targets (see Chapter 5.2).

Generally, the basic principles of MDE for SOA-AT are modularization and abstraction to promote reusability, comprehensibility, and adaptability of control software. A modular structuring of control procedures and the generation of high-level control functionality by means of composition enables the reduction of programming efforts. This is particularly enabled by the consequent reuse of control software. The prerequisite for a high degree of

reusability of services is a standardized naming concept and a systemic use of a service library. Although existing automation standards can serve as a base (see Chapter 7.1.2), the establishment of naming standards is challenging and the effective use of a service library heavily depending on an adequate tool support [Niss08].

Besides reusability another factor for effort reduction is a more comfortable generation and modification of control software. The decomposition into individual services according to the separation of concerns rises the abstraction degree of the process logic and therewith, the overall comprehensibility and adaptability. This permits an appropriate handling of increasingly complex control software for software engineers and users. Moreover, reconfigurations of modular software can be executed smoothly by either modifying individual services to add or change certain automation functions or by adapting the service orchestration.

The degree of adaptability is directly related to how and where the services of the designed SOA software architecture are deployed in a system architecture [Bieb05]. One aspect refers to the separation or encapsulation of hardware and software components. The realization of mechatronic components coming with hardware and software as one unit enables overall adaptive production systems whereas the implementation of the services in a separate controller enables adaptability just for the control software. Another aspect is associated with the implementation of the services in software. The proposed application concept chooses the SOA technology DPWS without focusing on non-functional control requirements like real-time and reliability. However, the services can also be implemented as PLC program when consequent modularization principles are applied (see Chapter 2.3.3). The benefits and limitations of the individual possibilities have to be carefully considered regarding the requirements of the respective application to choose the best implementation strategy.

The engineering methodology precisely describes the development steps starting with the specification of the requirements. Abstraction and stepwise concretization of control programs constitutes the enabler to close the gap between early phases of the production engineering process and the control engineering. By integrating the development methodology into early stages of the overall production engineering process and linking it to other engineering domains, an adequate consideration of control engineering according to its rising importance is enforced. Furthermore, the application of Systems Engineering principles allows to parallelize the concrete design of the individual engineering domains after the functional requirements on the whole production system are determined and dependencies between the domains are identified.

For the transfer into practice, the concepts of MDE for SOA-AT can be extended to meet the needs of specific application domains or certain company standards. The SOA-AT reference architecture can be further specified with additional service categories and even

service layers. The structure and content of the service library and device catalog can be derived from vendor catalogs or internal company libraries for devices and automation functions that especially big manufacturing companies established already. Moreover, the required design aspects can be enriched by other system properties, which need to be reflected within the modeling concepts.

The most crucial aspect for a successful application are mature implementation concepts that enable a seamless transfer of engineering results without media breaks and the instantiation of the design into a control system. The degree of how well the targets can be achieved heavily depends on whether a suitable IT infrastructure can be established to execute the MDE method and to transfer the designed service architecture into control software.

9.2 Outlook

The results of this thesis provide an engineering methodology and application concepts to improve the situation of control engineering regarding its execution and its position within the production engineering process. Further fields of work can be identified that are directly connected to these results. Several topics constitute a direct continuation of this thesis to improve the applicability of MDE for SOA-AT and to establish the required acceptance and trust for an adoption in industry:

- **Development of further control tasks:** So far MDE for SOA-AT covers the development of the control programs for the ideal process execution. However, to guarantee a robust and safe process execution, routines for the handling of fault states are required. Furthermore, software functions are needed to visualize and influence the production process for HMI applications and other uplink automation systems. An extension of the SOA-AT reference architecture by safety, HMI, SCADA, etc. services is proposed.
- **Additional functions of equipment services:** Within this thesis the definition of the equipment services concentrates on the description of the functions of field devices that are needed during run-time to execute the production process. It is conceivable that an equipment service comprises additional functions, for example for setting alarms, parametrization, testing or resetting. Moreover, functions for providing information about the device itself are considered helpful like information about the current internal state, its power consumption, and the retrieval of manuals, drawings, etc. Hereby, authorization concepts are required to regulate which functions are accessible and which data can be retrieved by certain user groups and in which state. Concepts like field device profiles and the “administration shell” defined by the Industrie 4.0 platform can help to define a standardized set of service operations for different purposes and with certain properties regarding access permissions [Plat16].

-
- **Detailed design of other engineering domains:** The PESGOP process provides the foundation for the parallel execution of the engineering domains during the detailed planning phase. Since MDE for SOA-AT targets the control engineering, development procedures for the other domains and guidelines for the coordination of interdisciplinary design questions have to be determined.
 - **Standardization of services:** The application concept of this thesis tries to apply existing standards to derive standardized names for services and service operations. However, the establishment of one general, cross-vendor, and cross-domain service standard is unrealistic. Thus, methods are needed to transfer device-specific functions directly into a user-specific library. Another step forward can be achieved by semantic descriptions of services to efficiently discover appropriate services by means of semantic technologies [Losk11].
 - **Tool chain:** In order to minimize efforts and to provide a high usability, a proper tool chain is required that supports the development from the analysis to the realization phase. The defined modeling rules need to be reflected within a design tool which ideally supports the engineer during the development steps. Furthermore, a fully seamless MDE procedure can be achieved by translating the models into software skeletons of the chosen implementation technology. A first approach has been realized where the service descriptions of the DPWS services (i.e., WSDL files) have been automatically generated based on the design model in UModel [Olli13].
 - **Implementation of designed services:** For the application concept of this thesis the realization of the designed services as distributed DPWS services on embedded devices is proposed. Since this realization concept doesn't meet the strict requirements on automation applications regarding real-time, reliability, etc. an adaption for industrial use cases is excluded. A rather conservative but much more practical realization concept uses conventional PLCs as execution platform where the designed SOA-AT system is translated into a modular PLC program. To establish more innovative implementation approaches where the services are deployed in a distributed way, a suitable migration path from today's common technologies (i.e., PLCs) is required. Within a collaboration with Phoenix Contact a prototype of a so-called "SOA-PLC" has been developed which implements DPWS services as PLC routines [Olli14].
 - **Quality Assurance:** The engineering methodology comprises the left leg of the V-model for the development of control systems in several concretization steps. An extension by quality assurance methods represented by the right leg of the V-model is logical and reasonable. Therefore, the engineering results captured in models have to be verified by means of fault analysis, test runs, or simulation and the verification results need to be fed back into the design again.

- **Education:** The new way of control engineering needs to be adapted by engineers as well as technicians. Today, PLC programmers are usually specialized electricians who have little knowledge of software engineering. On this occasion, a more holistic and interdisciplinary view on a production system needs to be taught so that engineers of all disciplines understand the connection points and dependencies between the engineering domains.

A second group of further working items considers the integration of the results into other research topics for innovative advancements in automation and production engineering:

- **Realization of distributed automation systems:** The MDE for SOA-AT methodology exclusively deals with the software design and not the system design. The benefits of a SOA-AT regarding flexibility and adaptability can be fully leveraged when the software can be flexibly deployed within a distributed automation system. Thereby, the SOA-AT services are executed on several controllers that are interconnected as proposed in the application concept (see Chapter 7.2.3). To establish distributed control systems in practice, mature methods and technologies are required to ensure critical non-functional requirements on run time behavior like performance, reliability, maintainability [Fran11].
- **Vertical integration of automation and IT systems:** By making automation functions available as encapsulated services via standardized communication interfaces, the connection of lower and higher automation levels is simplified. A promising use case for this is the coupling between automation processes and the business software system for a transparent and flexible allocation of Key Performance Indicators (KPIs) of the production process [Gerb14]. To ensure that the service execution for process control is not interrupted, access permissions for individual user groups have to be managed for each automation service.
- **Cyber-physical production systems:** The technical base for the new high-tech strategy Industrie 4.0 is the adoption of CPS for production systems (see Chapter 3.1). SOA-AT constitutes a promising approach to realize CPS on control and field level [Zühl12]. The before mentioned working items to establish distributed automation systems and the integration of SOA-AT with uplink automation and enterprise IT systems can be regarded as two key enablers to approach the realization of a production system as CPS. Moreover, an important factor constitutes the development and distribution of vendor-independent communication standards to enable the individual components of the CPS to communicate with each other independently from how and where their services are deployed [Weye15].
- **Digital Factory:** A great potential is expected by coupling SOA-AT for MDE with Digital Factory tools to execute simulations for verifying the planning results (see item “Quality assurance” above). Depending on the planning phase and the respective engineering domains other methods and tools are appropriate [VDI08]. For the MDE

for SOA-AT particularly methods to test the control logics are of interest like material flow simulation and virtual commissioning. Through the increasing use of modeling a big challenge constitutes the maintenance of the digital models. This situation can be improved by coupling the digital world and with the physical world by means of CPS to synchronize changes in both directions.

- **Organizational aspects:** The introduction of new engineering methods like MDE for SOA-AT and particularly distributed automation structures comprises, besides technical questions, also a number of organizational questions. During the complete production life cycle several different stakeholders are involved like manufacturer, plant constructors, and device suppliers. Their cooperation and responsibilities might change due to new engineering processes and deliverables. By defining encapsulated functions for production equipment and equipping the production equipment with more intelligence, an important topic constitutes also the protection of intellectual property of the individual parties.

10 Summary

In this thesis, a model-driven engineering methodology for the development of control procedures according to the SOA paradigm has been elaborated. The field of application for the methodology focusses the control engineering of discrete automated manufacturing processes. The theoretical methodology and the presented application concepts comprise all three criteria of an efficient factory planning process:

- **Modeling and structural concepts:** The general development procedure is described as reference model which defines the planning steps and the meta-models specifying how the planning information is depicted.
- **Design concepts and architectures:** Reference architectures specify structural blueprints of the models according to the scope of application, namely control procedures for manufacturing processes.
- **Procedures, methods, and tools:** An application concept defines how the theoretical concept can be applied for concrete problems by using existing standards, guidelines, modeling languages, and software tools.

First, an introduction into the topic of the thesis has been given in Chapter 1. A detailed examination of the state of the art in three chapters formed the basis for the methodology. In Chapter 2 today's situation of control engineering as being a part of the overall factory planning has been investigated. A lack of advanced software engineering methods of control software and the inadequate consideration of control engineering within the production planning have been identified as the main drawbacks of the state of the art. As a result, control programs are today typically characterized as highly complex, low-level software implementations with poor adaptability and reusability.

Chapter 3 dealt with academic concepts for distributed control systems for building modular and collaborative production systems. Three concepts have been examined: the IEC 61499, Multi-Agent Systems, and Service-oriented Architecture. Since Service-oriented Architecture already brings a methodological foundation supporting the design of software applications, methods for generally designing SOA applications and technologies for its implementation have been presented in greater detail.

Existing concepts for a more efficient control engineering have been reviewed in Chapter 4. Besides some general concepts from software engineering like object-orientation, the method of Model-driven Engineering has been introduced and applications for production automation have been examined. Moreover, some comprehensive engineering concepts and helpful engineering standards have been presented.

Based on these three chapters capturing the state of the art, the problem statement, the objective target, and the procedural method of this thesis have been specified in Chapter 5. The following three chapters comprised the content to meet the objectives set. The theory of the model-driven engineering methodology for service-oriented control procedures has been elaborated in Chapter 6. First, the basis for the methodology has been built by transferring principles from Service-oriented Architecture to the domain of production automation. After that, concepts for the design of services for manufacturing control and the PESGOP process as control engineering process have been defined whereby helpful paradigms like object-orientation and Systems Engineering have been applied. These concepts have then been integrated into a Model-driven Engineering methodology comprising the functional specification of the control system. It is described by a reference model which specifies a seamless modeling workflow with certain planning models and dedicated planning tasks. The individual planning models have been formally defined by meta-models serving as an abstract syntax to define a certain set of modeling concepts including their attributes and their relationships.

Chapter 7 defined application concepts for the theoretical methodology including a proposal for the standardized naming of planning objects and an implementation concept to transfer the planning concept into practical applications by using existing modeling languages. For the realization of service-oriented automated control architectures an innovative approach is proposed in terms of the creation of a distributed control system. It is built by mechatronic components as modularized hardware components of the production equipment with embedded controllers that deploy the services.

In Chapter 8 the proof of concept of the elaborated concepts has been given. Here, the model-driven engineering methodology has been applied for several application scenarios at an industry-typical use case. The application scenarios built the foundation for a succeeding assessment of the new methodology against the objective targets. Finally, Chapter 9 discussed the conclusions of the contents of the thesis and gave an outlook to further fields of work.

Acronyms and Abbreviations

AC:	Alternating Current
ACU:	Active Control Unit
AUT:	Automation Technology
BDI:	Belief Desire Intention
BPEL4WS:	Business Process Execution Language for Web Services
BPML:	Business Process Model Language
BPMN:	Business Process Modeling Notation
CAEX:	Computer Aided Engineering Exchange
CIM:	Computer Integrated Manufacturing
CNC:	Computerized Numerical Control
CPS:	Cyber-Physical System
CPU:	Central Processing Unit
DC:	Direct Current
DCS:	Distributed Control System
DPWS:	Devices Profile for Web Services
DIA:	Distributed Artificial Intelligence
ECC:	Execution Control Chart
EEPROM:	Electrically Erasable Programmable Read-only Memory
ERP:	Enterprise Resource Planning
FB:	Function Block
FIPA:	Foundation for Intelligent Physical Agents
FUN:	Function
GCU:	Group Control Unit
GEMMA:	Guide d'Étude des Modes de Marches et d'Arrêts
I/O:	Input/Output (relating to interfaces of a controller or program)
IL:	Instruction List
ICT:	Information and Communication Technology
IEC:	International Electrotechnical Commission
IMS:	Intelligent Manufacturing System
IPO:	Input Processing Output (relating to PLC cycle)
IT:	Information Technology

HMI:	Human Machine Interface
HMS:	Holonic Manufacturing System
HTTP:	Hypertext Transfer Protocol
IP:	Internet Protocol
JADE:	Java Agent Development Framework
KPI:	Key Performance Indicator
LAN:	Local Area Network
LD:	Ladder Diagram
MAS:	Multi-agent System
MBD:	Model-based Development
MC:	Motion Control
MDA:	Model-driven Architecture
MES:	Manufacturing Execution System
MDD:	Model-driven Development
MDE:	Model-driven Engineering
MeiA:	Methodology for Industrial Automation
MTS:	Mechatronic System
OASIS:	Organization for the Advancement of Structured Information Standards
OLE:	Object Linking and Embedding
OO:	Object-orientation
OPC:	OLE for Process Control
OPC UA:	OPC Unified Architecture
OOP:	Object-oriented Programming
PCC:	Process Control Component
PCU:	Process Control Unit
PDM:	Platform-definition Model
PESCOP:	Process-oriented Engineering for Service-oriented Control Procedures
PID:	Proportional Integral Derivative (relating to PID controller)
PIM:	Platform-independent Model
PLC:	Programmable Logic Controller
POU:	Program Organization Unit
PROG:	Program
PSM:	Platform-specific Model

RAM:	Random-access Memory
RFID:	Radio Frequency Identification
SCADA:	Supervisory Control and Data Acquisition
SCU:	Single Control Unit
SE:	Systems Engineering
SIPN:	Signal Interpreted Petri Net
SOA:	Service-oriented Architecture
SOA4D:	SOA for Devices
SOA-AT:	SOA in Production Automation
SOA-IT:	SOA for Business IT
SOPC:	Service-oriented Process Control
SysML:	Systems Modeling Language
TCP:	Transmission Control Protocol
UML:	Unified Modeling Language
UML AP:	UML Automation Profile
UML PA:	UML Process Automation
WS4D:	Web Service for Devices
WSDL:	Web Service Description Language
XMI:	XML Metadata Interchange
XML:	Extensible Markup Language

Bibliography

Books, Articles, and Web Pages

- [Abel11] Abele, E., Reinhart, G.: Zukunft der Produktion: Herausforderungen, Forschungsfelder, Chancen. Hanser, Carl, München, 2011.
- [Alva12] Alvarez, M.L., Burgos, A., Marcos, M.: GEMMA Based Approach for Generating PLCopen Automation Projects, in: Proceedings of the 1st Conference on Embedded Systems, Computational Intelligence and Telematics in Control. Würzburg, Germany, pp. 230–235, 2012.
- [Alva13] Alvarez, M.L., Estévez, E., Sarachaga, I., Burgos, A., Marcos, M.: A novel approach for supporting the development cycle of automation systems. The International Journal of Advanced Manufacturing Technology, 2013.
- [Arzé02] Arzén, K.-E., Olsson, R., Akesson, J.: Grafchart for Procedural Operator Support Tasks, in: Proceedings of the 15th IFAC World Congress. Barcelona, Spain, 2002.
- [Atki03] Atkinson, C., Kühne, T.: Model-driven development: a metamodeling foundation. IEEE Software Volume: 20, Issue: 5, pp. 36–41, 2003.
- [Babi05] Babiceanu, R.F.: Holonic-based control system for automated material handling systems, Dissertation. Virginia Polytechnic Institute, Blacksburg, Virginia, 2005.
- [Babi06] Babiceanu, R.F., Chen, F.F.: Development and Applications of Holonic Manufacturing Systems: A Survey. Journal of Intelligent Manufacturing 17, pp. 111–131, 2006.
- [Ball08] Ball, K.: How Programmable Logic Controllers Emerged from Industry Needs, URL http://www.controleng.com/index.php?id=2735&tx_ttnews%5Btt_news%5D=7552&cHash=273261 (accessed April, 17. 2016), 2008.
- [Bang08] Bangemann, T., Diedrich, C., Colombo, A.W., Karnouskos, S.: SOCRADES - Service Oriented Architecture in der Automatisierungstechnik, in: Proceedings of Automation Congress 2008. Baden-Baden, Germany, 2008.
- [Bang09] Bangemann, T., Diedrich, C., Riedl, M., Wuwer, D., Harrison, R., Monfared, R.P.: Integration of Automation Devices in Web Service supporting Systems, in: Proceedings of 30th IFAC Workshop on Real-Time Programming and 4th International Workshop on Real-Time Software. Mrągowo, Poland, 2009.
- [Beck13] Beck, A., Derksen, C., Lehnhoff, S., Linnenberg, T., Nieße, A., Rohbogner, G.: Energiesysteme und das Paradigma des Agenten, in: Agentensysteme in der Automatisierungstechnik. Springer Vieweg, Berlin, pp. 21–42, 2013.
- [Bell03] Bell, D.: UML basics: An introduction to the Unified Modeling Language, URL <http://www.ibm.com/developerworks/rational/library/769.html> (accessed April, 17. 2016), 2003.
- [Bell99] Bellifemine, F., Poggi, A., Rimassa, G.: JADE - A FIPA-compliant agent framework, in: Proceedings of the Conference on Practical Applications of Intelligent Agents. The Practical Application Company Ltd., pp. 97–108, 1999.
- [Berg06] Bergholz, M.: Objektorientierte Fabrikplanung, Dissertation. Shaker, University of Aachen, 2006.
- [Bieb05] Bieberstein, N., Bose, S., Fiammante, M., Jones, K., Shah, R.: Service-Oriented Architecture Compass: Business Value, Planning, and Enterprise Roadmap. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.
- [Bobe08] Bobek, A.: Serviceorientierte Infrastrukturen für vernetzte Dienste und eingebettete Geräte. Dissertation, University of Rostock, 2008.
- [Bock06] Bock, C.: SysML and UML 2 support for activity modeling. Systems Engineering 9, pp. 160–186, 2006.

-
- [Bohn06] Bohn, H., Bobek, A., Golatowski, F.: SIRENA - Service Infrastructure for Real-time Embedded Networked Devices: A service oriented framework for different domains, in: Proceedings of the International Conference on Networking, 2006.
- [Bohn09] Bohn, H.: Web Service Composition for Embedded Systems - WS-BPEL Extension for DPWS, Dissertation. Sierke Verlag, University of Rostock, 2009.
- [Bolt09] Bolton, W.: Programmable Logic Controllers. Newnes, 2009.
- [Bony11] Bony, B., Harnischfeger, M., Jammes, F.: Convergence of OPC UA and DPWS with a cross-domain data model, in: Proceedings of the 9th IEEE International Conference on Industrial Informatics. Lisbon, Portugal, pp. 187–192, 2011.
- [Brec11] Brecher, C. Integrative Produktionstechnik für Hochlohnländer. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.
- [Brus98] Brussel, H.V., Wyns, J., Valckenaers, P., Bongaerts, L., Peeters, P.: Reference Architecture for Holonic Manufacturing Systems: PROSA. Computers in Industry, special issue on intelligent manufacturing systems Vol. 37, pp. 255–274, 1998.
- [Când09a] Cândido, G., Barata, J., Colombo, A.W., Jammes, F.: SOA in reconfigurable supply chains: A research roadmap. Engineering Applications of Artificial Intelligence 22, pp. 939–949, 2009.
- [Când09b] Cândido, G., Jammes, F., Barata, J., Colombo, A.W.: Generic management services for DPWS-enabled devices, in: Proceedings of the 35th IEEE Conference on Industrial Electronics. IEEE, pp. 3931–3936, 2009.
- [Chri12] Christensen, J.H., Zoitl, A.: IEC 61499 - A Standard for Software Reuse in Embedded, Distributed Control Systems, URL <http://www.holobloc.com/papers/iec61499/overview.htm> (accessed April, 17. 2016), 2012.
- [Chri94] Christensen, J.H.: Holonic Manufacturing Systems: Initial Architecture and Standards Directions, in: Proceedings of the 1st European Conference on Holonic Manufacturing Systems. Hannover, Germany, 1994.
- [Clem10] Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Merson, P., Nord, R., Stafford, J.: Documenting Software Architectures: Views and Beyond, 2nd ed. Addison-Wesley Professional, 2010.
- [Coll06] Colla, M., Brusaferrri, A., Carpanzano, E.: Applying the IEC-61499 Model to the Shoe Manufacturing Sector, in: Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation. Prague, Czech Republic, pp. 1301–1308, 2006.
- [Dai12] Dai, W., Vyatkin, V.: Redesign Distributed PLC Control Systems Using IEC 61499 Function Blocks. IEEE Transactions on Automation Science and Engineering 9, pp. 390–401, 2012.
- [deDe06] de Deugd, S., Carroll, R., Kelly, K.E., Millett, B., Ricker, J.: SODA: Service Oriented Device Architecture. IEEE Pervasive Computing 5, pp. 94–96, 2006.
- [deSo10] de Sousa, M.: Analyzing the compatibility between ISA 88 and IEC 61499, in: Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation. Bilbao, Spain, pp. 1–8, 2010.
- [Dels12] Delsing, J., Rosenqvist, F., Carlsson, O., Colombo, A.W.: Migration of industrial process control systems into service oriented architecture, in: Proceedings of the 38th Annual Conference of the IEEE Industrial Electronics Society. Montreal, Canada, 2012.
- [Died08] Diedrich, C., Mühlhause, M., Riedl, M., Bangemann, T.: Mapping of smart field device profiles to web services, in: Proceedings of the IEEE International Workshop on Factory Communication Systems. Dresden, Germany, pp. 375–381, 2008.
- [Digi16] Digi.com: Digi Connect ME 9210 Specification, URL <http://www.digi.com/products/embedded-systems/system-on-modules/digiconnectme9210#specifications> (accessed April, 17. 2016), 2016.

- [Dohn10] Dohndorf, O., Kruger, J., Krumm, H., Fiehe, C., Litvina, A., Luck, I., Stewing, F.-J.: Towards the Web of Things: Using DPWS to bridge isolated OSGi platforms, in: Proceedings of the 8th IEEE International Conference on Pervasive Computing and Communications Workshops. Mannheim, Germany, pp. 720–725, 2010.
- [Drat09] Drath, R.: Datenaustausch in der Anlagenplanung mit Automationml: Integration Von Caex, Plcopen Xml und Collada. Springer DE, 2009.
- [Drat11] Drath, R., Barth, M.: Concept for interoperability between independent engineering tools of heterogeneous disciplines, in: Proceedings of the IEEE 16th Conference on Emerging Technologies & Factory Automation. Toulouse, France, pp. 1–8, 2011.
- [Dub05] Dubinin, V., Vyatkin, V., Pfeiffer, T.: Engineering of Validatable Automation Systems Based on an Extension of UML Combined With Function Blocks of IEC 61499, in: Proceedings of the 2005 IEEE International Conference on Robotics and Automation. pp. 3996–4001, 2005.
- [eCl@14] eCl@ass e.V.: eCl@ass Version 9.0 - Classification and product description, URL <http://www.eclass.de/eclasscontent/standard/overview.html.en> (accessed April, 17. 2016), 2014.
- [EIMa05] ElMaraghy, H.A.: Flexible and reconfigurable manufacturing systems paradigms. International Journal of Flexible Manufacturing Systems 17, pp. 261–276, 2005.
- [Epl10] Eppel, U.: Model Driven Engineering in Operative Industrial Process Control Environments, in: Graph Transformations and Model-Driven Engineering, Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 749–765, 2010.
- [Erl05] Erl, T.: Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall International, 2005.
- [Esté12] Estévez, E., Marcos, M.: Model-Based Validation of Industrial Control Systems. IEEE Transactions on Industrial Informatics 8, pp. 302–310, 2012.
- [Fant 11] Fantuzzi, Cesare: A Design Pattern for Translating UML Software Models into IEC 61131-3 Programming Languages, in: Proceedings of the 18th IFAC World Congress. Milan, Italy, pp. 9158–9163, 2011.
- [Fari08] Farid, A.M., McFarlane, D.C.: Production degrees of freedom as manufacturing system reconfiguration potential measures. Journal of Engineering Manufacture 222, pp. 1301–1314, 2008.
- [Favr06] Favre-Bulle, B., Zoitl, A., Dutzler, C.: Zukunftstrends der Automatisierungstechnik und die Bedeutung von adaptiven Produktionssystemen. Elektrotechnik und Informationstechnik 123, pp. 172–177, 2006.
- [Fay09] Fay, A., Schleipen, M., Mühlhause, M.: Wie kann man den Engineering-Prozess systematisch verbessern? atp - Automatisierungstechnische Praxis, pp. 80–85, 2009.
- [Fay12] Fay, A., Zimmer, F., Eckert, K., Drath, R., Barth, M.: Bewertung der Fähigkeit von Engineering-Werkzeugen zur Interoperabilität mit Hilfe einer Offenheitsmetrik, in: Proceedings of Congress AUTOMATION 2012. VDI Verlag, Baden-Baden, Germany, 2012.
- [Feld09] Feldhorst, S., Libert, S., ten Hompel, M., Krumm, H.: Integration of a legacy automation system into a SOA for devices, in: Proceedings of the IEEE Conference on Emerging Technologies Factory Automation. Presented at the ETFA 2009, Mallorca Spain, pp. 1–8, 2009.
- [Fell09] Felleisen, M., Ulrich, A., Fay, A., Enste, U., Polke, B.: Formalisierte Prozessbeschreibung in der praktischen Anwendung. atp edition 51, pp. 46–51, 2009.
- [Ferr05] Ferrarini, L., Veber, C.: Design and implementation of distributed hierarchical automation and control systems with IEC 61499, in: Proceedings of the 3rd IEEE International Conference on Industrial Informatics. Perth, Australia, pp. 74–79, 2005.
- [Fons02] Fonseca, S.P., Griss, M.L., Letsinger, R.: Agent behavior architectures a MAS framework comparison, in: Proceedings of the 1st International Joint Conference on

-
- Autonomous Agents and Multiagent Systems, AAMAS '02. ACM, New York, NY, USA, pp. 86–87, 2002.
- [Frag09] Frager, O., Nehr, W.: Modularität und Wiederverwendung im Engineering des Maschinen- und Anlagenbaus. atp edition 51, pp. 64–71, 2009.
- [Fran11] Frank, T., Merz, M., Eckert, K., Hadlich, T., Vogel-Heuser, B., Fay, A., Diedrich, C.: Dealing with non-functional requirements in distributed control systems engineering, in: Proceedings of the 16th International Conference on Emerging Technologies & Factory Automation. Toulouse, France, pp. 1–4, 2011.
- [Fran13] Frank, T., Schütz, D., Vogel-Heuser, B.: Funktionaler Anwendungsentwurf für agentenbasierte, verteilte Automatisierungssysteme, in: Agentensysteme in der Automatisierungstechnik, Xpert.press. Springer Berlin Heidelberg, pp. 3–19, 2013.
- [Frey02] Frey, G.: Design and formal Analysis of Petri Net based Logic Control Algorithms - Entwurf und formale Analyse Petrinetz-basierter Steuerungsalgorithmen, Dissertation. Shaker Verlag, University of Kaiserslautern, 2002.
- [Frey06] Frey, G., Wagner, F.: A toolbox for the development of logic controllers using Petri nets, in: Proceedings of 8th International Workshop on Discrete Event Systems. Ann Arbor, USA, pp. 473–474, 2006.
- [Frey11] Frey, G., Thramboulidis, K.: Einbindung der IEC 61131 in modellgetriebene Entwicklungsprozesse, in: Proceedings of AUTOMATION 2011. VDI, Baden-Baden, Germany, pp. 21–24, 2011.
- [Frie09] Friedrich, A.D.: Anwendbarkeit von Methoden und Werkzeugen des konventionellen Softwareengineering zur Modellierung und Programmierung von Steuerungssystemen, Dissertation. kassel university press GmbH, University of Kassel, 2009.
- [Frie11] Friedenthal, S., Moore, A., Steiner, R.: A Practical Guide to SysML: The Systems Modeling Language. Elsevier, 2011.
- [Gals08] Galster, M., Bucherer, E.: A Taxonomy for Identifying and Specifying Non-Functional Requirements in Service-Oriented Development, in: Proceedings of the IEEE Congress on Services. Honolulu, USA, pp. 345–352, 2008.
- [Gerb08] Gerber, C., Hanisch, H.-M., Ebbinghaus, S.: From IEC 61131 to IEC 61499 for distributed systems: a case study. EURASIP Journal on Embedded Syst. 2008, pp. 4:1–4:8, 2008.
- [Gerb14] Gerber, T.: Methodology for a Flexible and Transparent Allocation of Production Process KPI to Business Software Systems, Dissertation. University of Kaiserslautern, 2014.
- [Grob08] Groba, C., Braun, I., Springer, T., Wollschlaeger, M.: A service-oriented approach for increasing flexibility in manufacturing, in: Proceedings of the IEEE International Workshop on Factory Communication Systems. Dresden, Germany, pp. 415–422, 2008.
- [Gütt08] Güttel, K., Weber, P., Fay, A.: Automatic generation of PLC code beyond the nominal sequence, in: Proceedings of the IEEE International Conference on Emerging Technologies and Factory Automation. Hamburg, Germany, pp. 1277–1284, 2008.
- [Hani09] Hanisch, H.-M., Hirsch, M., Missal, D., Preuße, S., Gerber, C.: One Decade of IEC 61499 Modeling and Verification - Results and Open Issues, in: Proceedings of the 13th IFAC Symposium on Information Control Problems in Manufacturing. Moscow, Russia, pp. 211–216, 2009.
- [Häst11] Hästbacka, D., Vepsäläinen, T., Kuikka, S.: Model-driven development of industrial process control applications. Journal of Systems and Software 84, pp. 1100–1113, 2011.
- [Hegn08] Hegny, I., Hummer, O., Zoitl, A., Koppensteiner, G., Merdan, M.: Integrating software agents and IEC 61499 realtime control for reconfigurable distributed manufacturing

- systems, in: Proceedings of the International Symposium on Industrial Embedded Systems. Montpellier, France, pp. 249–252, 2008.
- [Helb10] Helbing, K.W., Mund, H., Reichel, M.: Handbuch Fabrikprojektierung, 1st ed. Springer Berlin Heidelberg, 2010.
- [Henn10] Hennig, S., Koycheva, E., Braune, A.: Domänenspezifische Sprachen und deren Bedeutung für die modellgetriebene Softwareentwicklung in der Automatisierungstechnik, in: Proceedings of AUTOMATION 2010. VDI, Baden-Baden, Germany, 2010.
- [Herm15] Hermann, M., Pentek, T., Otto, B.: Design principles for Industrie 4.0 scenarios: a literature review, URL http://www.snom.mb.tu-dortmund.de/cms/de/forschung/Arbeitsberichte/Design-Principles-for-Industrie-4_0-Scenarios.pdf (accessed April, 17. 2016), 2015.
- [Heut07] Heutschi, R.: Serviceorientierte Architektur: Architekturprinzipien und Umsetzung in Die Praxis. Springer-Verlag Berlin Heidelberg, 2007.
- [Holz07] Holzbaur, U.: Entwicklungsmanagement: Mit hervorragenden Produkten zum Markterfolg. Springer, Berlin, 2007.
- [Hovs06] Hovsepyan, A., Baelen, S.V., Vanhooff, B., Joosen, W., Berbers, Y.: Key Research Challenges for Successfully Applying MDD Within Real-Time Embedded Software Development, in: Embedded Computer Systems: Architectures, Modeling, and Simulation, Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 49–58, 2006.
- [Huss05] Hussain, T., Frey, G.: Migration of a PLC Controller to an IEC 61499 Compliant Distributed Control System: Hands-on Experiences, in: Proceedings of the 12th IEEE International Conference on Robotics and Automation. Seattle, USA, pp. 3984–3989, 2005.
- [Huss06] Hussain, T., Frey, G.: UML-based Development Process for IEC 61499 with Automatic Test-case Generation, in: Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation. Prague, Czech Republic, pp. 1277–1284, 2006.
- [Huss09] Hussain, T.: Development and Automatic Deployment of Distributed Control Applications, 1st ed, Dissertation. Shaker, University of Kaiserslautern, 2009.
- [INCO15] INCOSE: What is Systems Engineering, URL <http://www.incose.org/AboutSE/WhatIsSE> (accessed April, 17. 2016), 2015.
- [Ivan09] Ivanova, D., Batchkova, I., Panjaitan, S., Wagner, F., Frey, G.: Combining IEC 61499 and ISA S88 for Batch Control, in: Proceedings of the 13th IFAC Symposium on Information Control Problems in Manufacturing. Moscow, Russia, pp. 187–192, 2009.
- [Jamm05a] Jammes, F., Mensch, A., Smit, H.: Service-oriented device communications using the Devices Profile for Web Services, in: Proceedings of the 3rd International Workshop on Middleware for Pervasive and Ad-Hoc Computing. ACM, Grenoble, France, pp. 1–8, 2005.
- [Jamm05b] Jammes, F., Smit, H.: Service-oriented paradigms in industrial automation. IEEE Transactions on Industrial Informatics 1, pp. 62–70, 2005.
- [Jamm05c] Jammes, F., Smit, H., Lastra, J.L.M., Delamer, I.M.: Orchestration of Service-oriented Manufacturing Processes, in: Proceedings of the IEEE International Conference on Emerging Technologies in Factory Automation. Cernobio, Italy, pp. 617–624, 2005.
- [Jamm14] Jammes, F., Karnouskos, S., Bony, B., Nappey, P., Colombo, A.W., Delsing, J., Eliasson, J., Kyusakov, R., Stluka, P., Tilly, M., Bangemann, T.: Promising Technologies for SOA-Based Industrial Automation Systems, in: Industrial Cloud-Based Cyber-Physical Systems. Springer International Publishing, pp. 89–109, 2014.
- [Jenn98] Jennings, N.R., Wooldridge, M.: Applications of Intelligent Agents, in: Agent Technology. Springer Berlin Heidelberg, pp. 3–28, 1998.

-
- [John98] Johnsson, C., Arzén, K.-E.: On batch recipe structures using High-Level Grafchart, in: Preprints of Reglermöte '98. Lund Institute of Technology, Lund, Sweden, 1998.
- [John99] Johnsson, C.: A Graphical Language for Batch Control, Dissertation. University of Lund, 1999.
- [John08] Johnsson, C.: Graphical Languages for Business Processes and Manufacturing Operations, in: Proceedings of the 17th IFAC World Congress. Seoul, South Korea, pp. 13863–13868, 2008.
- [Kage13] Kagermann, H., Wahlster, W., Helbig, J.: Recommendations for implementing the strategic initiative INDUSTRIE 4.0. Report of the Communication Promoters Group of the Industry-Science Research Alliance, 2013.
- [Karn07] Karnouskos, S., Baecker, O., de Souza, L.M.S., Spiess, P.: Integration of SOA-ready networked embedded devices in enterprise systems via a cross-layered web service infrastructure, in: Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation. Patras, Greece, pp. 293–300, 2007.
- [Karn10] Karnouskos, S., Colombo, A.W., Jammes, F., Delsing, J., Bangemann, T.: Towards an architecture for service-oriented process monitoring and control, in: Proceedings of the 36th Annual Conference on IEEE Industrial Electronics Society. Phoenix, USA, pp. 1385–1391, 2010.
- [Katz09] Katzke, U.: Spezifikation und Anwendung einer Modellierungssprache für die Automatisierungstechnik auf Basis der Unified Modeling Language, Auflage: 1. ed, Dissertation. Kassel University Press, University of Kassel, 2009.
- [Kief06] Kiefer, J., Baer, T., Bley, H.: Mechatronic-oriented Engineering of manufacturing Systems Taking the Example of the Body Shop, in: Proceedings of the 13th CIRP International Conference on Life Cycle Engineering. Leuven, Belgium, 2006.
- [Kief08] Kiefer, J.: Mechatronikorientierte Planung automatisierter Fertigungszellen im Bereich Karosserierohbau, Dissertation. Saarland University, 2008.
- [Kirk08] Kirkham, T., Savio, D., Smit, H., Harrison, R., Monfared, R.P., Phaithoonbuathong, P.: SOA middleware and automation: Services, applications and architectures, in: Proceedings of 6th IEEE International Conference on Industrial Informatics. Daejeon, Korea, pp. 1419–1424, 2008.
- [Klein03] Klein, S., Frey, G., Minas, M.: PLC Programming with Signal Interpreted Petri Nets, in: Applications and Theory of Petri Nets, Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 440–449, 2003.
- [Kleu10] Kleuker, S.: Grundkurs Software-Engineering mit UML: Der pragmatische Weg zu erfolgreichen Softwareprojekten, 2nd ed. Vieweg+Teubner Verlag, Wiesbaden, 2010.
- [Kraf05] Krafzig, D., Banke, K., Slama, D.: Enterprise SOA: Service-Oriented Architecture Best Practices. Prentice Hall, New Jersey, USA, 2005.
- [Kys11] Kyusakov, R., Makitaavola, H., Delsing, J., Eliasson, J.: Efficient XML Interchange in factory automation systems, in: Proceedings of the 37th Annual Conference on IEEE Industrial Electronics Society. IEEE, Melbourne, Australia, pp. 4478–4483, 2011.
- [Lee15] Lee, J., Bagheri, B., Kao, H.-A.: A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems. Manufacturing Letters 3, pp. 18–23, 2015.
- [Leit04] Leitão, P.: An agile and adaptive holonic architecture for manufacturing control, Dissertation. University of Porto, 2004.
- [Leit06] Leitão, P., Colombo, A.W.: Petri net based Methodology for the Development of Collaborative Production Systems, in: Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation. Prague, Czech Republic, pp. 819–826, 2006.
- [Leit07] Leitner, S.H., Mahnke, W.: OPC UA–Service-oriented Architecture for Industrial Applications. ABB Corporate Research Center, 2007.

- [Leit08] Leitão, P., Restivo, F.J.: Implementation of a Holonic Control System in a Flexible Manufacturing System. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* 38, pp. 699–709, 2008.
- [Leit09] Leitão, P.: Agent-based distributed manufacturing control: A state-of-the-art survey. *Engineering Applications of Artificial Intelligence* 22, pp. 979–991, 2009.
- [Lepu 08] Lepuschitz, W., Zoitl, A.: An engineering method for batch process automation using a component oriented design based on IEC 61499, in: *Proceedings of the IEEE International Conference on Emerging Technologies and Factory Automation*. Hamburg, Germany, pp. 207–214, 2008.
- [Lepu11] Lepuschitz, W., Zoitl, A., Vallée, M., Merdan, M.: Toward Self-Reconfiguration of Manufacturing Systems Using Automation Agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* 41, pp. 52–69, 2011.
- [Li12] Li, B., Bayrak, G., Kernschmidt, K., Vogel-Heuser, B.: Specification of the Requirements to Support Information Technology-Cycles in the Machine and Plant Manufacturing Industry, in: *Proceedings of the 14th IFAC Symposium on Information Control Problems in Manufacturing*. Bucharest, Romania, pp. 1077–1082, 2012.
- [Losk11] Loskyll, M., Schlick, J., Hodek, S., Ollinger, L., Gerber, T., Pirvu, B.: Semantic service discovery and orchestration for manufacturing processes, in: *Proceedings of the IEEE 16th Conference on Emerging Technologies & Factory Automation*. IEEE, Toulouse, France, pp. 1–8, 2011.
- [Lüde10] Lüder, A., Foehr, L.M., Wagner, T., Zaddach, J.-J., Holm, T.: Manufacturing system engineering with mechatronical units, in: *Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation*. IEEE, Bilbao, Spain, pp. 1–8, 2010.
- [Lukm13] Lukman, T., Godena, G., Gray, J., Heričko, M., Strmčnik, S.: Model-driven engineering of process control software – beyond device-centric abstractions. *Control Engineering Practice* 21, pp. 1078–1096, 2013.
- [Luth12] Luthria, H., Rabhi, F.A.: Service-Oriented Architectures: Myth or Reality? *IEEE Software* 29, pp. 46–52, 2012.
- [Maju11] Majumdar, A., Szigeti, H.: ICT for Manufacturing - The ActionPlanT Roadmap for Manufacturing 2.0, URL http://cordis.europa.eu/result/rcn/47681_en.html (accessed April, 16.2016), 2011.
- [Marc08] Marcos, M., Estevez, E.: Model-driven design of Industrial Control Systems, in: *Proceedings of the IEEE International Conference on Computer-Aided Control Systems*. San Antonio, USA, pp. 1253–1258, 2008.
- [Math09a] Mathes, M.: Time-Constrained Web Services for Industrial Automation, Dissertation. Suedwestdeutscher Verlag fuer Hochschulschriften, University of Marburg, 2009.
- [Math09b] Mathes, M., Stoidner, C., Heinzl, S., Freisleben, B.: SOAP4PLC: Web Services for Programmable Logic Controllers, in: *Proceedings of the 17th Euromicro International Conference on Parallel, Distributed and Network-Based Processing*. Weimar, Germany, pp. 210–219, 2009.
- [Math12] MathWorks: Simulink PLC Coder (www document), URL <http://www.mathworks.de/products/sl-plc-coder/> (accessed April, 18.2016), 2012.
- [Maye10] Mayer, P., Koch, N., Schroeder, A., Knapp, A.: The UML4SOA Profile (Technical Report), LMU Muenchen, 2010.
- [Melz08] Melzer, I.: *Service-orientierte Architekturen mit Web Services: Konzepte - Standards - Praxis*, 3. Aufl. ed. Spektrum Akademischer Verlag, Heidelberg, Germany, 2008.
- [Mend08a] Mendes, J.M., Leitão, P., Colombo, A.W., Restivo, F.: Service-oriented control architecture for reconfigurable production systems, in: *Proceedings of the 6th IEEE International Conference on Industrial Informatics*. Daejeon, Korea, 2008.
- [Mend08b] Mendes, J.M., Leitão, P., Colombo, A.W., Restivo, F.: High-Level Petri Nets control modules for service-oriented devices: A case study, in: *Proceedings of the 34th*

-
- Annual Conference of IEEE Industrial Electronics. IEEE, Orlando, USA, pp. 1487–1492, 2008.
- [Mend08c] Mendes, J.M., Leitão, P., Colombo, A.W., Restivo, F.: Service-oriented process control using High-Level Petri Nets, in: Proceedings of the 6th IEEE International Conference on Industrial Informatics. Daejeon, Korea, pp. 750–755, 2008.
- [Mend09a] Mendes, J.M., Bepperling, A., Pinto, J., Leitão, P., Restivo, F., Colombo, A.W.: Software Methodologies for the Engineering of Service-Oriented Industrial Automation: The Continuum Project, in: Proceedings of the 33rd Annual IEEE International Computer Software and Applications Conference. Seattle, USA, pp. 452–459, 2009.
- [Mend09b] Mendes, J.M., Leitão, P., Restivo, F., Colombo, A.W.: Service-Oriented Agents for Collaborative Industrial Automation and Production Systems, in: Holonic and Multi-Agent Systems for Manufacturing, Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 13–24, 2009.
- [Mend11] Mendes, J.M., Leitão, P., Colombo, A.W.: Service-oriented computing in manufacturing automation: A SWOT analysis, in: Proceedings of the 9th IEEE International Conference on Industrial Informatics. Cardiff, UK, pp. 346–351, 2011.
- [Mens11] Mensch, A.: Introduction to SOA4D projects, Proceedings of the 3rd TeKoMed Workshop. Lübeck, Germany, 2011.
- [Mers10] Mersch, H., Schlutter, M., Epple, U.: Classifying services for the automation environment, in: Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation. Bilbao, Spain, pp. 1–7, 2010.
- [Mers11a] Mersch, H., Behnen, D., Schmitz, D., Epple, U., Brecher, C., Jarke, M.: Gemeinsamkeiten und Unterschiede der Prozess- und Fertigungstechnik. at - Automatisierungstechnik, pp. 7–17, 2011.
- [Mers11b] Mersch, H., Epple, U.: Requirements on distribution management for service-oriented automation systems, in: Proceedings of the 16th IEEE Conference on Emerging Technologies Factory Automation. Toulouse, France, pp. 1–8, 2011.
- [Mers12] Mersch, H., Epple, U.: Concepts of service-orientation for process control engineering, in: Proceedings of 9th International Multi-Conference on Systems, Signals and Devices. Chemnitz, Germany, pp. 1–6, 2012.
- [Minh11] Minhas, S.U.-H., Lehmann, C., Städter, J.P., Berger, U.: Reconfigurable Strategies for Manufacturing Setups to confront Mass Customization Challenges, in: Proceedings of the 21st International Conference on Production Research. Stuttgart, Germany, 2011.
- [Miss07] Missal, D., Hirsch, M., Hanisch, H.-M.: Hierarchical distributed controllers - design and verification, in: Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation. Patras, Greece, pp. 657–664, 2007.
- [Müll14] Müller, J.P., Fischer, K.: Application Impact of Multi-agent Systems and Technologies: A Survey, in: Agent-Oriented Software Engineering. Springer Berlin Heidelberg, pp. 27–53, 2014.
- [Nguy08] Nguyen, D.K., Savio, D.: Exploiting SOA for adaptive and distributed manufacturing with cross enterprise shop floor commerce, in: Proceedings of the 10th International Conference on Information Integration and Web-Based Applications & Services. New York, USA, pp. 318–323, 2008.
- [Niss08] Nissen, V., Petsch, M., Schorcht, H.: Service-orientierte Architekturen: Chancen und Herausforderungen bei der Flexibilisierung und Integration von Unternehmensprozessen. DUV, 2008.
- [Nof09] Nof, S.Y.: Handbook of Automation, 1st ed. Springer, 2009.
- [Ober15] Obermeier, M., Braun, S., Vogel-Heuser, B.: A Model-Driven Approach on Object-Oriented PLC Programming for Manufacturing Systems with Regard to Usability. IEEE Transactions on Industrial Informatics 11, pp. 790–800, 2015.

- [Olli13] Ollinger, L., Wehrmeister, M., Pereira, C., Zühlke, D.: An Integrated Concept for the Model-Driven Engineering of Distributed Automation Architectures on Embedded Systems, in: Proceedings of the IFAC Workshop on Intelligent Manufacturing Systems. São Paulo, Brazil, 2013.
- [Olli14] Ollinger, L., Abdo, A., Zühlke, D., Heutger, H.: SOA-PLC Dynamic Generation and Deployment of Web Services on a Programmable Logic Controller, in: Proceedings of the IFAC World Congress, Volume. Cape Town, South Africa, 2014.
- [Olss05] Olsson, R.: Batch Control and Diagnosis, Dissertation. Department of Automatic Control, Lund University, University of Lund, Sweden, 2005.
- [OMG05] OMG: Introduction to OMG's Unified Modeling Language, URL http://www.omg.org/gettingstarted/what_is_uml.htm (accessed April, 18. 2012), 2005.
- [Pang10] Pang, C., Vyatkin, V.: IEC 61499 function block implementation of Intelligent Mechatronic Component, in: Proceedings of the 8th IEEE International Conference on Industrial Informatics. Osaka, Japan, pp. 1124–1129, 2010.
- [Panj05] Panjaitan, S., Frey, G.: Functional design for IEC 61499 distributed control systems using UML activity diagrams, in: Proceedings of the International Conference Instrumentation, Communication and Information Technology. Bandung, Indonesia, pp. 64–70, 2005.
- [Panj06] Panjaitan, S., Frey, G.: Combination of UML Modeling and the IEC 61499 Function Block Concept for the Development of Distributed Automation Systems, in: Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation. Prague, Czech Republic, pp. 766–773, 2006.
- [Panj07] Panjaitan, S.D.: Development process for distributed automation systems based on elementary mechatronic functions, Dissertation. Shaker, University of Kaiserslautern, 2007.
- [Paol05] Paolucci, M., Sacile, R.: Agent-Based Manufacturing and Control Systems: New Agile Manufacturing Solutions for Achieving Peak Performance. CRC Press, Boca Raton, Florida, 2005.
- [Parn72] Parnas, D.L.: On the Criteria to Be Used in Decomposing Systems into Modules. *Commun. ACM* 15, pp. 1053–1058, 1972.
- [Pawe08] Pawellek, G.: Ganzheitliche Fabrikplanung: Grundlagen, Vorgehensweise, EDV-Unterstützung. Springer, Berlin; Heidelberg, 2008.
- [Pelt07] Peltola, J., Christensen, J., Sierla, S., Koskinen, K.: A Migration Path to IEC 61499 for the Batch Process Industry, in: Proceedings of the 5th IEEE International Conference on Industrial Informatics. Vienna, Austria, pp. 811–816, 2007.
- [Pere07] Pereira, C.E., Carro, L.: Distributed real-time embedded systems: Recent advances, future trends and their impact on manufacturing plant control. *Annual Reviews in Control* 31, pp. 81–92, 2007.
- [Pesc05] Peschke, J., Luder, A., Kuhnle, H.: The PABADIS'PROMISE architecture - a new approach for flexible manufacturing systems, in: Proceedings of the 10th IEEE Conference on Emerging Technologies and Factory Automation. Catania, Italy. 2005.
- [Pint09] Pinto, J., Mendes, J.M., Leitao, P., Colombo, A.W., Bepperling, A., Restivo, F.: Decision support system for Petri nets enabled automation components, in: Proceedings of the 7th IEEE International Conference on Industrial Informatics. IEEE, Cardiff, UK, pp. 289–294, 2009.
- [Plat16] Plattform Industrie 4.0: Structure of the administration shell, working paper, URL http://www.plattform-i40.de/I40/Redaktion/EN/Downloads/Publikation/structure-of-the-administration-shell.pdf?__blob=publicationFile&v=5 (accessed Sept, 02. 2016), 2016.
- [Pohl08] Pohlmann, E.G.: Methodik zur prozessorientierten Planung serviceorientierter Fabriksteuerungssysteme, Dissertation. University of Kaiserslautern, 2008.

-
- [Preu11] Preuße, S., Missal, D., Gerber, C., Hirsch, M., Hanisch, H.-M.: On the Use of Model-Based IEC 61499 Controller Design. *International Journal of Discrete Event Control Systems*, pp. 115–128, 2011.
- [Ramo10] Ramos, A.L., Ferreira, J.V.: Revisiting the similar process to engineer the contemporary systems. *Journal of Systems Science and Systems Engineering* 19, pp. 321–350, 2010.
- [Rein07] Reinhart, G., Wunsch, G.: Economic application of virtual commissioning to mechatronic production systems. *Production Engineering* 1, pp. 371–379, 2007.
- [Ribe08] Ribeiro, L., Barata, J., Mendes, P.: MAS and SOA: Complementary Automation Paradigms, in: *Innovation in Manufacturing Networks*, IFIP International Federation for Information Processing. Springer Boston, pp. 259–268, 2008.
- [Rita07] Ritala, T., Kuikka, S.: UML Automation Profile: Enhancing the Efficiency of Software Development in the Automation Industry, in: *Proceedings of the 5th IEEE International Conference on Industrial Informatics*. Vienna, Austria, pp. 885–890, 2007.
- [Royc70] Royce, W.W.: Managing the development of large software systems, in: *Proceedings of IEEE WESCON Conference*. Los Angeles, Los Angeles, USA, 1970.
- [Rupa10] Ruparelia, N.B.: Software Development Lifecycle Models. *SIGSOFT Software Engineering Notes* 35, pp. 8–13, 2010.
- [Schl08a] Schleipen, M.: OPC UA supporting the automated engineering of production monitoring and control systems, in: *IEEE International Conference on Emerging Technologies and Factory Automation*, 2008. ETFA 2008. Presented at the IEEE International Conference on Emerging Technologies and Factory Automation, 2008. ETFA 2008, IEEE, pp. 640–647, 2008.
- [Schl08b] Schleipen, M., Drath, R., Sauer, O.: The system-independent data exchange format CAEX for supporting an automatic configuration of a production monitoring and control system, in: *Proceedings of the IEEE International Symposium on Industrial Electronics*. Cambridge, UK, pp. 1786–1791, 2008.
- [Schm05] Schmidgall, G., Kiefer, J., Bär, T.: Objectives of integrated digital production engineering in the automotive industry, in: *Proceedings of the 16th IFAC World Congress*. Prague, Czech Republic, pp. 1457–1457, 2005.
- [Schm06] Schmidt, D.C.: Model-Driven Engineering. *IEEE Computer* 39, pp. 25–31, 2006.
- [Schn92] Schneeweiß, P.D.C.: *Methodologische Grundlagen der Planung*, in: *Planung*, Springer-Lehrbuch. Springer Berlin Heidelberg, pp. 227–264, 1992.
- [Schr11] Schreiber, S., Jerenz, S., Fay, A.: Anforderungen an Steuerungskonzepte für moderne Fertigungsanlagen - Herausforderungen für dezentrale Ansätze in der Automatisierungstechnik, in: *Proceedings of the Congress AUTOMATION 2011*. VDI Verlag, Baden-Baden, Germany, pp. 7–11, 2011.
- [Schü09] Schünemann, U.: Objektorientierung in der Anlagenentwicklung - eine Vision, in: *Automation & Embedded Systems: Effizienzsteigerung Im Engineering*, Tagungen Und Berichte. Kassel University Press, 2009.
- [Schü11] Schütz, D., Schraufstetter, M., Folmer, J., Vogel-Heuser, B., Gmeiner, T., Shea, K.: Highly Reconfigurable Production Systems Controlled by Real-Time Agents, in: *Proceedings of the 16th IEEE International Conference on Emerging Technologies & Factory Automation*. Toulouse, France, 2011.
- [Secc07] Secchi, C., Bonfe, M., Fantuzzi, C.: On the Use of UML for Modeling Mechatronic Systems. *IEEE Transactions on Automation Science and Engineering* 4, pp. 105–113, 2007.
- [Seit08] Seitz, M.: *Speicherprogrammierbare Steuerungen*. Hanser Verlag, 2008.
- [Seli03] Selic, B.: The Pragmatics of Model-Driven Development. *IEEE Software* 20, pp. 19–25, 2003.

- [Shen06] Shen, W., Hao, Q., Yoon, H.J., Norrie, D.H.: Applications of agent-based systems in intelligent manufacturing: An updated review. *Advanced Engineering Informatics* 20, pp. 415–431, 2006.
- [Siem04] Siemens AG: Siemens: Nahtabdichten nahtlos sicher. Move Up, 2004.
- [Soft06] Software Engineering Group, University of Patras: CORFU Engineering Support System, URL <http://seg.ece.upatras.gr/Corfu/dev/corfu.htm> (accessed April, 18. 2016), 2006.
- [Souz08] Souza, L., Spiess, P., Guinard, D., Köhler, M., Karnouskos, S., Savio, D.: SOCRADES: A Web Service based Shop Floor Integration Infrastructure, in: *Proceedings of the Conference on Internet of Things*. Zurich, Switzerland, pp. 50–67, 2008.
- [Spie09] Spiess, P., Karnouskos, S., Guinard, D., Savio, D., Baecker, O., Souza, L.M.S. de, Trifa, V.: SOA-Based Integration of the Internet of Things in Enterprise Services, in: *Proceedings of the IEEE International Conference on Web Services*, Washington DC, USA, pp. 968–975, 2009.
- [Sünd06] Sünder, C., Zoitl, A., Dutzler, C.: Functional structure-based modelling of automation systems. *International Journal of Manufacturing Research* 1, pp. 405–420, 2006.
- [Tabu06] Tabuada, P.: Cyber-physical systems: Position paper, in: *NSF Workshop on Cyber-Physical Systems*, 2006.
- [Tan10] Tan, V.V., Yi, M.-J.: Flexibility and Interoperability in Automation Systems by Means of Service Oriented Architecture, in: *Advanced Intelligent Computing Theories and Applications. With Aspects of Artificial Intelligence*, Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 554–563, 2010.
- [Tane06] Tanenbaum, A.S., Steen, M. van: *Distributed Systems. Principles and Paradigms*, 2nd ed. Prentice Hall International, 2006.
- [Thap09] Thapa, D., Park, C.M., Park, S.C., Wang, G.-N.: Auto-generation of IEC standard PLC code using t-MPSG. *International Journal of Control, Automation and Systems* 7, pp. 165–174, 2009.
- [Thar98] Tharumarajah, A., Wells, A.J., Nemes, L.: Comparison of emerging manufacturing concepts, in: *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*. San Diego, USA, pp. 325–331, 1998.
- [Theo13] Theorin, A., Ollinger, L., Johnsson, C.: Service-Oriented Process Control with Grafchart and the Devices Profile for Web Services, in: *Service Orientation in Holonic and Multi Agent Manufacturing and Robotics*, Studies in Computational Intelligence. Springer Berlin Heidelberg, pp. 213–228, 2013.
- [Thom12] Thomas, A., Trentesaux, D., Valckenaers, P.: Intelligent distributed production control. *Journal of Intelligent Manufacturing* 23, pp. 2507–2512, 2012.
- [Thra04] Thramboulidis, K.C., Tranoris, C.S.: Developing a CASE tool for distributed control applications. *The International Journal of Advanced Manufacturing Technology* 24, pp. 24–31, 2004.
- [Thra05] Thramboulidis, K.: Model-Integrated Mechatronics – Toward a New Paradigm in the Development of Manufacturing Systems. *IEEE Transactions on Industrial Informatics* 1, pp. 54–61, 2005.
- [Thra06] Thramboulidis, K., Koumoutsos, G., Doukas, G.: Towards a Service-Oriented IEC 61499 compliant Engineering Support Environment, in: *Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation*. Prague, Czech Republic, pp. 758–765, 2006.
- [Thra07a] Thramboulidis, K., Perdikis, D., Kantas, S.: Model driven development of distributed control applications. *The International Journal of Advanced Manufacturing Technology* 33, pp. 233–242, 2007.

-
- [Thra07b] Thramboulidis, K., Sierla, S., Papakonstantinou, N., Koskinen, K.: An IEC 61499 Based Approach for Distributed Batch Process Control, in: Proceedings of the 5th IEEE International Conference on Industrial Informatics. Patras, Greece, pp. 177–182, 2007.
- [Thra09] Thramboulidis, K.: IEC 61499 function block model: Facts and fallacies. IEEE Industrial Electronics Magazine 3, pp. 7–26, 2009.
- [Thra11a] Thramboulidis, K., Frey, G.: An MDD Process for IEC 61131-based Industrial Automation Systems, in: In Proceedings of the IEEE Conference on Emerging Technologies & Factory Automation. Toulouse, France, 2011.
- [Thra11b] Thramboulidis, K., Frey, G.: Towards a Model-Driven IEC 61131-Based Development Process in Industrial Automation. Journal of Software Engineering and Applications 4, pp. 217–226, 2011.
- [Thra12] Thramboulidis, K.: IEC 61131 as enabler of OO and MDD in industrial automation, in: Proceedings of the 10th IEEE International Conference on Industrial Informatics. Beijing, China, pp. 425–430, 2012.
- [Thra13] Thramboulidis, K.: IEC 61499 as an Enabler of Distributed and Intelligent Automation: A State-of-the-Art Review. Journal of Engineering 2013, 2013.
- [Tieg09] Tiegelkamp, M., John, K.H.: SPS-Programmierung mit IEC 61131-3. Springer, Berlin, 2009.
- [Tran06] Tranoris, C., Thramboulidis, K.: A tool supported engineering process for developing control applications. Computers in Industry 57, pp. 462–472, 2006.
- [Tren09] Trentesaux, D.: Distributed control of production systems. Engineering Applications of Artificial Intelligence 22, pp. 971–978, 2009.
- [Ulew12] Ulewicz, S., Schutz, D., Vogel-Heuser, B.: Design, implementation and evaluation of a hybrid approach for software agents in automation, in: Proceedings of the IEEE 17th Conference on Emerging Technologies Factory Automation. Krakow, Poland, pp. 1–4, 2012.
- [Vall09] Vallée, M., Kaindl, H., Merdan, M., Lepuschitz, W., Arnautovic, E., Vrba, P.: An automation agent architecture with a reflective world model in manufacturing systems, in: Proceedings of the IEEE International Conference on Systems, Man and Cybernetics. San Antonio, USA, pp. 305–310, 2009.
- [Vall11] Valles-Barajas, F.: A survey of UML applications in mechatronic systems. Innovations in Systems and Software Engineering 7, pp. 43–51, 2011.
- [Vánc11] Váncza, J., Monostori, L., Lutters, D., Kumara, S.R., Tseng, M., Valckenaers, P., Van Brussel, H.: Cooperative and responsive manufacturing enterprises. CIRP Annals - Manufacturing Technology 60, pp. 797–820, 2011.
- [VDI13] VDI GMA: Automation 2020 - Bedeutung der Automation bis zum Jahr 2020, 2013.
- [VDMA15] VDMA: Trendstudie: IT und Automation in den Produkten des Maschinenbau bis 2015, URL <http://itatautomation.vdma.org/article/-/articleview/792227> (accessed April, 18. 2016), 2012.
- [VDW97] VDW: Abteilungsübergreifende Projektierung komplexer Maschinen und Anlagen (VDW-Bericht). Aachen, 1997.
- [Virt10] Virta, J.: Application integration for production operations management using OPC Unified Architecture, Dissertation. Aalto University, Finland, 2010.
- [Voge05] Vogel-Heuser, B., Witsch, D., Katzke, U.: Automatic code generation from a UML model to IEC 61131-3 and system configuration tools, in: Proceedings of the International Conference on Control and Automation. Budapest, Hungary, pp. 1034–1039, 2005.
- [Voge09a] Vogel-Heuser, B.: Automation & Embedded Systems: Effizienzsteigerung im Engineering, Tagungen und Berichte. Kassel University Press, 2009.

- [Voge09b] Vogel-Heuser, B., Kegel, G., Wucherer, K.: Global information architecture for industrial automation. *atp-edition*, pp. 108–115, 2009.
- [Voge11] Vogel-Heuser, B., Braun, S., Kormann, B., Friedrich, D.: Implementation and evaluation of UML as modeling notation in object oriented software engineering for machine and plant automation, in: *Proceedings of the 18th IFAC World Congress*. Milan, Italy, 2011.
- [Voge14a] Vogel-Heuser, B., Diedrich, C., Fay, A., Jeschke, S., Kowalewski, S., Wollschlaeger, M., Göhner, P.: Challenges for Software Engineering in Automation. *Journal of Software Engineering and Applications* 7, pp. 440–451, 2014.
- [Voge14b] Vogel-Heuser, B., Schütz, D., Frank, T., Legat, C.: Model-driven engineering of Manufacturing Automation Software Projects – A SysML-based approach. *Mechatronics* 24, pp. 883–897, 2014.
- [Vrba09] Vrba, P., Radakovič, M., Obitko, M., Mařík, V.: Semantic Extension of Agent-Based Control: The Packing Cell Case Study, in: *Holonic and Multi-Agent Systems for Manufacturing*, *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, pp. 47–60, 2009.
- [Vyat03] Vyatkin, V.: Intelligent mechatronic components: control system engineering using an open distributed architecture, in: *Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation*. Lisbon, Portugal, pp. 277–284, 2003.
- [Vyat11] Vyatkin, V.: IEC 61499 as Enabler of Distributed and Intelligent Automation: State-of-the-Art Review. *Industrial Informatics*, *IEEE Transactions on* 7, pp. 768–781, 2011.
- [Vyat13] Vyatkin, V.: Software Engineering in Industrial Automation: State-of-the-Art Review. *IEEE Transactions on Industrial Informatics* 9, pp. 1234–1249, 2013.
- [Walt13] Walter, A., Götz, O.: Zur Zusammenarbeit von Automatisierungslieferant und Maschinenbauer. *SPS-Magazin Atlas STE*, 2013.
- [Warn93] Warnecke, H.-J.: *The Fractal Company: A Revolution in Corporate Culture*, Softcover reprint of the original 1st ed. 1993. ed. Springer-Verlag, Berlin, Heidelberg, 1993.
- [Wehr09] Wehrmeister, M.A.: An aspect-oriented model-driven engineering approach for distributed embedded real-time systems, *Dissertation*. University of Paderborn, 2009.
- [Weil08] Weilkiens, T.: *Systems Engineering with SysML/UML: Modeling, Analysis, Design*. Morgan Kaufmann, 2008.
- [Wern09] Werner, B.: Object-oriented extensions for iec 61131-3. *IEEE Industrial Electronics Magazine* 3, pp. 36–39, 2009.
- [West09] Westkämper, E., Zahn, E.: *Wandlungsfähige Produktionsunternehmen: Das Stuttgarter Unternehmensmodell*. Springer, 2009.
- [Weye15] Weyer, S., Schmitt, M., Ohmer, M., Gorecky, D.: Towards Industry 4.0 - Standardization as the crucial challenge for highly modular, multi-vendor production systems, in: *Proceedings of the 15th IFAC Symposium on Information Control Problems in Manufacturing*. Ottawa, Canada, pp. 579–584, 2015.
- [Weyr11] Weyrich, M., Klein, P., Laurowski, M., Wang, Y.: A Function-Oriented Approach for a Mechatronic Modularization of a Sensor-Guided Manufacturing System, in: *Proceedings of the 56th International Scientific Colloquium*. Illmenau, Germany, 2011.
- [Will07] Willard, B.: UML for systems engineering. *Computer Standards & Interfaces, ADC Modelling and Testing* 29, pp. 69–81, 2007.
- [Wits09] Witsch, D., Vogel-Heuser, B.: Close integration between UML and IEC 61131-3: New possibilities through object-oriented extensions, in: *IEEE Conference on Emerging Technologies Factory Automation*, 2009. *ETFA 2009*, pp. 1–6, 2009.
- [Wits10] Witsch, D., Ricken, M., Kormann, B., Vogel-Heuser, B.: PLC-statecharts: An approach to integrate umlstatecharts in open-loop control engineering, in: *Proceedings of the 8th IEEE International Conference on Industrial Informatics*. Osaka, Japan, pp. 915–920, 2010.

-
- [Wool95] Wooldridge, M., Jennings, N.R.: Intelligent Agents: Theory and Practice. Knowledge Engineering Review 10, pp. 115–152, 1995.
- [Wool99] Wooldridge, M.: Intelligent Agents, in: Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. MIT Press, Cambridge, MA, USA, pp. 27–79, 1999.
- [Wüns07] Wunsch, G.: Methoden für die virtuelle Inbetriebnahme automatisierter Produktionssysteme, Dissertation. University of Munich, 2007.
- [Yu10] Yu, L., Quirós, G., Epple, U.: Service-oriented process control for complex multifunctional plants: Concept and case study, in: Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation. Bilbao, Spain, pp. 1–8, 2010.
- [Yu11] Yu, L., Quirós, G., Epple, U.: An Assistance System Approach for Flexible Product Transport Operations in Process Plants, in: Proceedings of the 18th IFAC World Congress. Milan, Italy, pp. 7304–7309, 2011.
- [Zäh05] Zäh, M.F., Wunsch, G.: Schnelle Inbetriebnahme von Produktionssystemen - Qualitätssicherung von automatisierten Maschinen durch Simulation. wt - Werkstattstechnik, pp. 699–704, 2005.
- [Zeeb08] Zeeb, E., Priiter, S., Golatowski, F., Berger, F.: A Context Aware Service-Oriented Maintenance System for the B2B Sector, in: Proceedings of the 22nd International Conference on Advanced Information Networking and Applications - Workshops. Ginowan, Japan, pp. 1381–1386, 2008.
- [Zeeb10] Zeeb, E., Moritz, G., Timmermann, D., Golatowski, F.: WS4D: Toolkits for Networked Embedded Systems Based on the Devices Profile for Web Services, in: Proceedings of the 39th International Conference on Parallel Processing Workshops. San Diego, USA, pp. 1–8, 2010.
- [Zoit09a] Zoitl, A., Strasser, T., Sunder, C., Baier, T.: Is IEC 61499 in harmony with IEC 61131-3? IEEE Industrial Electronics Magazine 3, pp. 49–55, 2009.
- [Zoit09b] Zoitl, A., Vyatkin, V.: IEC 61499 architecture for distributed automation: The “glass half full” view. IEEE Industrial Electronics Magazine 3, pp. 7–23, 2009.
- [Zühl10] Zühlke, D.: SmartFactory -Towards a factory-of-things. Annual Reviews in Control 34, pp. 129–138, 2010.
- [Zühl12] Zühlke, D., Ollinger, L.: Agile Automation Systems Based on Cyber-Physical Systems and Service-Oriented Architectures, in: Advances in Automation and Robotics, Vol.1, Lecture Notes in Electrical Engineering. Springer Berlin Heidelberg, pp. 567–574, 2012.

Standards and Guidelines

- [DIN03] DIN 8580: Fertigungsverfahren - Begriffe, Einteilung/ManufacturingProcesses - Terms and definitions, division, 2003.
- [FIPA02] FIPA: FIPA Abstract Architecture Specification. Foundation for Intelligent Physical Agents, 2002.
- [IEC02] IEC 60848: GRAFCET - Specification language for sequential function charts, 2002.
- [IEC05] IEC 61499-1: Function blocks - Part 1: Architecture, 2005.
- [IEC08] IEC 62264-1: Enterprise-control System Integration - Part 1: Models and terminology, 2008.
- [IEC13] IEC 61131-3: Programmable Controllers Part 3: Programming Languages, 2013.
- [ISA00] ISA: ISA S95.00.01-2000 - Enterprise-Control System Integration - Part 1: Models and Terminology, 2000.
- [ISA95] ISA: ISA S88.01 - Batch Control, 1995.

- [OASI06] OASIS: Reference Model for Service Oriented Architecture 1.0, 2006.
- [OMG03] OMG: MDA Guide V1.0.1 (www document), URL <http://www.omg.org/cgi-bin/doc?omg/03-06-01> (accessed April, 19. 2016), 2003.
- [OMG06] OMG: Meta Object Facility (MOF) Core Specification Version 2.0, 2006.
- [OMG09] OMG: Service oriented architecture Modeling Language (SoaML) Specification, 2009.
- [OMG11] OMG: Unified Modeling Language (UML) V2.4.1, 2011.
- [OMG12] OMG: Systems Modeling Language (SysML) V1.3, 2012.
- [OMG15] OMG: XML Metadata Interchange (XMI) V2.5.1, 2015.
- [OPC07] OPC Foundation: OPC Unified Architecture Specification Part 10: Programs, 2007.
- [PLCo11a] PLCopen: PLCopen Functions for Motion Control Version 2.0 (www document), URL http://www.plcopen.org/pages/tc2_motion_control/ (accessed April, 18.2016), 2011.
- [PLCo11b] PLCopen: Technical Specification: Function blocks for motion control, Version 2.0, 2011.
- [VDI90] VDI 2860: Montage- und Handhabungstechnik - Handhabungsfuntionen, Handhabungseinrichtungen; Begriffe, Definitionen, Symbole, 1990.
- [VDI04] VDI/VDE - GMA: VDI/VDE 2206 - Design methodology for mechatronic systems, 2004.
- [VDI05] VDI/VDE - GMA: VDI/VDE 3682 - Formalised process descriptions, 2005.
- [VDI08] VDI/VDE - GMA: VDI/VDE 4499 Part 1 - Digital factory - Fundamentals, 2008.
- [VDI10] VDI/VDE - GMA: VDI/VDE 3695 Part 3 - Engineering of industrial plants - Evaluation and optimization: Subject methods, 2010.
- [VDI11] VDI/VDE - GMA: VDI/VDE 5200 Part 1 - Factory planning - Planning procedures, 2011.
- [W3C04] W3C Working Group: Web Services Architecture, 2004.

Student Projects

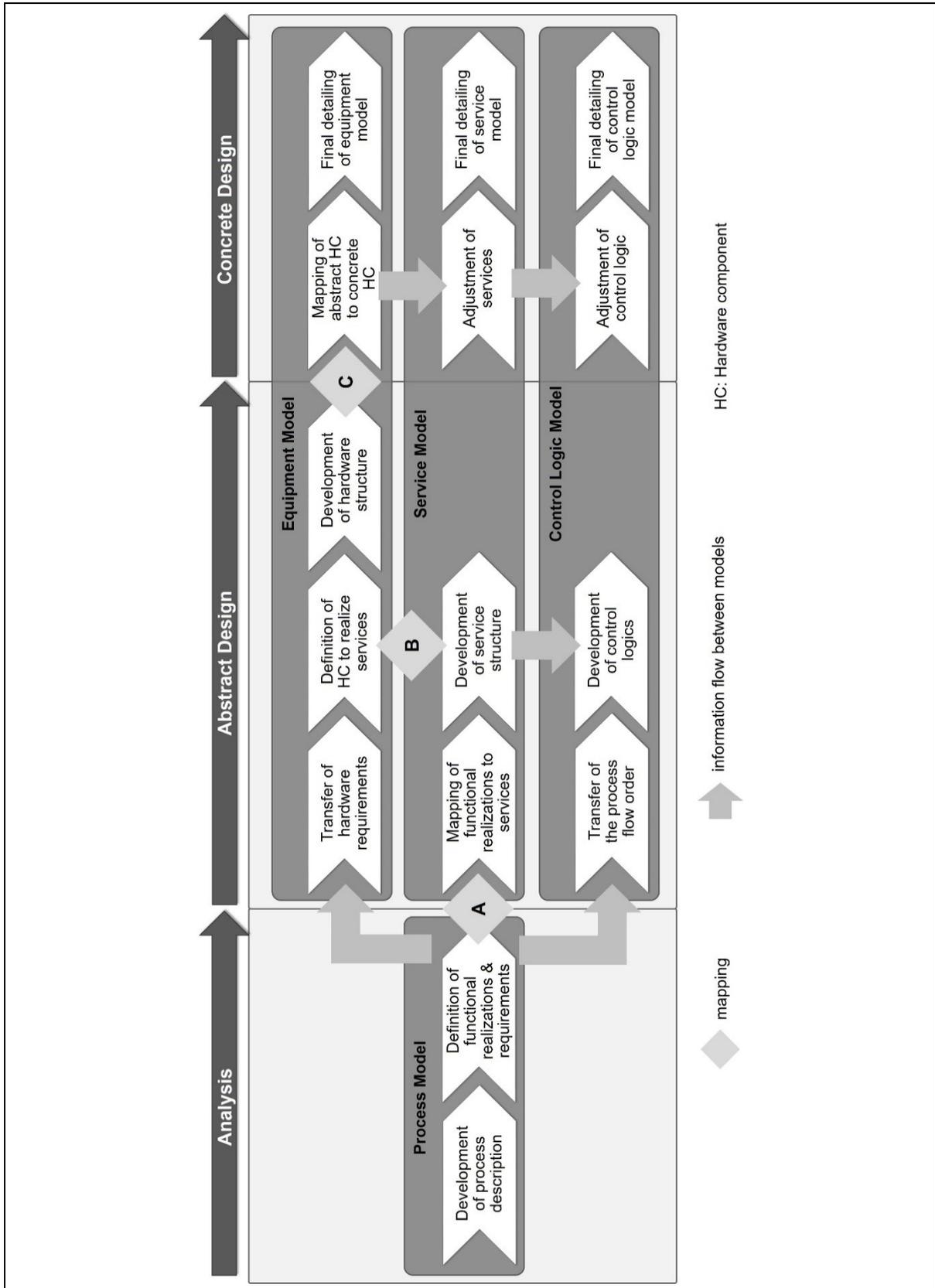
- [Ade13] Adler, A.: Automatic Model Transformation of the SOA Planning Model from SysML to JGrafchart, Seminar Paper. Technische Universität Kaiserslautern, Kaiserslautern, 2013.
- [Glei11] Gleis, P., Heilmann, C., Schuricht, F.: Vergleich und Umsetzung von drei verschiedenen Steuerungsansätzen: IEC 61499, Agentensysteme und Serviceorientierte Architekturen, Seminar Paper. Technische Universität Kaiserslautern, Kaiserslautern, 2011.
- [Hamm11] Hammami, M.: Entwicklung von Steuerungsprogrammen für DPWS-fähige Feldgeräte mit Grafchart, Seminar Paper. Technische Universität Kaiserslautern, Kaiserslautern, 2011.
- [Henn12] Hennecke, A., Stelter, C.: Entwicklung und Anwendung eines Modell- und Bibliothekskonzeptes zur Unterstützung einer service-orientierten Planungsmethodik, Seminar Paper. Technische Universität Kaiserslautern, Kaiserslautern, 2012.
- [Henn13] Hennecke, A.: Realisierung einer SOA-AT-Architektur für die Pick'&Place-Einheit des Industrie 4.0 – Demonstrationssystems, Seminar Paper. Technische Universität Kaiserslautern, Kaiserslautern, 2013.
- [Kaze10] Kazemi, R: Forschungsaktivitäten zu Serviceorientierten Architekturen im Produktionsumfeld, Seminar Paper. Technische Universität Kaiserslautern, Kaiserslautern, 2011.
- [Rou12] Roubanov, D: Prozessorientierte Anlagenplanung zur Unterstützung der Entwicklung von Serviceorientierten Steuerungssystemen, Diploma Thesis. Technische Universität Kaiserslautern, Kaiserslautern, 2012.

-
- [Senf11] Senft, M.: Entwicklung einer dienstorientierten Prozesssteuerung, Seminar Paper. Technische Universität Kaiserslautern, Kaiserslautern, 2011.
- [Senf13] Senft, M.: „SOA-SPS“ - Entwicklung von Serviceorientierten Steuerungsarchitekturen auf Basis Speicherprogrammierbarer Steuerungen, Diploma Thesis. Technische Universität Kaiserslautern, Kaiserslautern, 2013.
- [Stel11] Konzeption und Realisierung eines Qualitätssicherungsszenarios auf Basis eines service-orientierten Steuerungskonzepts, Seminar Paper. Technische Universität Kaiserslautern, Kaiserslautern, 2011.
- [Stel12] Stelter, C. Evaluation der serviceorientierten Planungsmethodik für industrielle Steuerungssysteme anhand praxisnaher Planungsszenarien, Diploma Thesis. Technische Universität Kaiserslautern, Kaiserslautern, 2012.

Figures

- [FigDigi01] Image of a Digi ME9210 embedded system:
<http://www.digi.com/products/embedded-systems/system-on-modules/digiconnectme9210>
- [FigSim01] Image of Simatic S7 controller:
https://mall.industry.siemens.com/collaterals/files/21/jpg/P_ST70_XX_02242i.jpg
- [FigSick01] Image of Sick IM18-12NPO-ZC1 inductive proximity switch:
<https://www.sick.com/de/de/naeherungssensoren/induktive-naeherungssensoren/im-standard/im18-12npo-zc1/p/p235577>
- [FigPF01] Image of Pepperl & Fuchs NJ20+U1+E2-Y287109 inductive proximity switch:
http://www.pepperl-fuchs.com/global/de/classid_143.htm?view=productdetails&prodid=73136

Appendix A: Model-driven Engineering Methodology



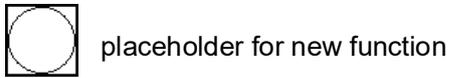
Appendix B: Function Library

Assembly and handling functions according to the VDI 2868		
store:	check:	modify quantities:
 store oriented	 check	 divide
 store partially oriented	 check presence	 unite
 store disoriented	 check identity	 partition
move:	 check shape	 apportion
 rotate	 check size	 branch
 shift	 check color	 join
 displace	 check weight	 sort
 orientate	 check position	
 position	 check orientation	lock/maintain:
 order	 measure	 hold
 guide	 count	 release
 pass	 measure orientation	 clamp
 convey	 measure position	 release

Manufacturing process according to the VDI 8580

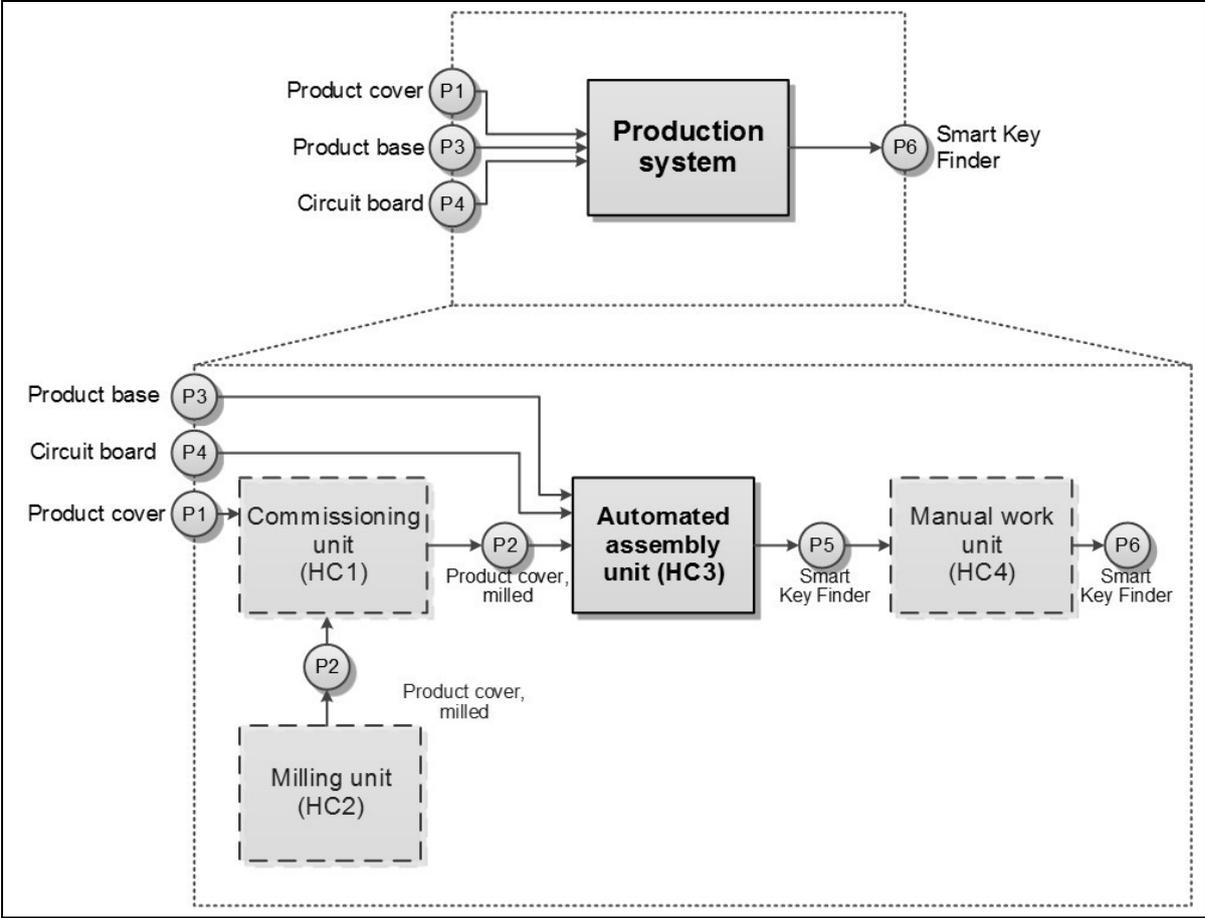


Functions defined in this thesis

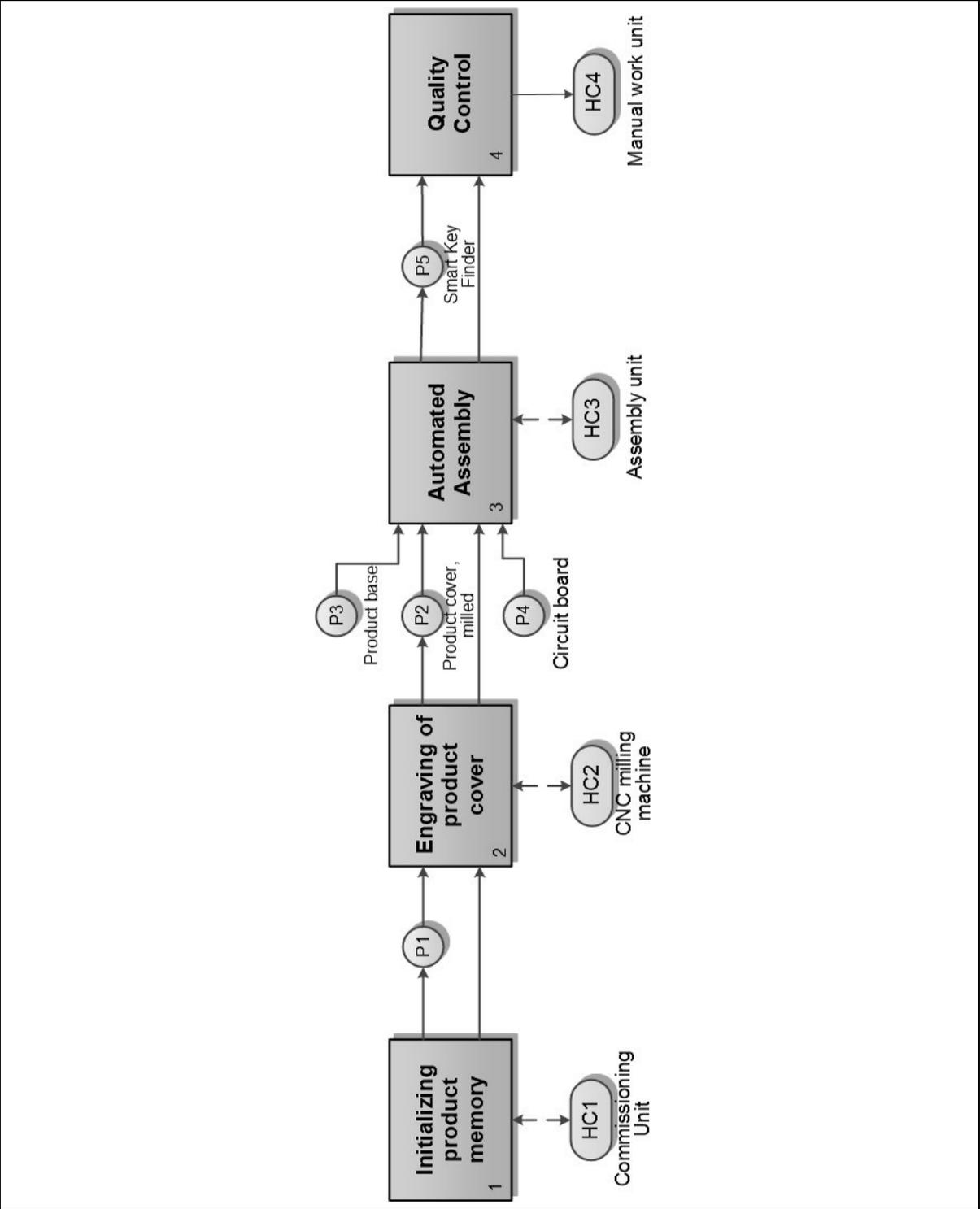


Appendix C: Process Model of Use Case

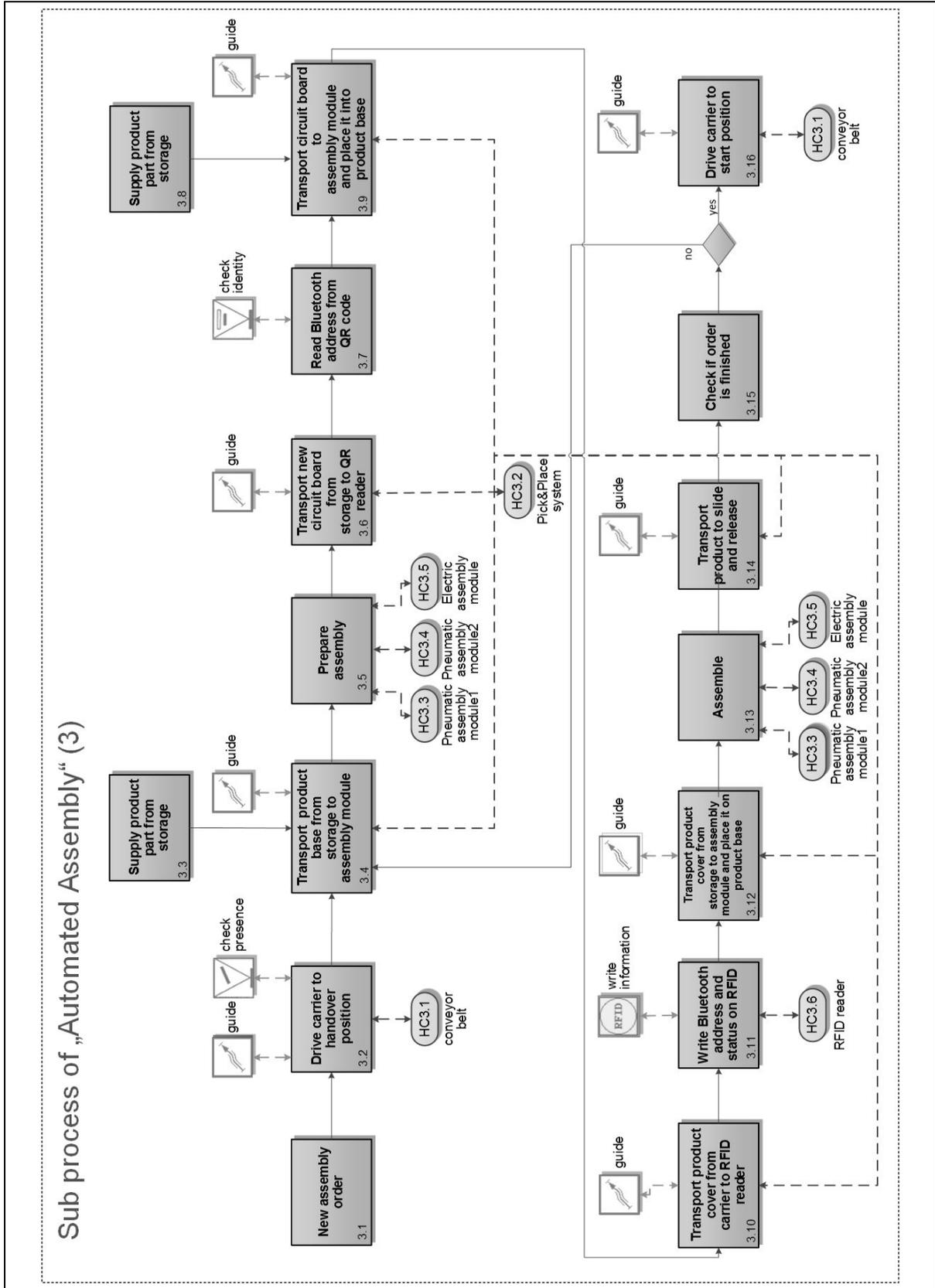
C.1: Rough line layout



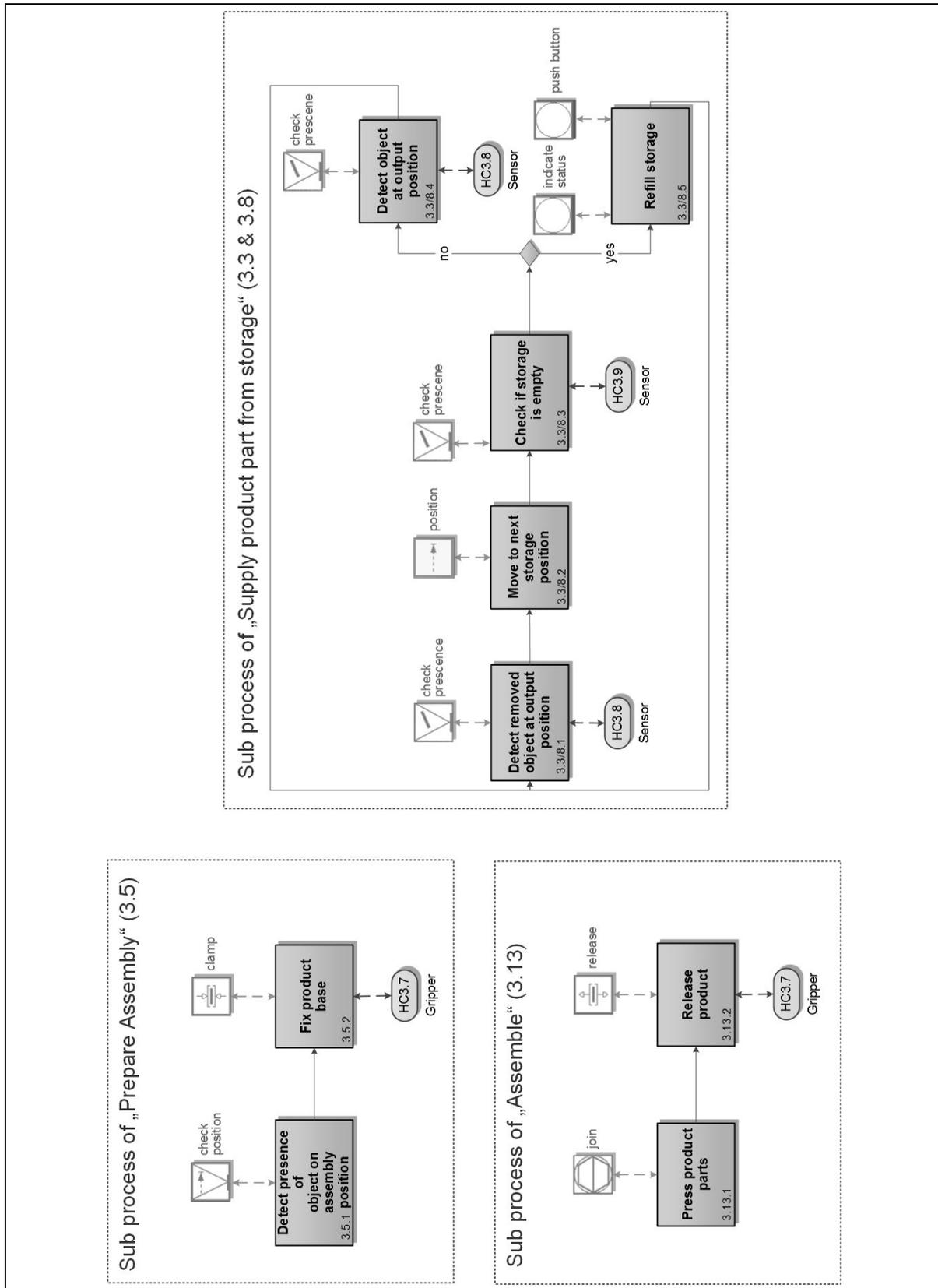
C.2: Process model level 0



C.3: Process model level 1



C.4: Process model level 2



Appendix D: Mappings of Use Case

D.1: Mapping A

	Process step ID	Function	Abstract Hardware Component	Abstract Service	
Level 0	1	/	HC1		
	2	/	HC2		
	3	/	HC3		
	4	/	HC4		
Level 1	3.1	/	/	/	
	3.2		guide	HC3.1	translateElectric
			check presence	/	detectInductive
	3.3	/	/	/	
	3.4		guide	HC3.2	pickAndPlace
	3.5	/	HC3.3, HC3.4, HC3.5	/	
	3.6		guide	HC3.2	pickAndPlace
	3.7		check identity	/	identifyOptic
	3.8	/	/	/	
	3.9		guide	HC3.2	pickAndPlace
	3.10		guide	HC3.2	pickAndPlace
	3.11		write information	HC3.6	identifyRFID
	3.12		guide	HC3.2	pickAndPlace
	3.13	/	/	/	
	3.14		guide	HC3.2	pickAndPlace
	3.15	/	/	/	
3.16		guide	HC3.1	translateElectric	

	Process step ID	Function	Abstract Hardware Component	Abstract Service	
Level 2	3.3.1		check presence	HC3.8	detectOptic
	3.3.2		position	/	translateElectric
	3.3.3		check presence	HC3.9	detectOptic
	3.3.4		check presence	HC3.8	detectOptic
	3.3.5		indicate status	/	lightElectric
			push button	/	detectHaptic
	3.5.1		check position	/	detectOptic
	3.5.2		clamp	HC3.7	gripPneumatic gripElectric
	3.13.1		join	/	translatePneumatic translateElectric
3.13.2		release	HC3.7	gripPneumatic gripElectric	

D.2: Mapping B and Refinement of Abstract Service and Hardware Structure

Composed Services		Basic Services	Hardware Component Field Devices	Hardware Component Modules
verifyingTranslate ElectricBelt1	translationElectric1		conveyor belt	conveyor belt module
			conveyor belt	
			servo controller	
	detectInductive1	inductive proximity switch		
	detectInductive2	inductive proximity switch		
pickAndPlace1	tbd		handling system	pick-and-place module
			portal module	
	identifyOptic1		matrix code reader	identification system
	identifyRFID1		RFID RW	
control storage 1/2	detectOptic1/3		one-way light barrier	storage product base/ storage circuit board
	translationElectric6/7		stepping motor	
	detectOptic2/4		fork sensor	
	lightElectric1/2		indicationsignal lamp	
	detectHaptic1/2		pushbutton	
assemblyPneumatic1		detectOptic5	one-way light barrier	pneumatic assembly module
	VerifyingGrip Pneumatic1	gripPneumatic1	angular air gripper	
		detectMagnetic1	magnetic field sensitive proximity switch	
		detectMagnetic2	magnetic field sensitive proximity switch	
	verifyingTransla tion Pneumatic1	translationPneumatic1	compact cylinder pneumatic	
		detectMagnetic3	magnetic field sensitive proximity switch	
		detectMagnetic4	magnetic field sensitive proximity switch	

Composed Services		Basic Services	Hardware Component Field Devices	Hardware Component Modules
assemblyPneumatic2		detectOptic6	one-way light barrier	pneumatic assembly module PnP
	verifyingTranslationPneumatic1	translationPneumatic2	compact cylinder pneumatic	
			valve terminal	
		detectMagnetic5	magnetic field sensitive proximity switch	
	detectMagnetic6	magnetic field sensitive proximity switch		
	detectInductive3	inductive proximity switch		
assemblyElectric1		detectOptic7	one-way light barrier	electric assembly module
		gripElectric1	electric gripper	
		translateElectric1	compact cylinder electric	

D.3: Refinement of Concrete Services based on Mapping C

Refinement of Composed Service “pickAndPlace1”:

Composed Services	Basic Services	Hardware Component Field Devices		Hardware Component Modules
pickAndPlace1	rotatePneumatic1	rotation module		pick-and-place module
	gripVacuumEject1	vacuum pad		
	DetectMagnetic9	magnetic field sensitive proximity switch		
	DetectMagnetic10	magnetic field sensitive proximity switch		
	TranslateElectricLinMot1	linear motion module		
		servo controller		
	TranslateElectricLinMot2	handling system	linear servo motor	
TranslateElectricLinMot3	linear servo motor			

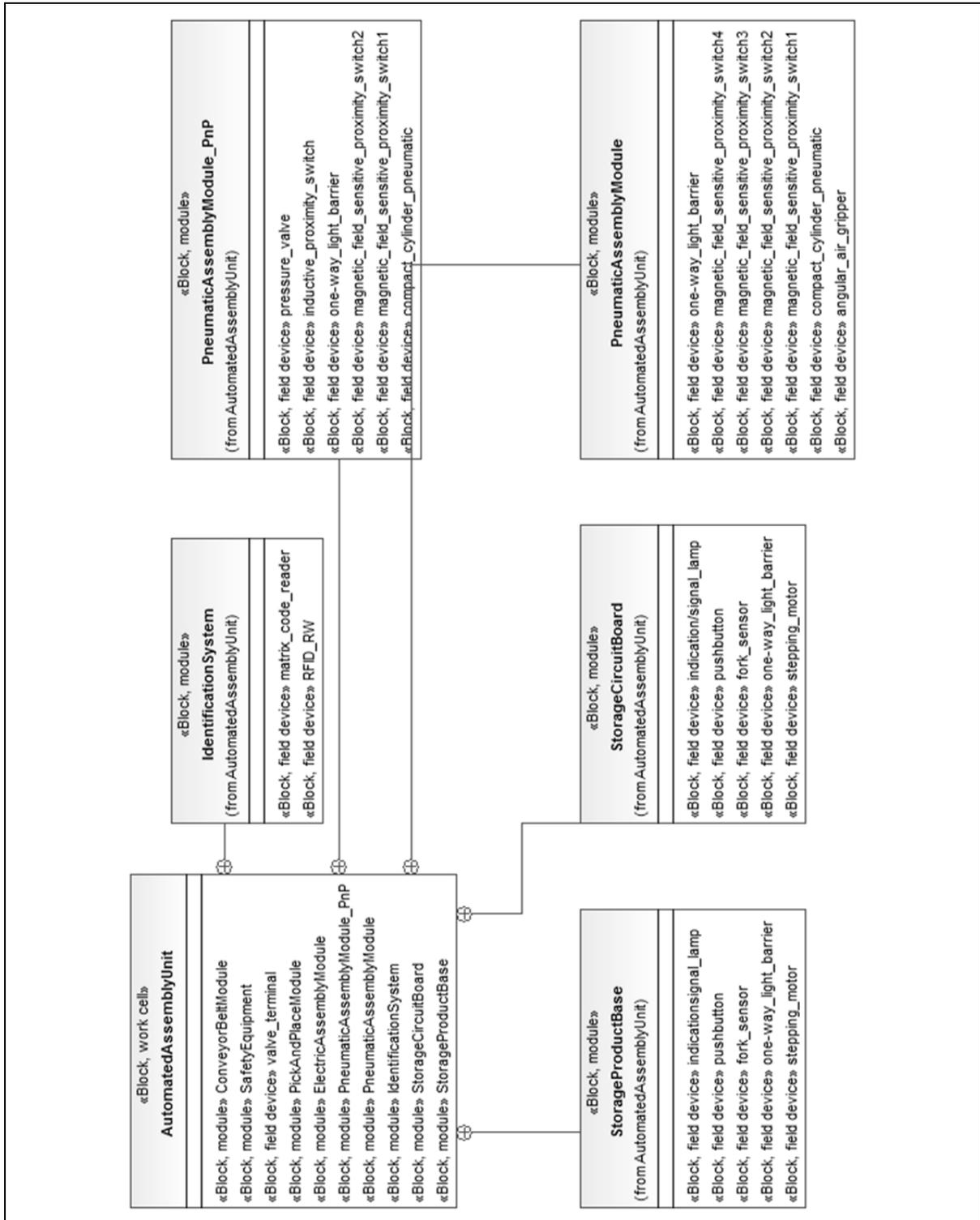
Refinement of device-specific concrete services:

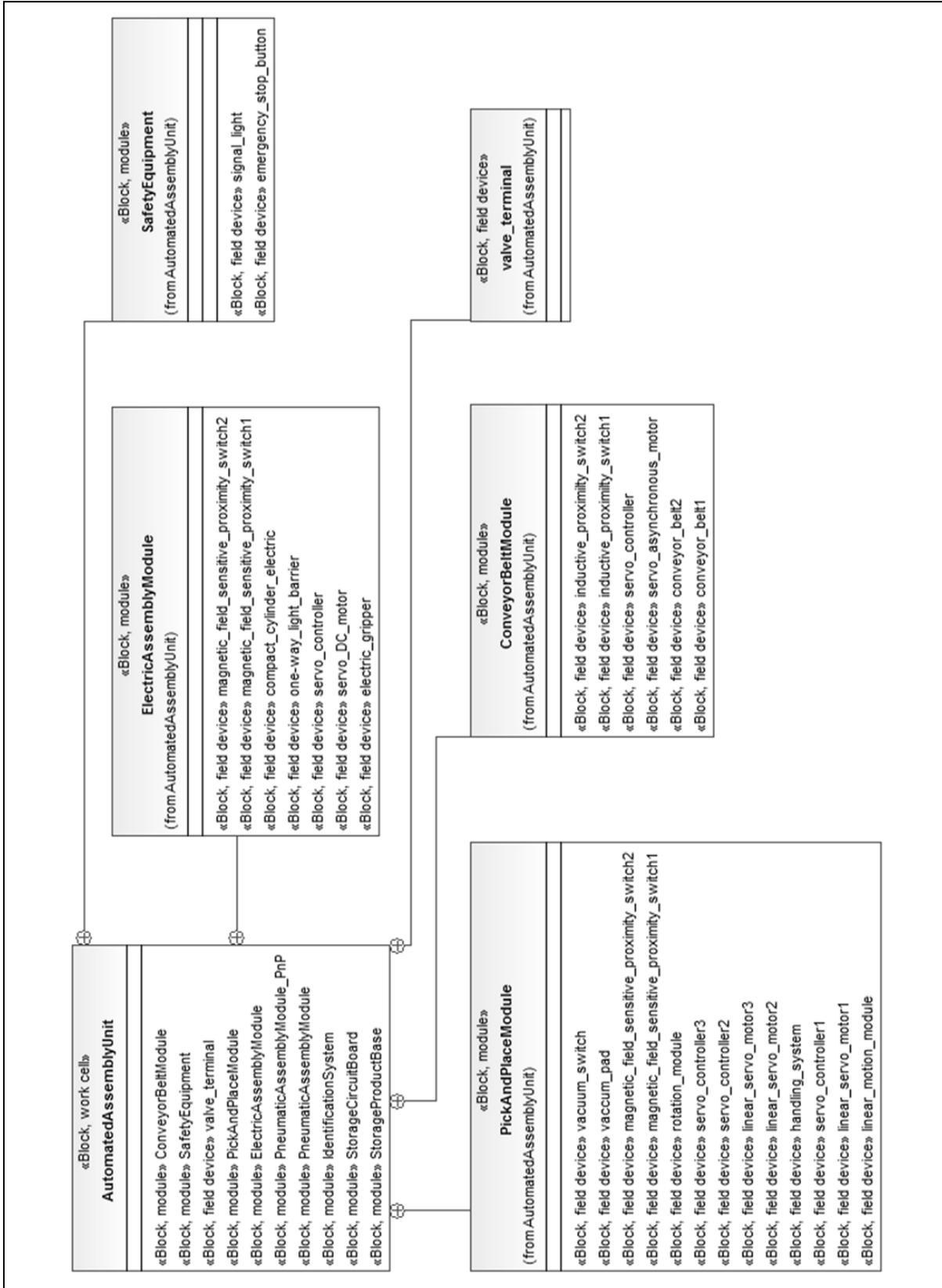
Abstract Services	Concrete Services	Hardware Component Field Devices
translationElectric1	translationElectricBelt1	conveyor belt
		conveyor belt
		servo controller
translationElectric6/7	translationElectricStep1/2	stepping motor
gripElectric	gripElectricRoehm1	electric gripper
translateElectric1	translateElectricFesto1	compact cylinder electric

Appendix E: Abstract Equipment Model of Use Case

Abstract equipment model in SysML: “AutomatedAssemblyUnit”

Section 1

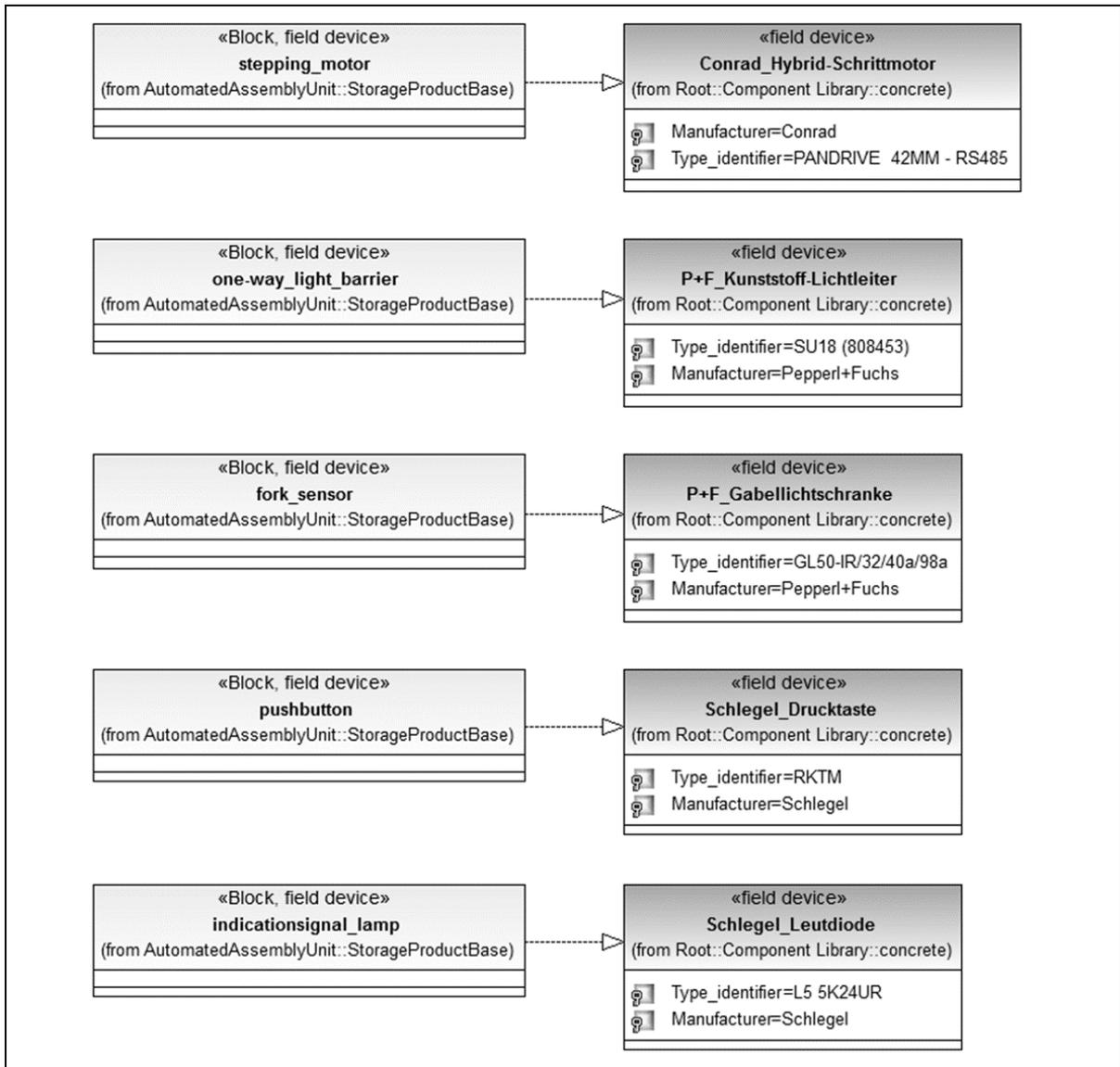




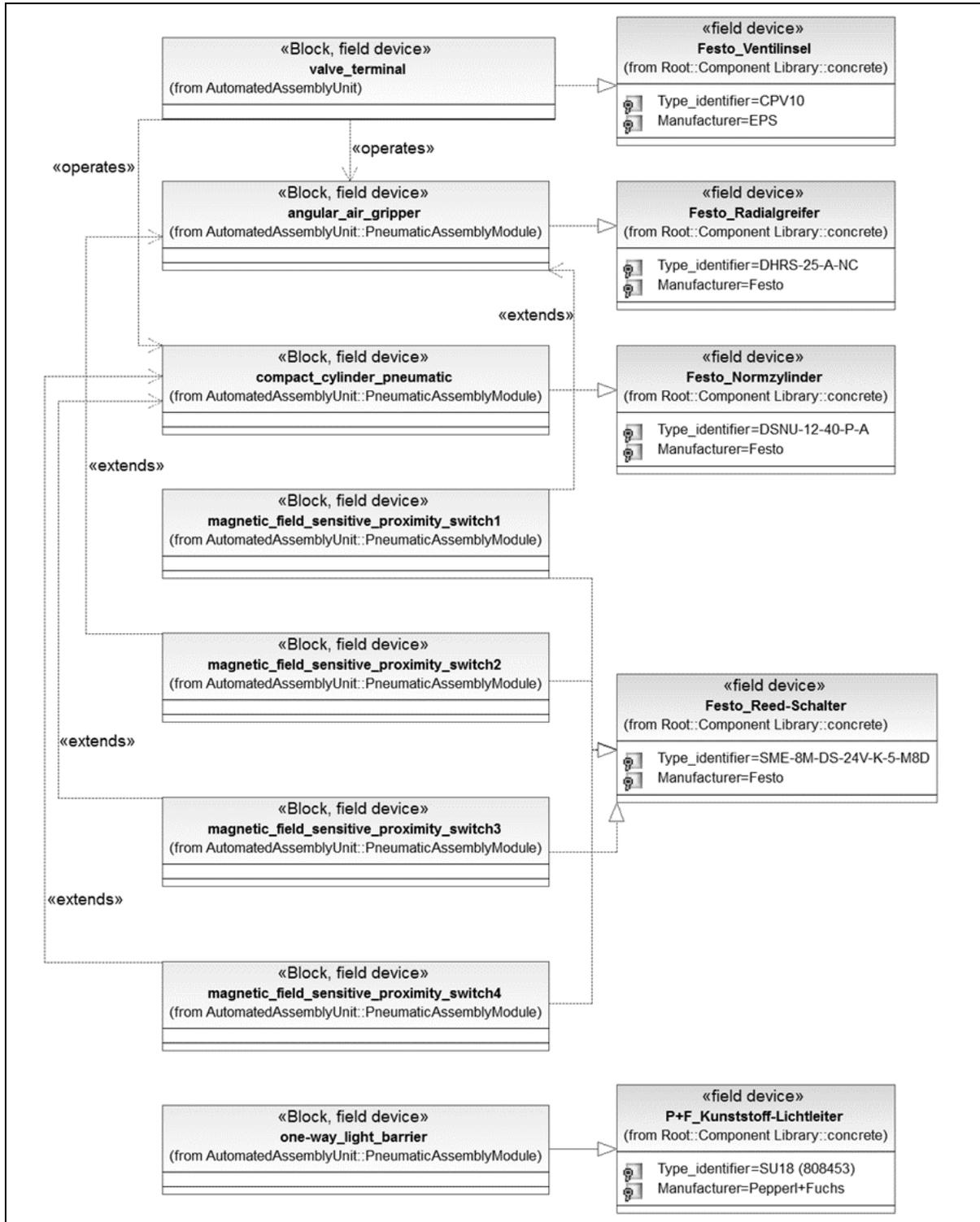
Appendix F: Concrete Equipment Model of Use Case

F.1: Concrete equipment model in SysML: StorageProductBase

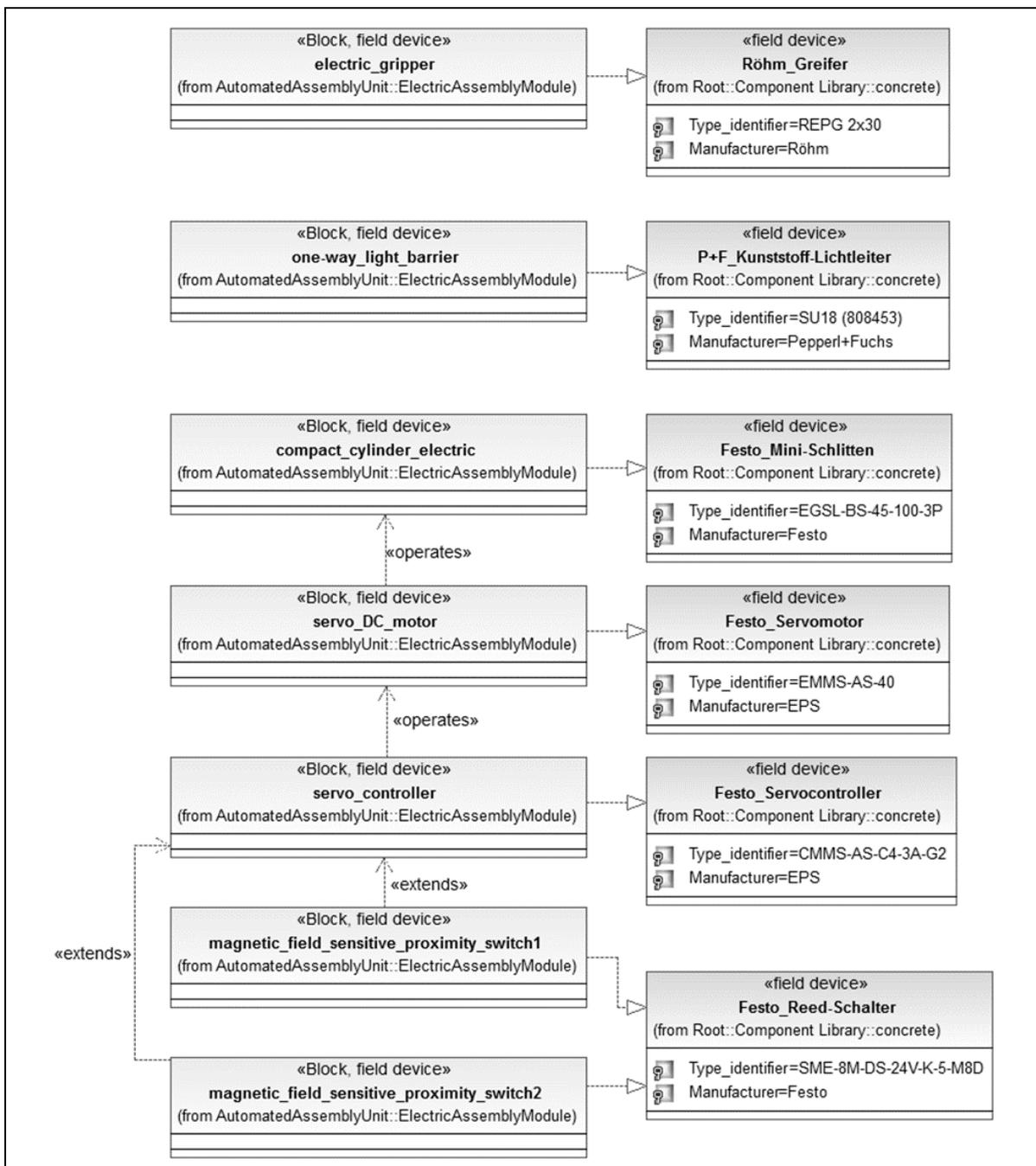
(StorageCircuitBoard is equivalent)



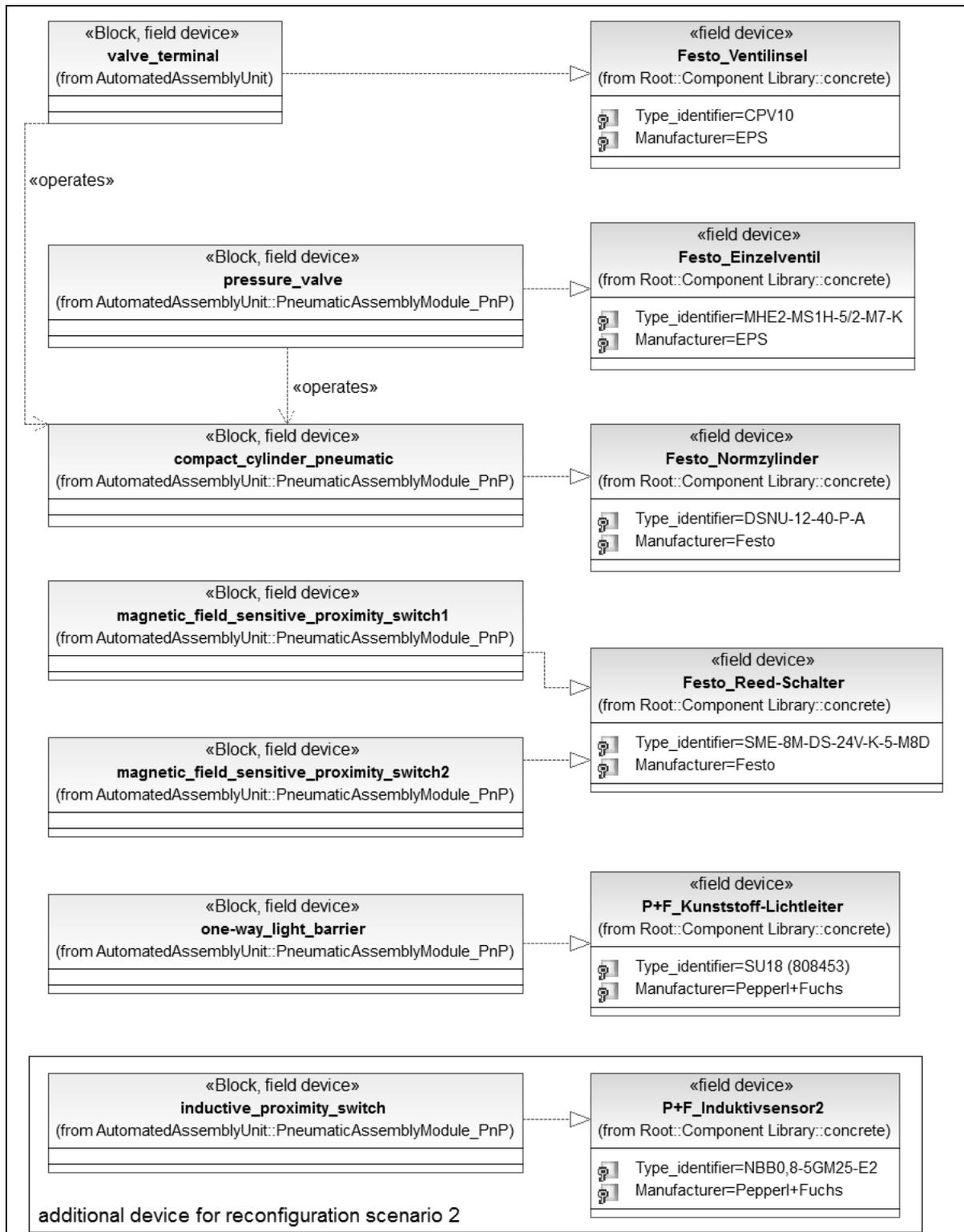
F.2: Concrete equipment model in SysML: PneumaticAssemblyModule



F.3: Concrete equipment model in SysML: ElectricAssemblyModule

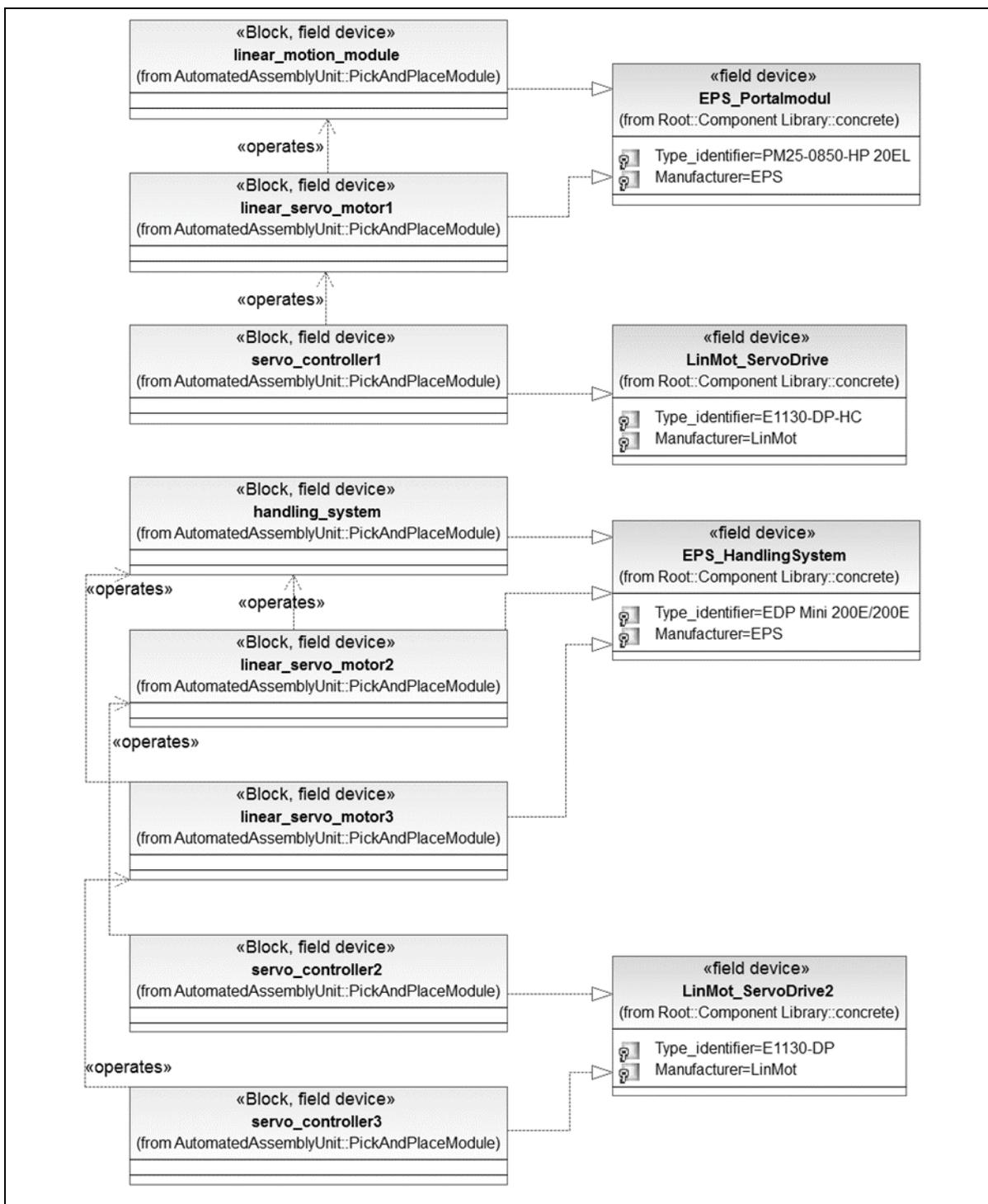


F.4: Concrete equipment model in SysML: PneumaticAssemblyModulePnP

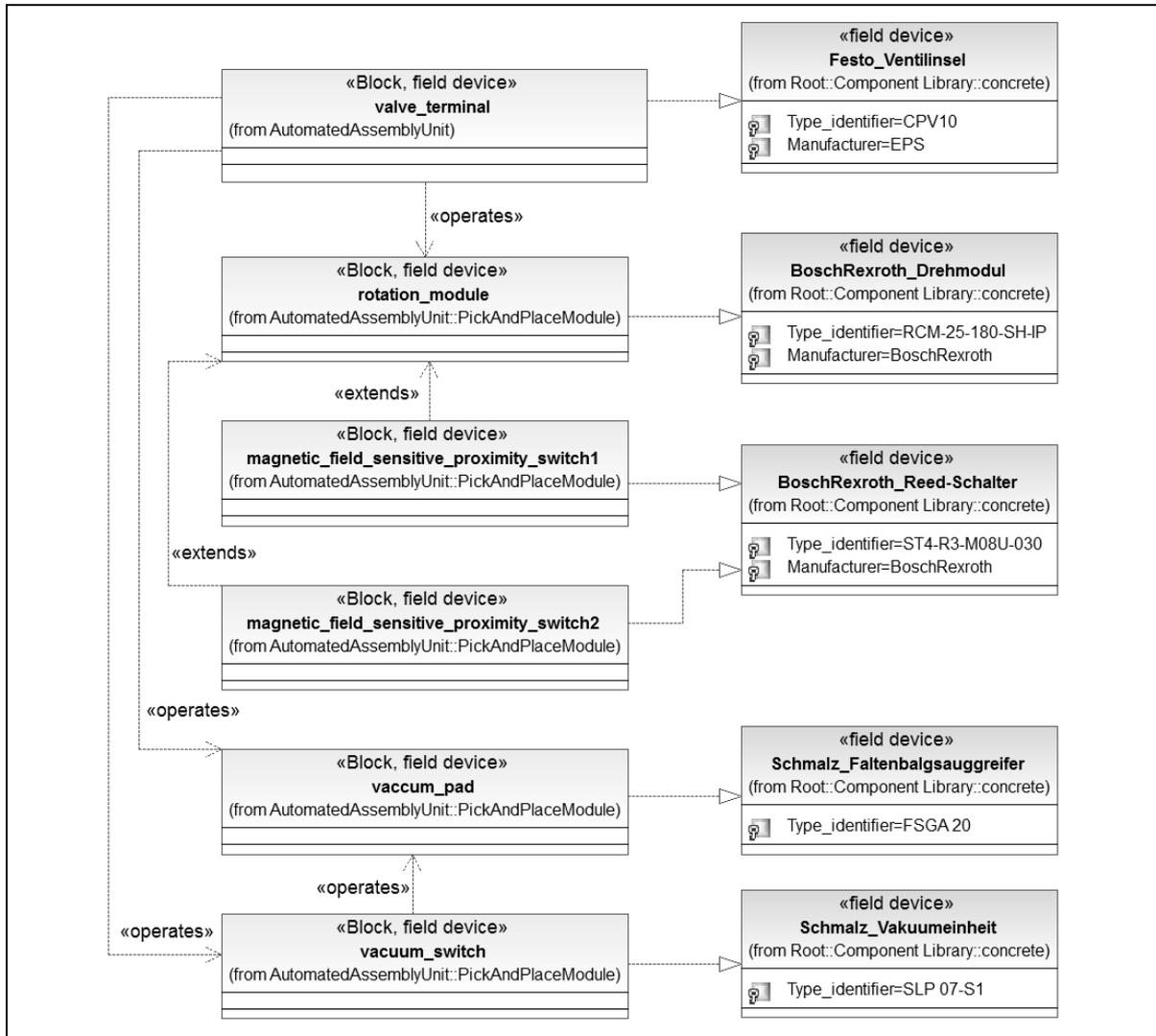


F.5: Concrete equipment model in SysML: PneumaticAssemblyModulePnP

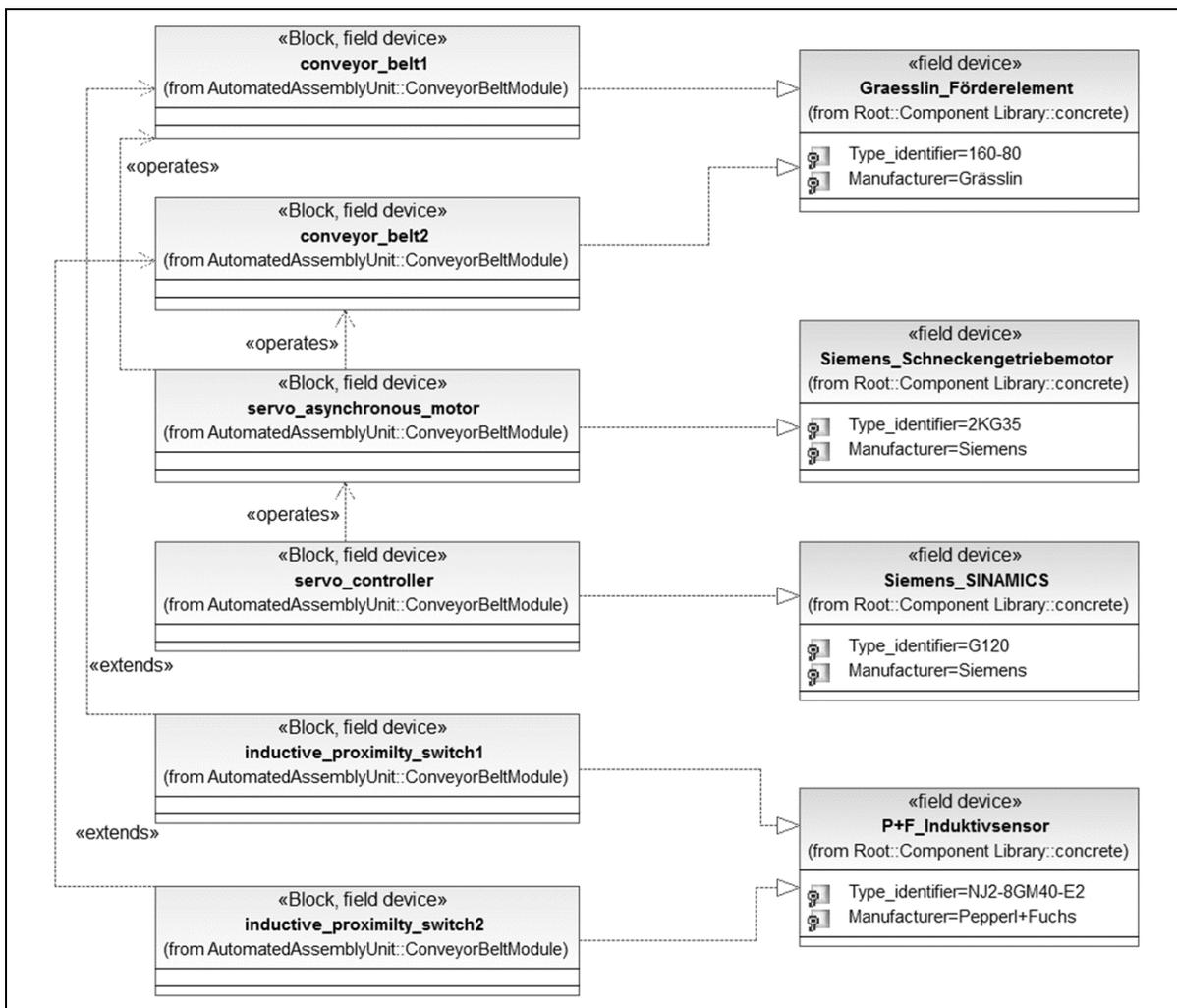
Section 1



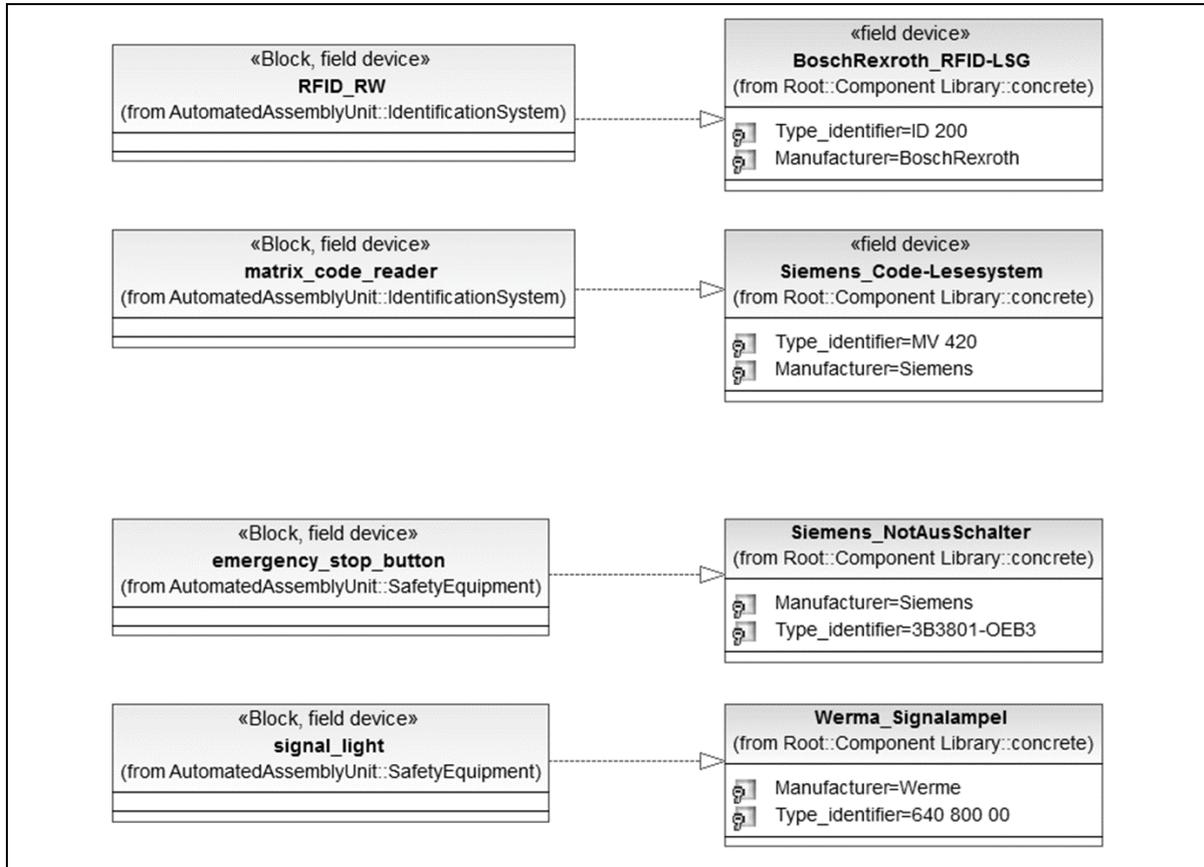
Section 2



F.6: Concrete equipment model in SysML: ConveyorBeltModule

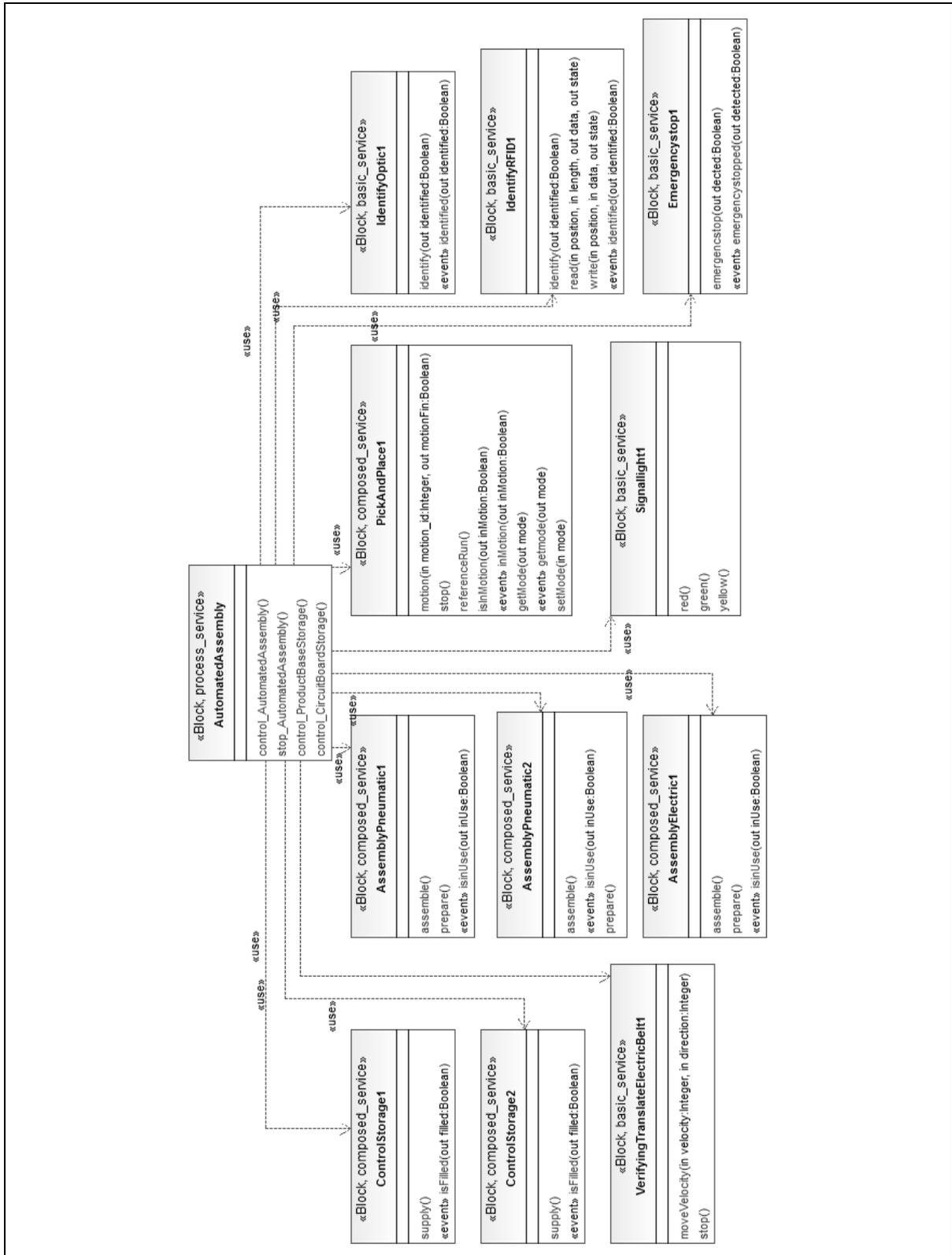


F.7: Concrete equipment model in SysML: Other hardware components of AutomatedAssemblyUnit

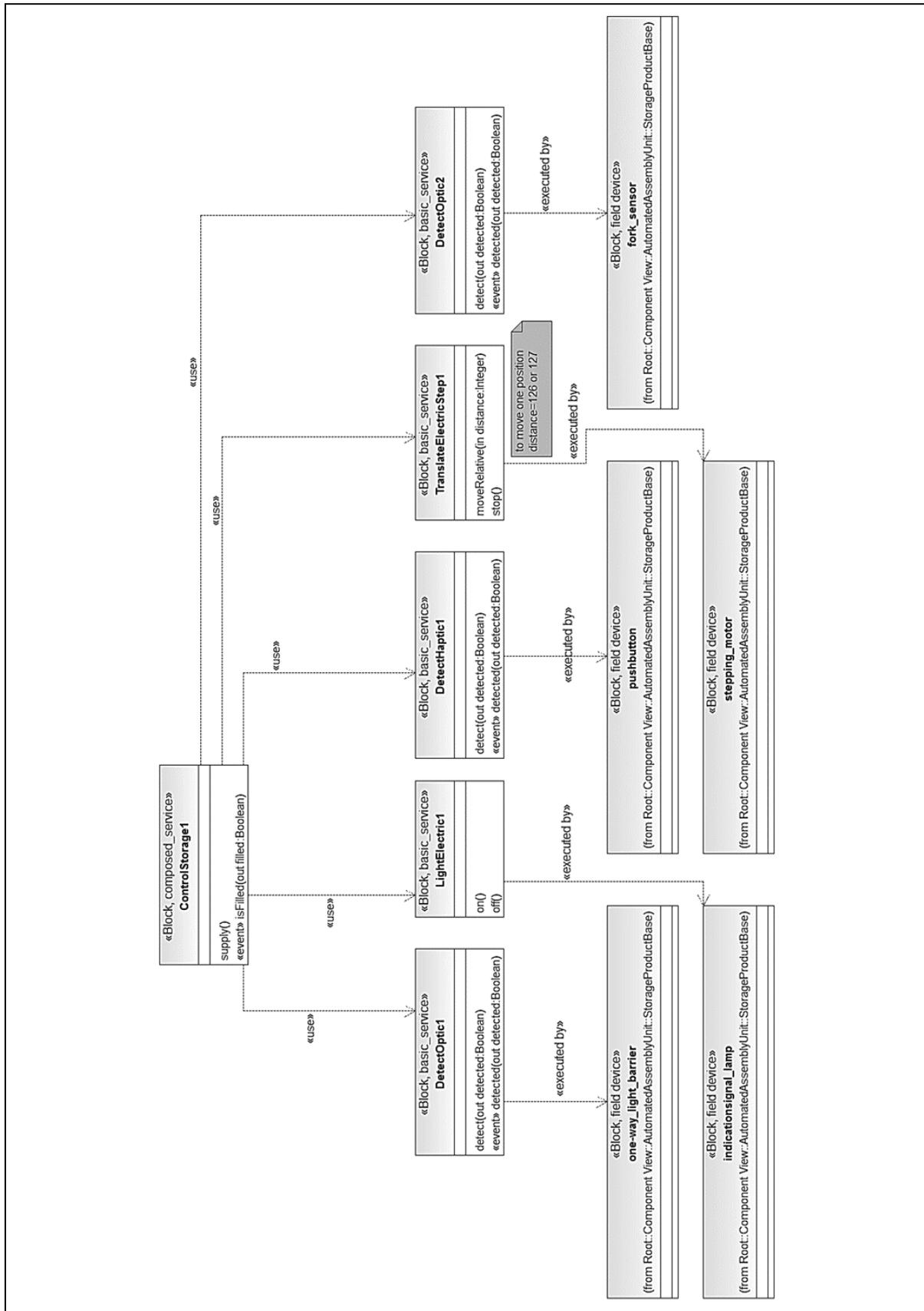


Appendix G: Service Model of Use Case

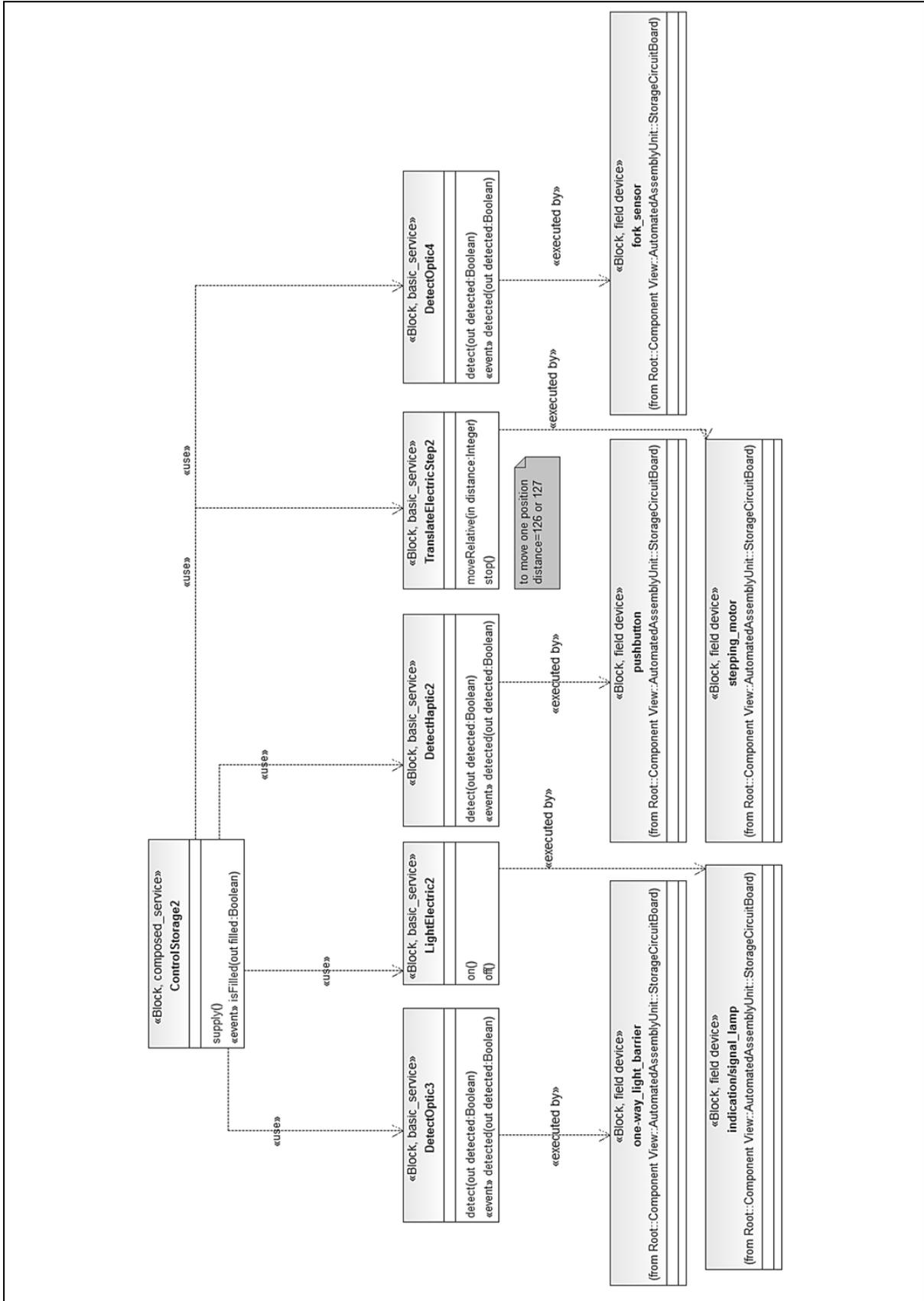
G.1: Service model in SysML: Process service and the services it uses



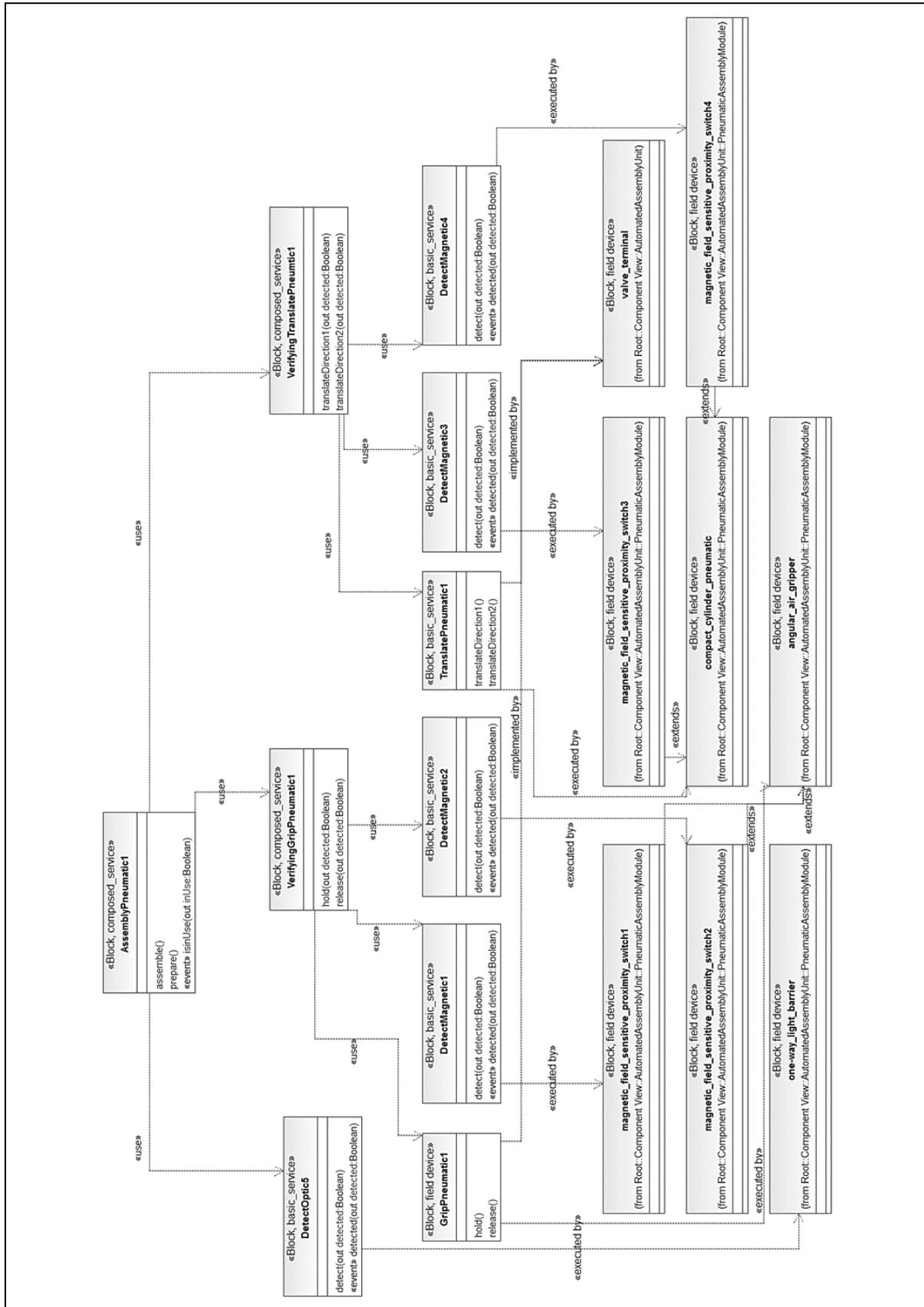
G.2: Service model in SysML: “ControlStorage1”



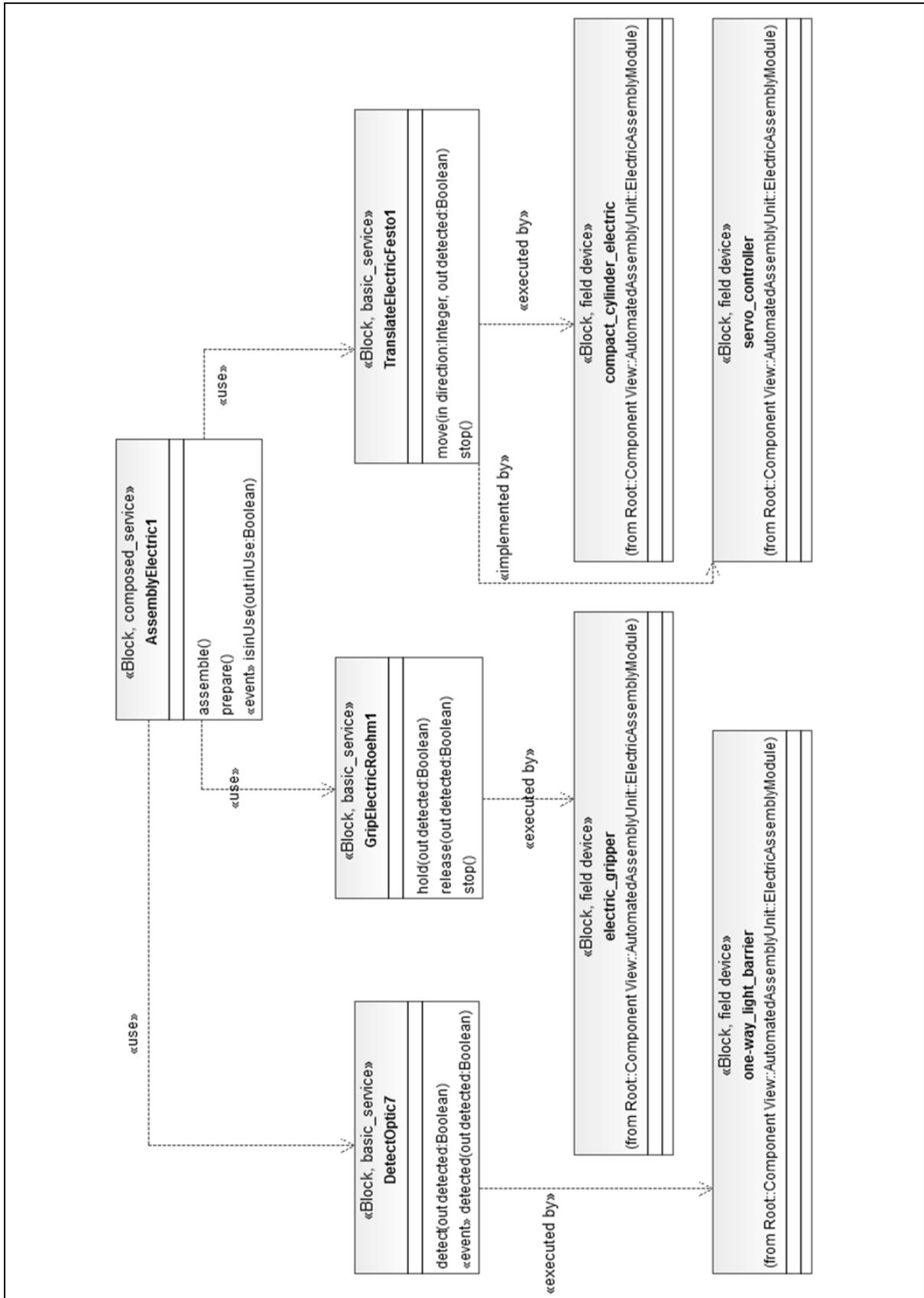
G.3: Service model in SysML: “ControlStorage2”



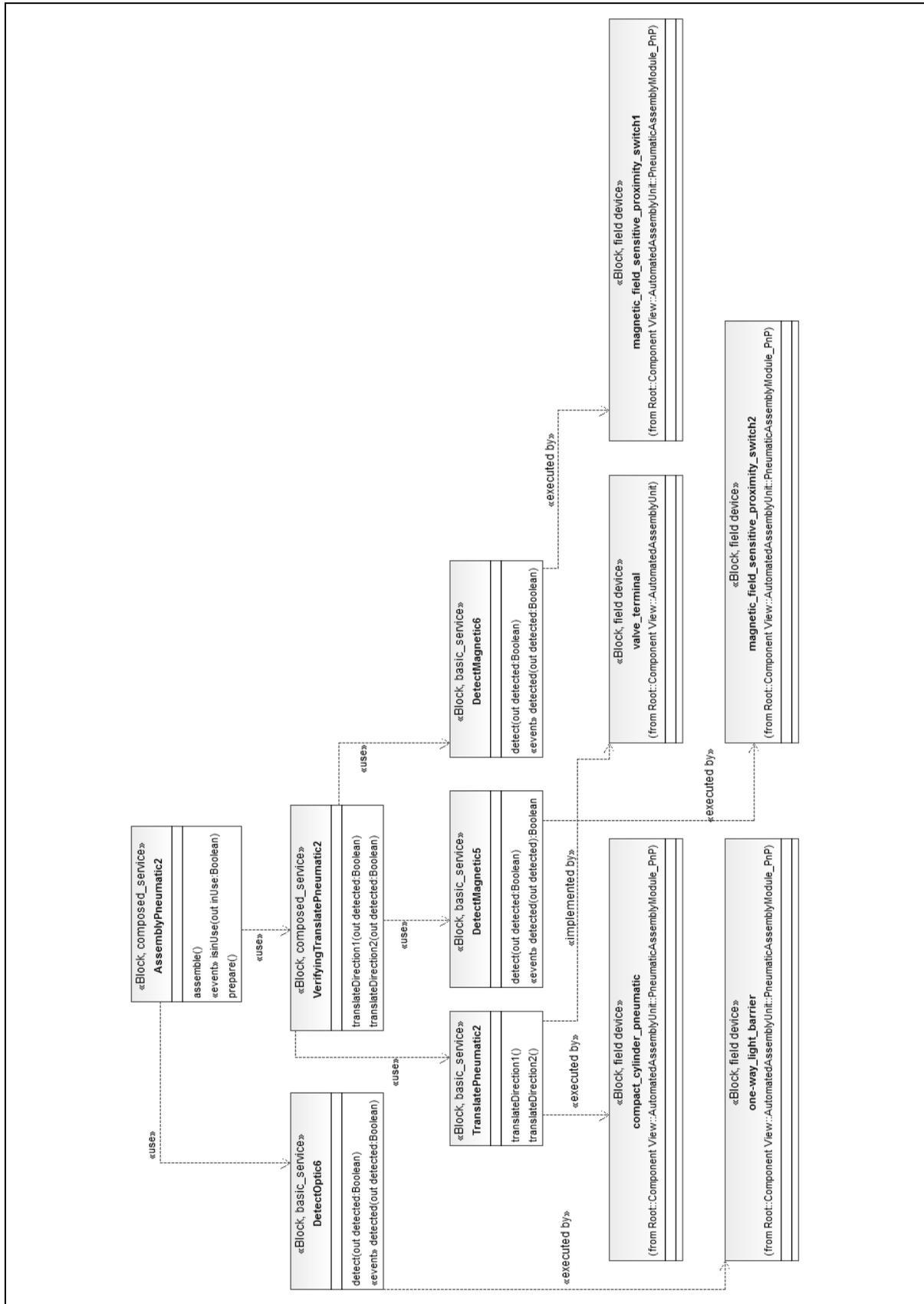
G.4: Service model in SysML: “AssemblyPneumatic1”



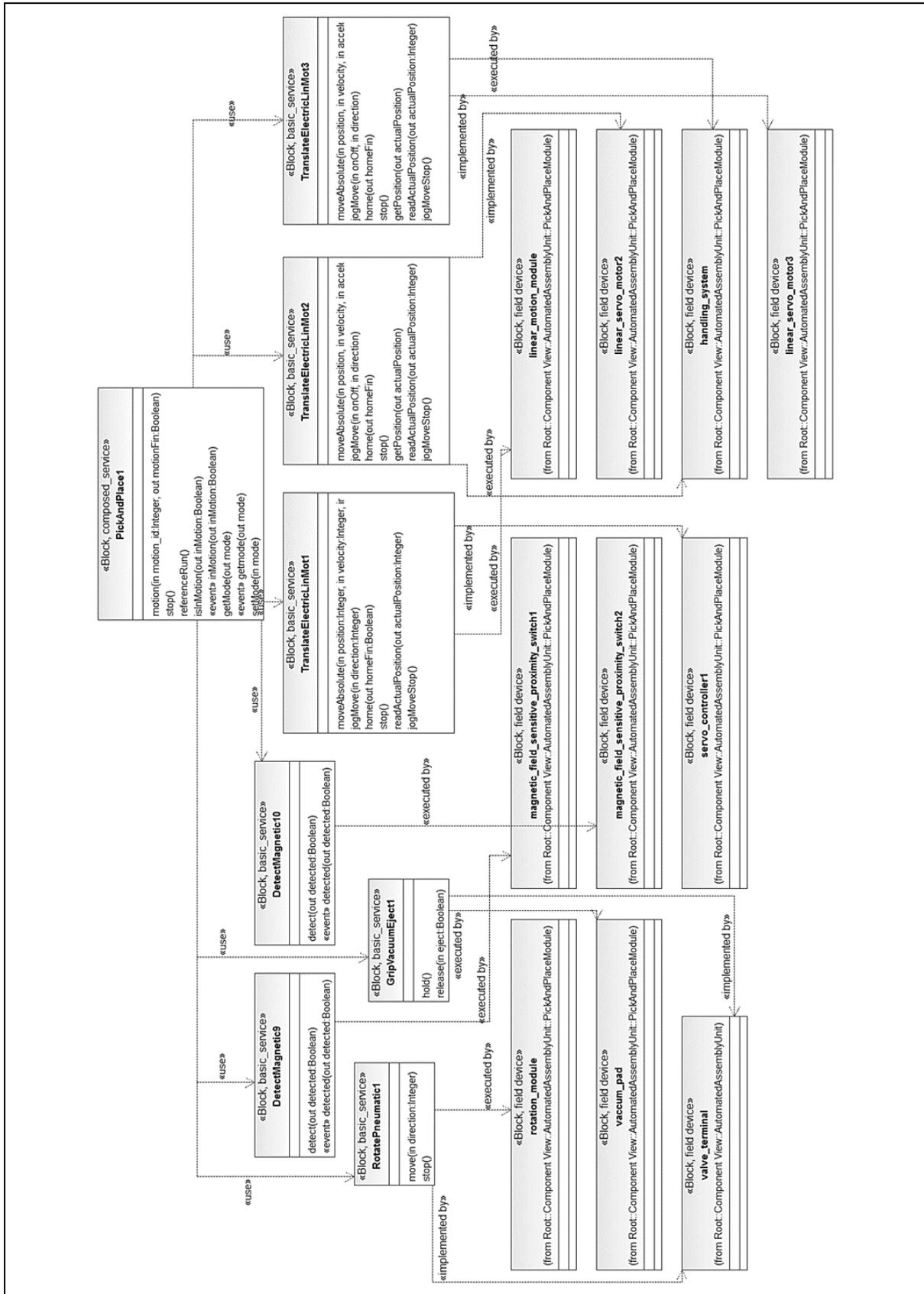
G.5: Service model in SysML: “AssemblyElectric1”



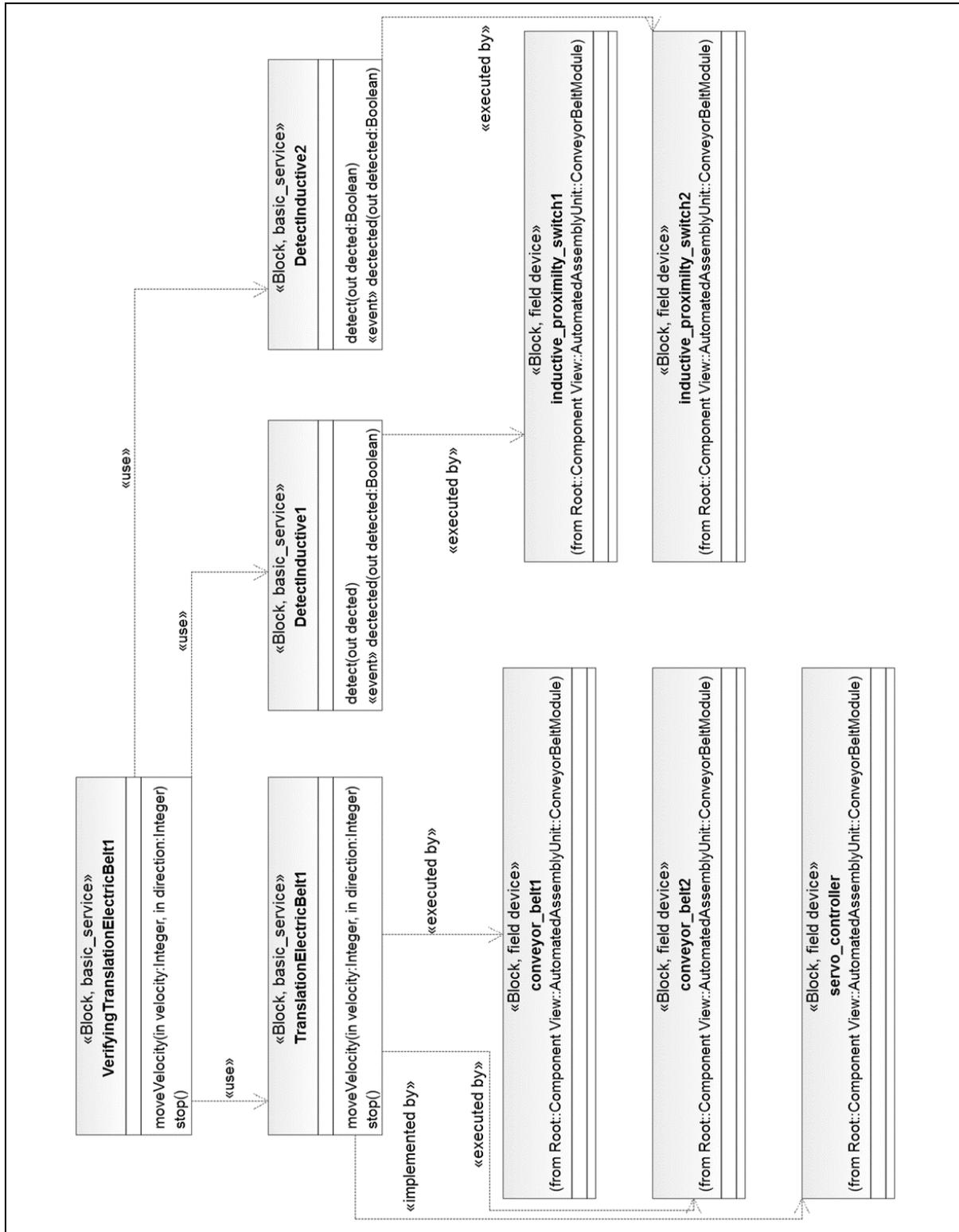
G.6: Service model in SysML: “AssemblyPneumaticPneumatic2”



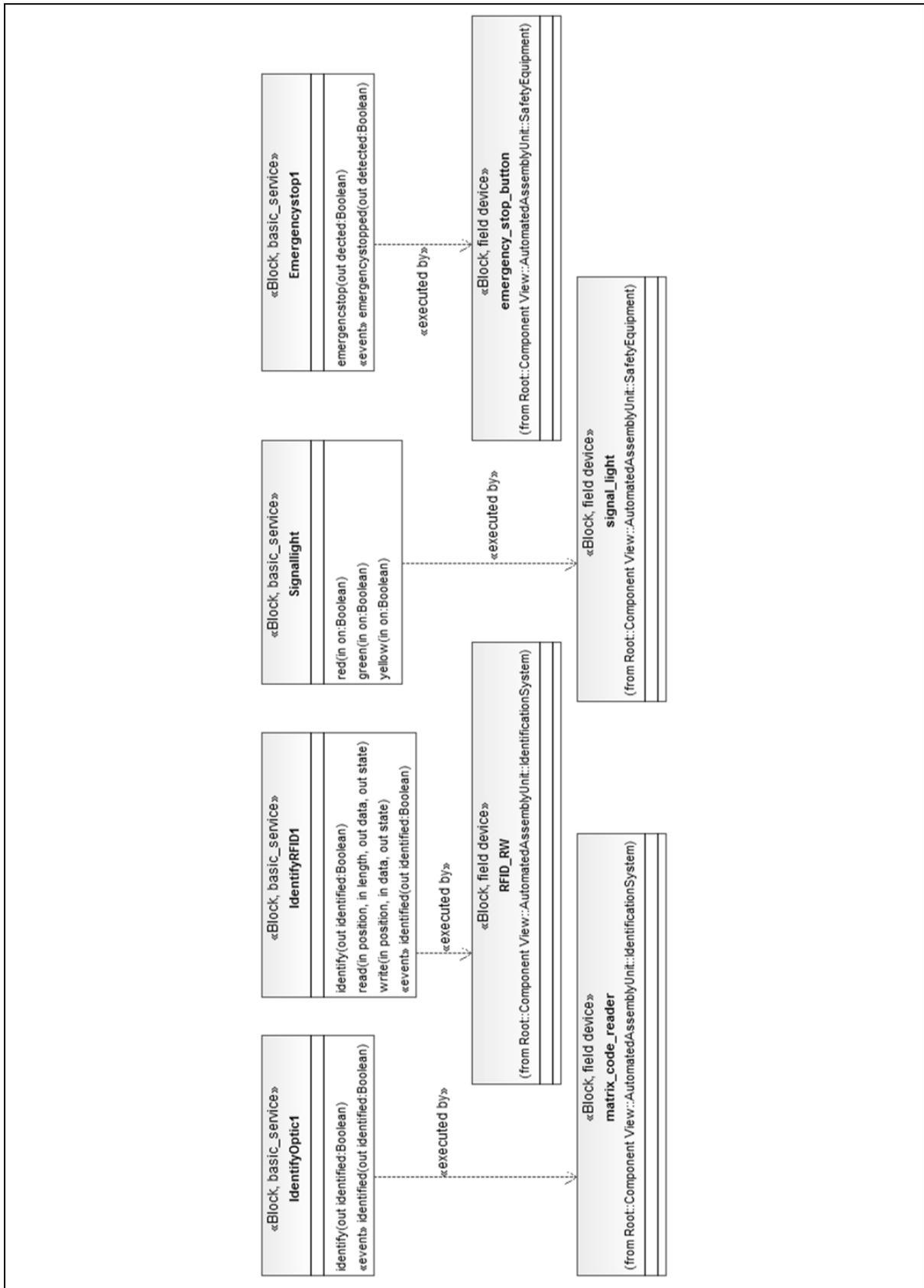
G.7.: Service model in SysML: “PickAndPlace1”



G.8.: Service model in SysML: “VerifyingTranslationElectricBelt1”

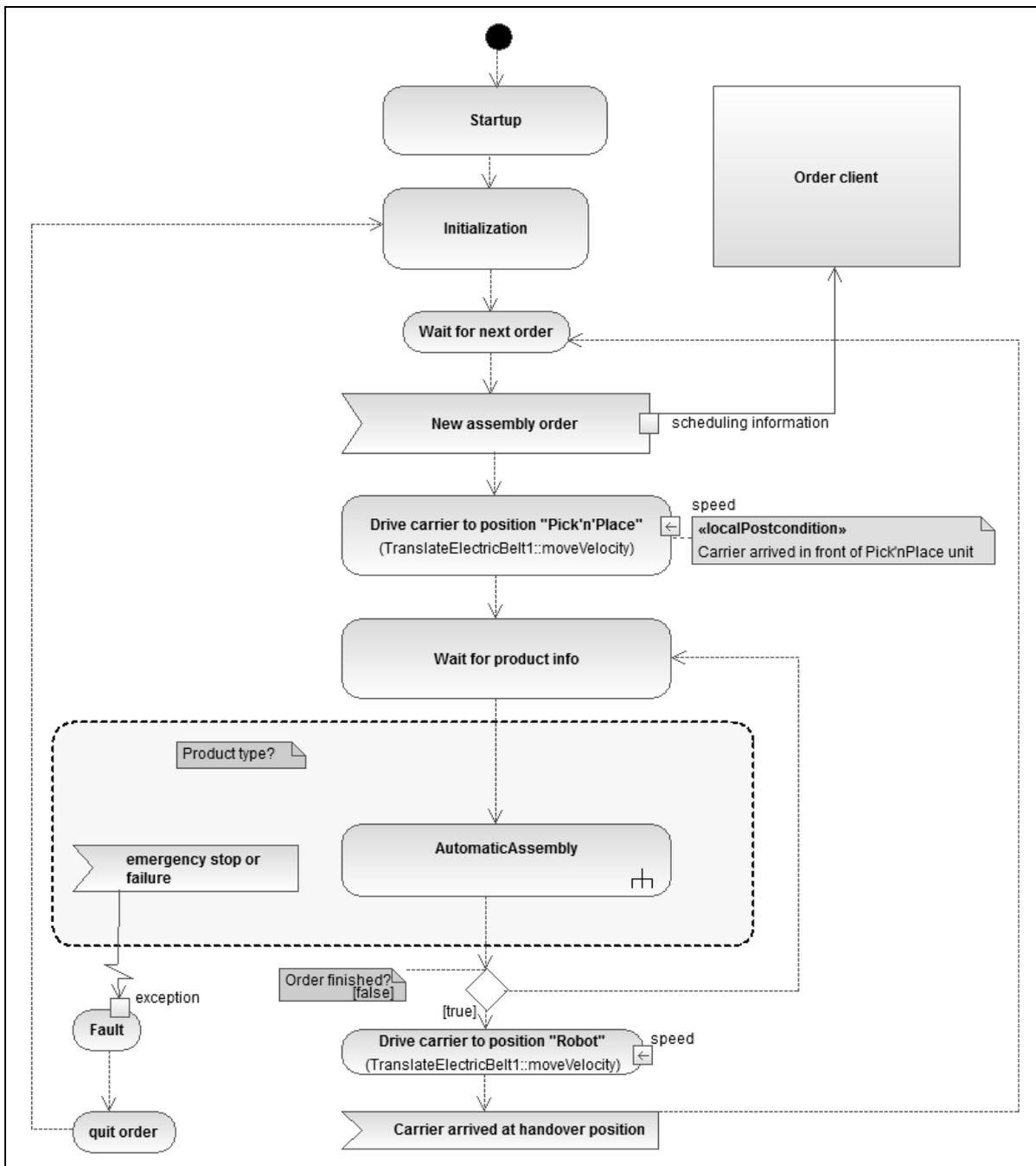


G.9: Service model in SysML: Other basic services

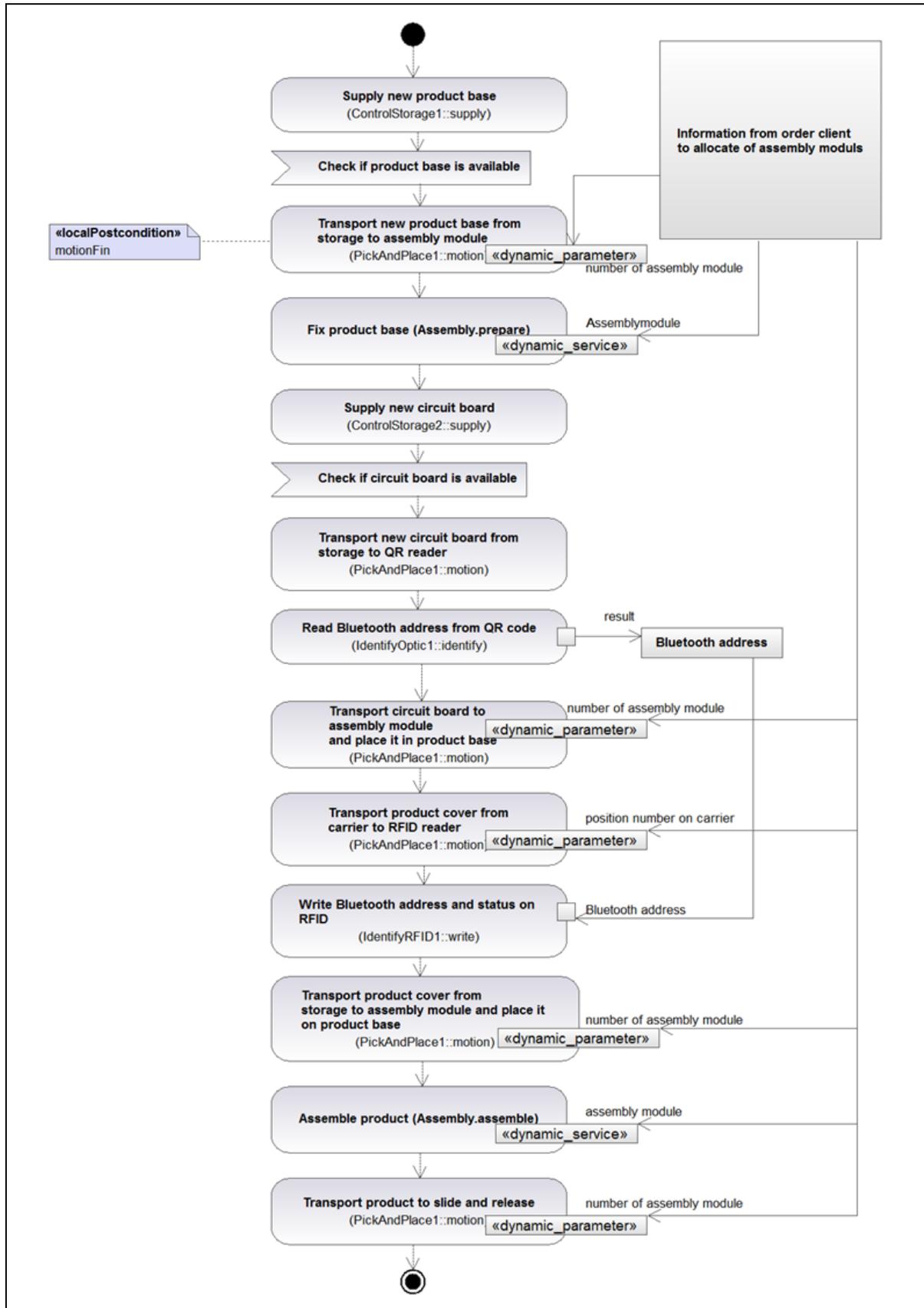


Appendix H: Control Logic Models of Use Case

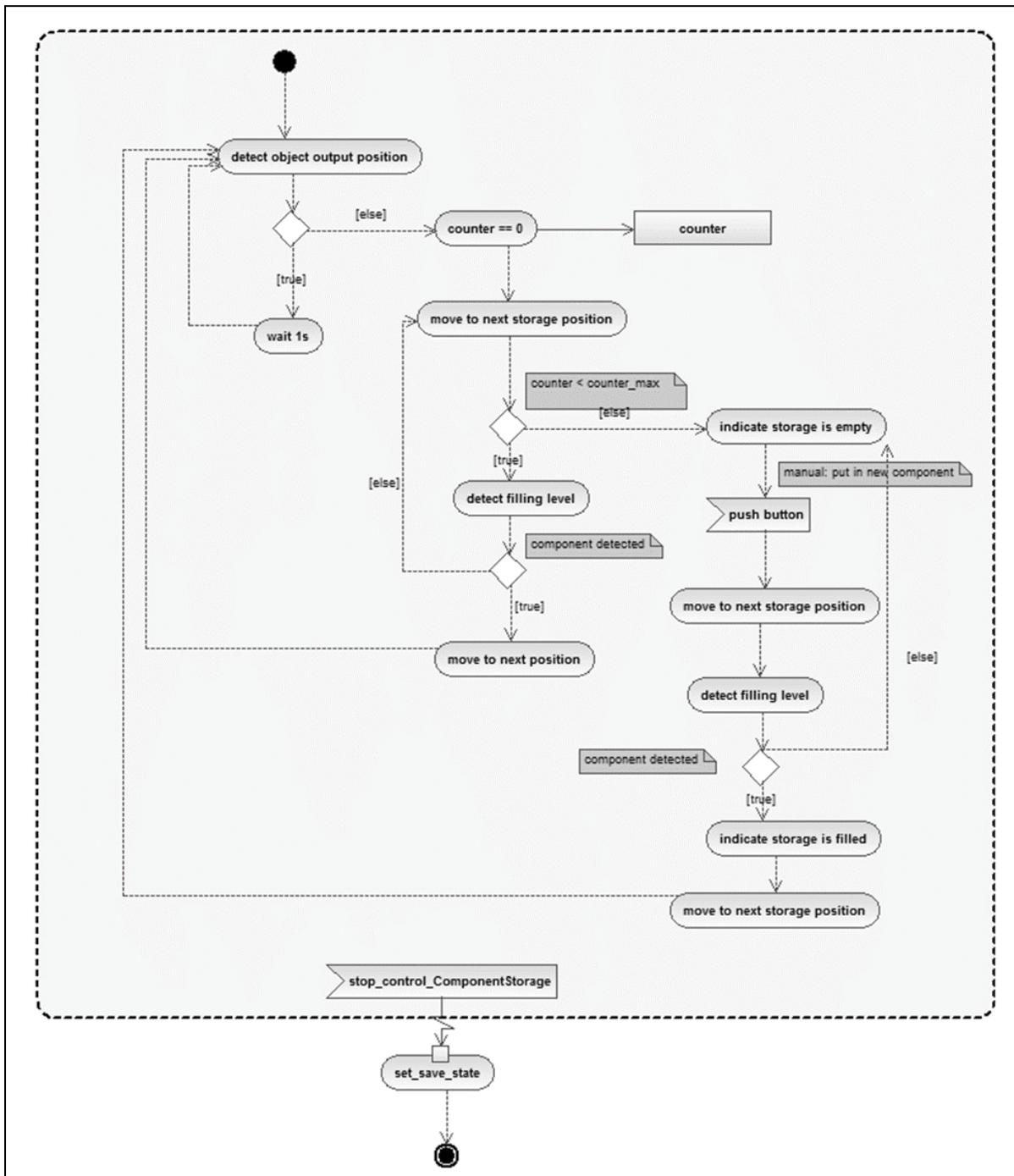
H.1: Control logic model in SysML: Process Service “AutomatedAssembly”



H.2: Control logic model in SysML: Sub process “AutomaticAssembly”

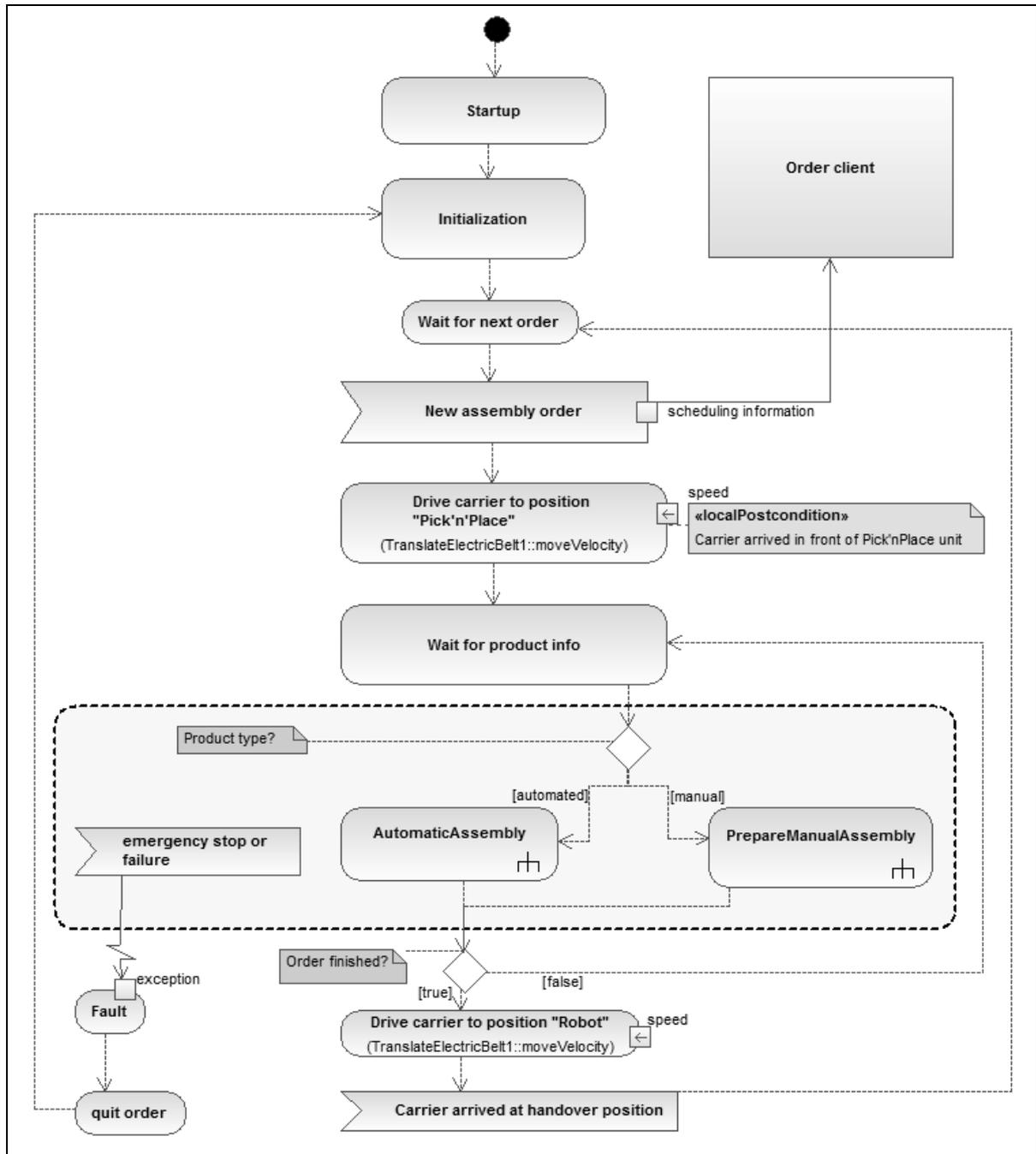


H.3: Control logic model in SysML: Composed Service “ControlStorage1/2”

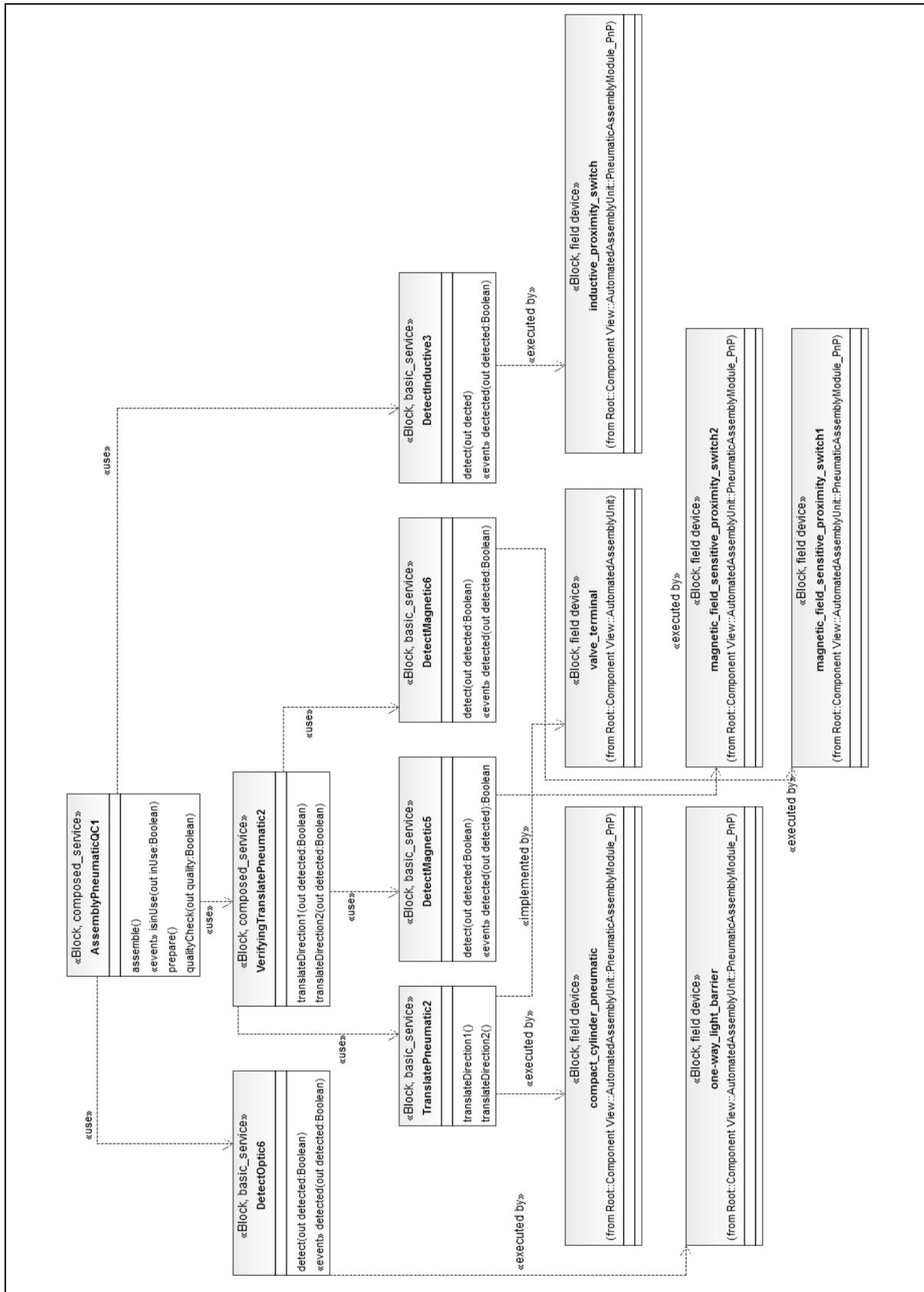


Appendix I: Configuration Tasks

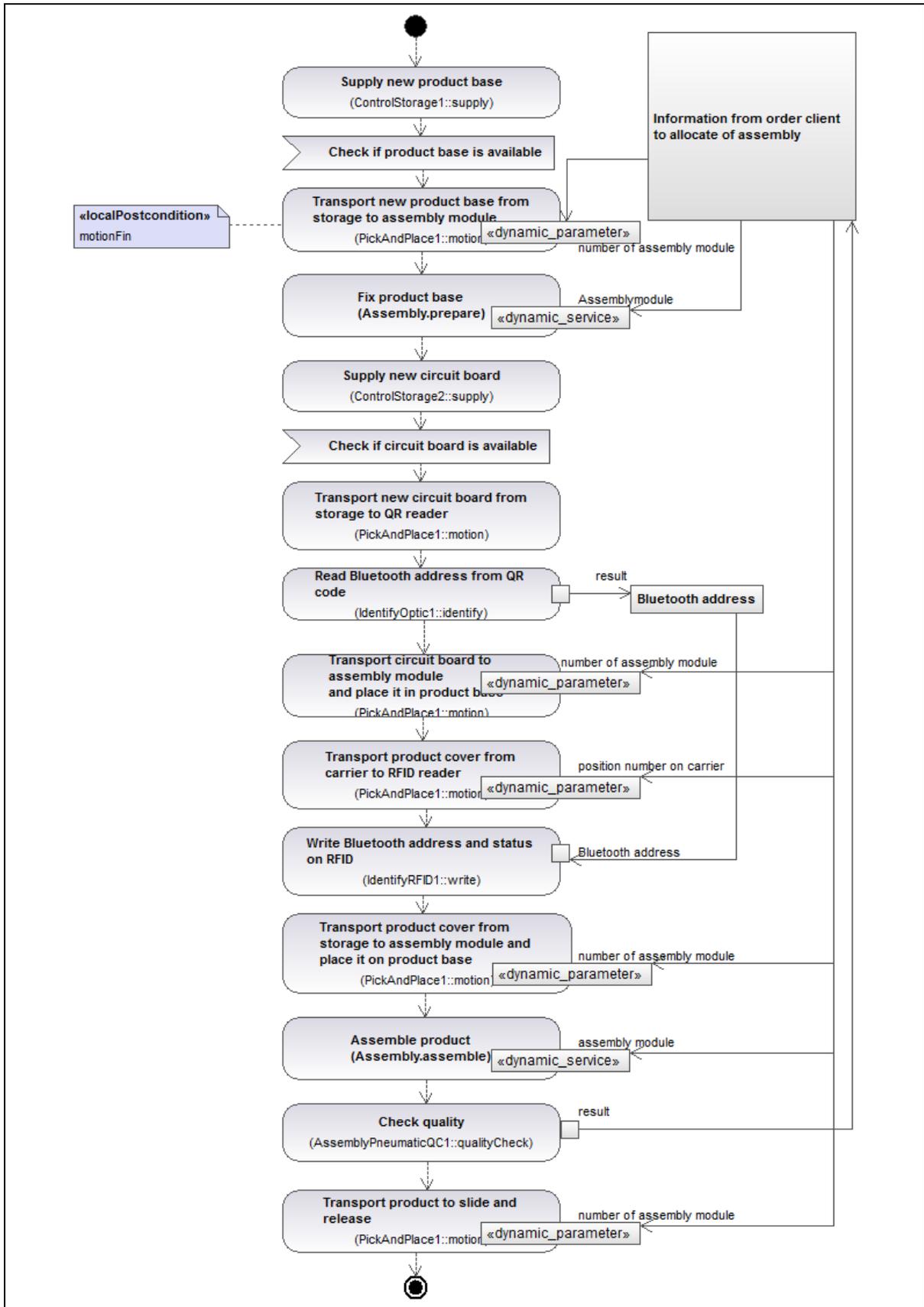
I.1: Modified control logic model of Process Service “AutomatedAssembly”



I.2: Modified service “AssemblyPneumatic2” to “AssemblyPneumaticQC1”



I.3: Modified control logic model of sub process “AutomaticAssembly”



Appendix J: SysML2JG Transformation Rules

Source: XMI file of a SysML/UML Activity Diagram

Target: JGrafchart file

1. The XMI node element of the type “uml:CallOperationAction” is represented by a GCStep element in JGrafchart.
 - 1.1 The value of the `name` attribute of a XMI node element of the type “uml:CallOperationAction” is copied to the `name` attribute of the corresponding GCStep element.
 - 1.2 If a XMI node element of the type “uml:CallOperationAction” has a child of the type “uml:InputPin”, the value of the `name` attribute of the child element is noted in the `actionText` attribute of the corresponding GCStep element and is reported as a “recommended input value”
 - 1.3 If a XMI node element of the type “uml:CallOperationAction” has a child of the type “uml:InputPin”, a `RealVariable` element is created in the *JGrafchart* document. The value of the `name` attribute of the child element is copied to the `name` attribute of the `RealVariable` element.
 - 1.4 If a XMI node element of the type “uml:CallOperationAction” has an attribute named `operation` that contains the ID of an operation of a service, the service name and the operation name are stored in the `ActionText` attribute of the corresponding GCStep element and are reported as “recommended services”, respectively “recommended operation”.
 - 1.5 If a XMI node element of the type “uml:CallOperationAction” has an attribute named `operation` that contains the ID of an operation of a service, a `DPWSObject` element is created and added to the Grafchart. The name of the service is copied to the `name` attribute of the `DPWSObject` element.
 - 1.6 If a XMI node element of the type “uml:CallOperationAction” has a child element `localPostcondition` that has a child element `specification`, a `GCTransition` element is added to the Grafchart as a follower of the corresponding GCStep element. The content of the `value` attribute of the `specification` element is copied to the `actionText` attribute of the `GCTransition` element.
2. The XMI node element of the type “uml:CallBehaviorAction” is represented by a MacroStep or a ProcedureStep element in JGrafchart. If a stereotype “product_service” or “supporting_service” is attached to the behavior of an element of the type “uml:CallBehaviorAction”, the corresponding JGrafchart element is a ProcedureStep, otherwise the corresponding JGrafchart element is a MacroStep.

-
- 2.1 If a `node` element of the type `uml:CallBehaviorAction` is represented by a `MacroStep` element in *JGrafchart*, the corresponding behavior model is implemented in the `MacroStep` element.
 - 2.2 If a `node` element of the type `uml:CallBehaviorAction` is represented by a `ProcedureStep` element in *JGrafchart*, a `Procedure` element is added to the *Grafchart* and the corresponding behavior model is implemented in the `ProcedureStep` element.
 - 2.3 If a `node` element of the type `uml:CallBehaviorAction` is represented by a `ProcedureStep` element in *JGrafchart*, the value of the `name` attribute of the corresponding behavior is copied to the `name` attribute of the `ProcedureStep` element.
 - 2.4 If a `node` element of the type `uml:CallBehaviorAction` is represented by a `ProcedureStep` element in *JGrafchart*, a *JGrafchart*-confirm call to the corresponding `procedure` element is implemented in the `grafcetProcedure` attribute of the `ProcedureStep` element.
 3. The content of the `name` attribute of a XML element of the type `uml:AcceptEventAction` is interpreted as a condition for the transition between the prior element of the `uml:AcceptEventAction` element and its follower. Therefore the content of the `name` attribute is copied to the `actionText` attribute of the `GCTransition` element that is located between the follower and ancestor elements of the `uml:AcceptEventAction` element.
 4. The XML node element of the type `uml:ForkNode` is represented by a `ParallelSplitElement` in *JGrafchart*.
 5. The XML node element of the type `uml:JoinNode` is represented by a `ParallelJoinElement` in *JGrafchart*.
 6. The XML node element of the type `uml:DecisionNode` is represented by two `GCTransition` elements in *JGrafchart*.
 - 6.1 If the XML `edge` elements of the type `uml:ControlFlow` that connects the `uml:DecisionNode` element and its followers have a `guard` element as child, the content of the `value` attribute of the child element is copied to the `actionText` attribute of the corresponding `GCTransition` elements.
 7. The XML `group` element of the type `uml:InterruptableActivityRegion` is represented by a `MacroStep` element in *JGrafchart*.
 - 7.1. All XML `node` elements referred by the `group` element of the type `uml:InterruptableActivityRegion` are modeled within the workspace of the `MacroStep` element in *JGrafchart*.
 - 7.2. If the `group` element of the type `uml:InterruptableActivityRegion` contains a `node` element of the type `uml:AcceptEventAction` that is connected to another `node` element outside the interruptible activity region represented by an `edge` element of

the type “uml:ExceptionHandler”, an `ExceptionTransition` element is added to the Grafchart.

8. The XMI node element of the type “uml:InitialNode” is represented by a `GCInitialStep` element in *JGrafchart*, if the model is transformed to the top-level Grafchart or by an `EnterStep` element or if the model is transformed to a sub-grafchart (e.g., Grafchart of macro step or procedure).
9. The XMI node element of the type “uml:ActivityFinalNode” is represented by an `ExitStep` element in *JGrafchart*, if the model is transformed to a sub-grafchart. Otherwise it is not mapped to the Grafchart.
10. The control flow of the SysML Activity Diagram, which is represented by `edge` elements of the type “uml:ControlFlow” in the XMI file, is transformed to *JGrafchart* by using `GCLink` and `Stroke` elements.
11. If no `GCTransition` element is located between two `GCStep`, `MacroStep` or `ProcedureStep` elements, a `GCTransition` element must be added in order to ensure the executability of the *JGrafchart*. In this case the value of the `actionText` attribute created `GCTransition` element is set to “0”.

Curriculum Vitae

Name: Lisa Maria Ollinger

Geburtsort: Idar-Oberstein

Studium

10/2003 – 05/2009 Technische Universität Kaiserslautern
Studiengang Elektrotechnik
Fachrichtung Automatisierungstechnik, Diplom

Berufserfahrung und Praktika

10/2007 – 03/2008 Porsche AG, Weissach
Praktikantin

10/2008 – 04/2009 Daimler AG, Ulm
Diplomandin

06/2009 – 12/2011 TU Kaiserslautern
Lehrstuhl für Produktionsautomatisierung pak
Wissenschaftlicher Mitarbeiter

01/2012 – 02/2014 Deutsches Forschungszentrum für Künstliche Intelligenz GmbH,
Kaiserslautern
Forschungsgruppe Innovative Fabriksysteme – IFS
Researcher

seit 03/2014 Procter & Gamble Service GmbH
Baby Care, Euskirchen
Technology Leader