Technische Universität München – Fakultät für Maschinenwesen

Model-driven System Architectures for Data Collection in Automated Production Systems

Emanuel Trunzer

Vollständiger Abdruck der von der Fakultät für Maschinenwesen
der Technischen Universität München zur Erlangung des akademischen Grades eines
Doktor-Ingenieurs
genehmigten Dissertation.

Vorsitzende: Prof. Dr. rer. nat. Sonja Berensmeier

Prüfende/-r der Dissertation:

1. Prof. Dr.-Ing. Birgit Vogel-Heuser

2. Prof. Dr.-Ing. Florian Holzapfel

3. Prof. Mag. Dr. Manuel Wimmer

Die Dissertation wurde am 28.05.2020 bei der Technischen Universität München eingereicht und durch die Fakultät für Maschinenwesen am 16.09.2020 angenommen.

Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.ddb.de> abrufbar.

Model-driven System Architectures for Data Collection in Automated Production Systems

Autor:
Emanuel Trunzer

ISBN 13: 978-3-96548-087-2

1. Auflage 2021

sierke WWS GmbH
Sternstraße 7
37083 Göttingen
Tel.: +49 (0)551 5036645

Coverdesign: sierke MEDIA

People under pressure don't work better; they just work faster.

Tom DeMarco, "Peopleware: Productive Projects and Teams," 1987.

Acknowledgments

First of all, I am deeply grateful to Prof. Birgit Vogel-Heuser, who gave me the chance to follow a Ph.D. under her supervision. Her constant encouragement and the possibilities that were given to me contributed significantly to my research. Moreover, I want to thank her for the freedom I was given to develop my topic and to evaluate concepts on the hardware and demonstrators of our institute. During the time of my Ph.D. I was able to learn a lot from her.

Furthermore, I want to especially thank Prof. Florian Holzapfel and Prof. Manuel Wimmer for their agreement to examine my work. Moreover, I want to thank Prof. Sonja Berensmeier for chairing the examination committee.

Many students were involved in my research and contributed significantly, either through fruitful discussions or by active collaboration with me. I always enjoyed and greatly benefited from working with them. I want to thank all of them for their inspiring ideas, efforts, and also the support that I experienced. Among the many students that I supervised, I want to highlight and especially thank Moritz Kohnle, Jan-Kristof Chen, Anne Wullenweber, Pedro Prata, Mathis Pundel, Micha Müller, Thomas Schilling, Bernhard Rupprecht, Simon Lötzerich, Simon Felderer, and Oskar Landenberger.

Throughout my time at the institute, I always enjoyed working with my colleagues. Many discussions, not necessarily on research, constantly motivated me and helped me through all challenges. I would like to take the opportunity to especially thank my office colleague Iris Weiß, as well as Juliane Fischer, on whom I could always genuinely rely.

I am very grateful to Thorsten Pötter, who, after the end of the SIDAP project, offered his mentorship and, in recurrent web conferences, made sure that I ran out of excuses. He always pushed me forward and provided me with the additional motivation that I needed. Furthermore, I want to thank Gary Combes, who played no active part during this thesis, but greatly influenced me in my initial motivation for challenging myself with this thesis. He taught me how to handle challenges and how to overcome them.

Moreover, I want to thank my family for always supporting me and always being considerate, even if I was stressed, occupied or distracted with other things, or hard to reach. They were always a great aid and always motivated me.

Last, I want to thank all involved industrial experts and project partners that enabled me to evaluate this work under industrial requirements and with real-world use-cases.

Table of Contents

1. Introduction

The fourth industrial revolution, called Industrie 4.0 (I 4.0), is pushing the limits of industrial automation and production. Intelligent, autonomous production systems [GB12; LCK16], cloud manufacturing [Zha+12], as well as the Industrial Internet of Things, and big data methods [Bi17; BXW14; XD18] rapidly transform the industry. These new concepts and approaches allow greater production flexibility (lot size one) [Spa13], self-diagnosis, -configuration, and -healing [Bar+15; GB12], as well as closer human-machine interaction [Gor+14].

1.1. Motivation

A significant prerequisite for the realization of these approaches and concepts is better integration of systems and data, as well as improved connectivity of all relevant systems to leverage the ever-increasing amount of generated data [KWH13; VH16]. Before data can be used for the analysis of processes and their optimization, the data needs to be collected and integrated. However, production systems in industrial automation are organized in a hierarchical architecture, called the automation pyramid, only providing limited communication capabilities. This architecture follows the ISA-95 layout [ISA95] and is a result of divergent requirements on the field level and superordinate business levels. This rigid structure of the automation pyramid limits the connectivity of systems [CPC17]. Furthermore, due to the long life cycles of production plants in industrial automation of up to 40 years, a large number of existing legacy systems need to be interfaced and integrated before their data can be used [Bir+10; Vog+15; WSJ17]. Therefore, improved integration and connectivity are not just major prerequisites, but also significant obstacles for industrial adoption of I 4.0 principles.

The identified problems can be manifested based on a questionnaire conducted with industrial experts in the course of the NAMUR Annual General Meeting 2016. The NAMUR is an industrial association representing German operators of chemical plants, as well as equipment suppliers for the process industry. A total of 23 industrial experts working for large German plant operators and component manufacturers were questioned about their assessment of data mining and big data principles in the process industry. One of the questions was related to the main difficulties with data integration (cf. Figure 1). The experts confirmed that the large number of heterogeneous data sources and the variety of interfaces that need to be addressed are significant obstacles for industrial data integration. Furthermore, the experts assessed the high implementation efforts due to a large number of existing and heterogeneous systems as problematic.

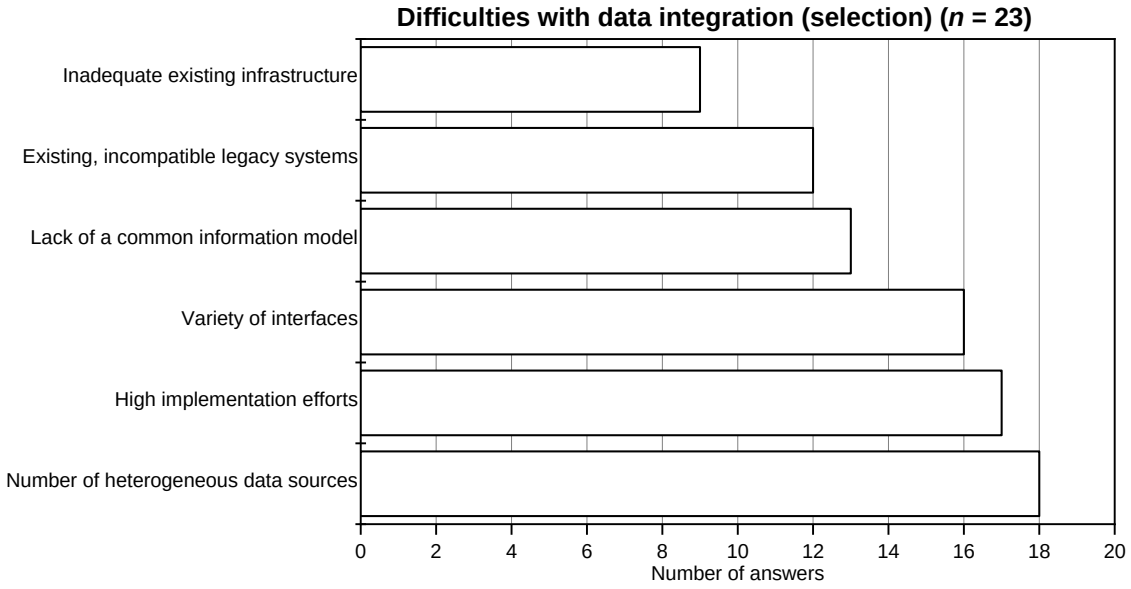


Figure 1: Difficulties with data integration (selection of answers) as given during a questionnaire in the course of a workshop on the NAMUR Annual General Meeting 2016. Total number of participants n = 23.

According to Jardim-Gonçalves et al. [JPG12] there is a lack of accepted system architectures for interoperability and data analysis in industry. Therefore, data buses and system architectures for collection of the data were identified as the most critical enabler of novel I 4.0 paradigms by Raptis et al. [RPC19]. Further, Dotoli et al. [Dot+18] state that reliable communication in heterogeneous systems for data collection and integration is a fundamental challenge in factory automation.

On the other hand, Dotoli et al. [Dot+18] conclude that suitable technologies for integration of systems are already available, but in industrial practice, the significant implementation efforts to interface systems and to collect the data renders the data unused. One additional aspect is the substantial complexity in the development, configuration, and deployment of data collection architectures [JPG12]. Also, Strasser et al. [Str+18] argue that current digitalization trends cause increasing engineering complexity and related implementation costs due to the vast number of systems and interfaces.

Therefore, the reduction of engineering and implementation efforts is one of the foremost priorities for the successful realization of I 4.0 principles in the industry [Dot+18]. Model-driven development of data collection architectures has the potential to significantly decrease manual implementation efforts for their realization [JPG12]. However, the missing formalism for the modeling of networks and the lack of approaches for model-driven architectures are challenges that need to be overcome [PJM12]. This is especially valid as industrial data integration and analysis are characterized by their multi-disciplinary nature [ITK19]. In industrial data analysis, knowledge and re-

quirements from several involved disciplines need to be considered, including engineering information, expert knowledge on the production process, as well as the methods of data analytics [ITK19; Vog+14b]. Despite the multi-disciplinary character, He and Xu [HX14], as well as Penas et al. [Pen+17], identified a lack of interdisciplinary modeling techniques.

In this thesis, a model-driven approach for the development of data collection architectures, which addresses the identified industrial problems, is developed. Therefore, a generic proposal for data collection architectures is presented, serving as a basis for future implementations. Furthermore, a domain-specific language with a graphical modeling notation and supporting metamodel for interdisciplinary modeling of these architectures is conceptualized and evaluated in several use-cases. Based on the formalized models, a model-driven toolchain that allows the automatic generation of data collection architectures to minimize manual implementation efforts is developed.

Throughout the thesis, the term *architecture* is defined as the connection of systems that enables the sharing of data and services. Every system that is part of the architecture is referred to as a *participant* [Tru+19c]. Furthermore, the author of this dissertation defines the term *data collection architecture* as an architecture for the collection and integration of data from multiple *participants*. A *data collection architecture* generally consists of the *communication architecture*, related to the communication functions that allow the transfer of data between participants, and the *application-specific logic in each participant* that generates, forwards, manipulates, or actively uses the data.

1.2. Hypotheses

Based on the identified challenges concerning data collection in industrial automation, this thesis aims to provide a solution for model-driven generation of data collection architectures. Therefore, the following hypotheses **(H1)** to **(H3)** will be investigated:

(H1)	A technology-neutral concept for a data collection architecture can bridge operational technology (OT) and information technology (IT) and allow data collection from production systems.
(H2)	A special domain-specific language with a graphical notation for data collection architectures supports the understanding and structuring of information during the engineering phase of these architectures by multi-disciplinary teams composed of engineers, IT architects, programmers, process experts, and data analysis.
(H3)	A model-based approach for automatic generation of data collection architectures reduces the effort for implementation and migration of these architectures.

The contents and contributions of this dissertation are based on previous publications by the author, namely [Fol+17; TLV18; Tru+17; Tru+19a; Tru+19b; Tru+19c; Tru+20a; Tru+20b; TWV20; Vog+20]. A short summary of the contributions and contents of the respective publications is given in the following:

[Fol+17] Motivation of the relevance of data analytics and data collection/integration for process industries based on industrial problems and possible solutions.

[Tru+19a] Proposal of an industrial data analytics process model for the process industry. Emphasize on the relevance of interdisciplinary teams during the analysis process, as well as the crucial role of proper data collection and preparation in industrial use-cases.

[Tru+20b] Overview and summary of system architectures for data integration in the scope of I 4.0. Derivations of requirements and practical implications based on industrial boundary conditions.

[Tru+17] First publication on the architecture concept for data integration. Conceptual application of the architecture with multi-disciplinary experts and expert evaluation.

[Tru+19c] Comparison of the architecture proposal with other relevant approaches in the scope of I 4.0 with co-authors from the BaSys4.0 and PERFoRM projects. Mapping of the respective system architectures based on divergent requirements in the respective projects, and derivation of a generic architecture proposal applicable to a wide variety of use-cases.

[TLV18] Follow up of architecture proposal with a more detailed overview of the concept. Furthermore, prototypical implementation and feasibility study using a lab-scale use-case, including the xPPU demonstrator [Vog+14c]. Support for the MQTT and OPC UA protocols.

[Tru+20a] Comparison of various protocol-specific architecture approaches for data collection in literature. Moreover, filling the gap between specific implementations and abstract reference architecture by deriving first architecture patterns. Prototypical implementation using a lab-scale setup with the myJoghurt demonstrator (see Section 7.3) based on a Data Distribution Service.

[Tru+19b] Characterization and comparison of relevant protocols for the Industrial Internet of Things characterization. Proposal and prototypical implementation of a technology-neutral software framework with unified interfaces. Evaluation in a lab-scale use-case with support for AMQP and Kafka and comparison of implementation effort compared to P2P architecture for initial deployment and a migration scenario.

[TWV20] Graphical modeling notation for data collection architectures with system and data flow viewpoints, as well as a data mapping table. Application of the graphical language to three industrial use-cases and evaluation with industrial experts.

[Vog+20] Introduction of the underlying metamodel structure to yield a domain-specific language. In this publication, tailored to another version of the graphical modeling notation that describes the combination of real-time aspects and data analytics for industrial automation, so-called hybrid distributed networked control systems.

1.3. Structure of this Dissertation

This thesis follows a design science approach [Hev+04] and is structured as follows: Chapter 2 (p. 7) introduces and specifies the field of investigation. In Chapter 3 (p. 27), the requirements for an industrial data collection architecture are derived from industrial practice and current research. Based on these requirements, the state-of-the-art in system architectures, modeling notations, and model-driven system architectures are reviewed in Chapter 4 (p. 35), and a research gap is identified. Chapter 5 (p. 57) presents the developed approach that aims to fill the research gap. The implementation of the approach is described in Chapter 6 (p. 97). It is followed by Chapter 7 (p. 103), which captures the evaluations that were performed to assess the suitability of the approach. The chapter is split into six Sections: the first two describe the results of industrial case-studies with industrial experts that evaluate the feasibility and quality of the developed architecture concept and the graphical notation. Afterward, a prototypical implementation on a lab-scale is performed to assess the model-driven generation of the communication architecture and to compare it to manual software development. The next Section demonstrates the scalability of the developed model-driven approach by applying it for an industrial use-case. Section 7.5 examines the implementation efforts of classical software development and the model-driven approach based on an extrapolation case-study. Chapter 7 closes with an expert questionnaire on a comparison between the developed approach and current industrial practice. Chapter 8 (p. 143) assesses if the developed approach is capable of adequately addressing the derived requirements. A summary and an outlook on future research directions are presented in Chapter 9 (p. 145). Figure 2 reflects the structure of this thesis graphically.

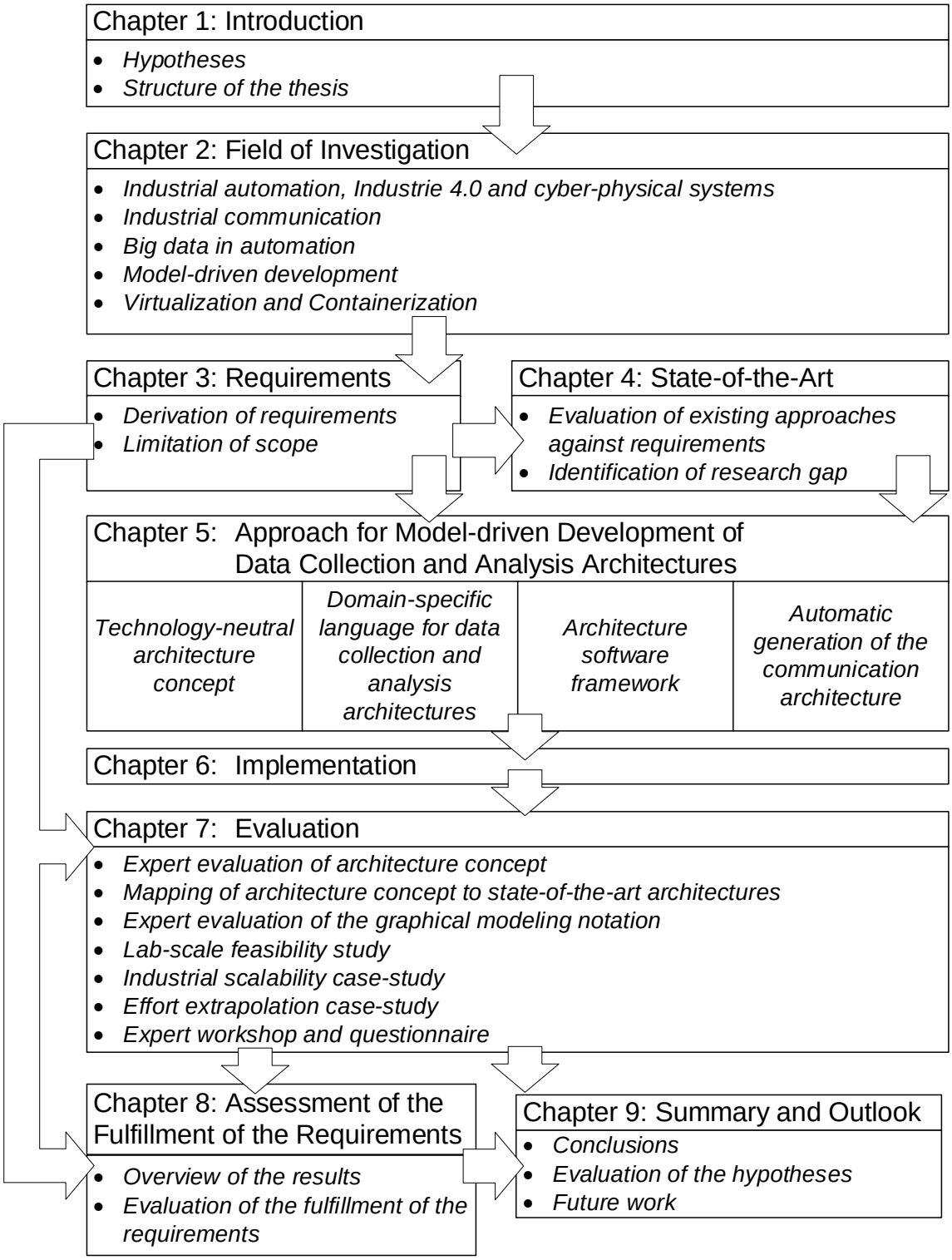


Figure 2: Overview of the structure of this dissertation.

2. Field of Investigation

The presented approach was developed for the area of data collection for data analysis applications in automated industrial production and Industrie 4.0. In this Chapter, the specifics of the domain and related aspects are introduced to provide a better understanding of the definitions and the particular requirements from the field of application.

First, an introduction to the domain of industrial automation and related terms is given. Afterward, the concepts of I 4.0 and the Industrial Internet of Things, which disrupt the classical organization and challenges of industrial automation, are introduced. Furthermore, an introduction to a new class of production systems, called Cyber-physical Production Systems, and reference architectures for I 4.0 are given. As communication is a central aspect of interfacing systems and collecting their data, an overview of communication technologies on the field level and the superordinate levels is presented. On both levels, a multitude of different technologies evolved and complicate the interfacing. As data collection is a challenging pre-step for subsequent data analysis, the basics of big data and data mining in automation are introduced. An introduction to the concepts of model-driven development follows, which employs modeled information and model transformations to decrease manual effort during software development. At last, the basics of virtualization and containerization as recent trends in IT are presented.

2.1. Industrial Automation

The aim of automating technical processes characterizes the field of industrial automation. A technical process, in general, is a process that manipulates the state of a material, energy, or information. Technical processes are executed in technical systems and can be automated with process automation systems. If the automated technical process is a production process, one can speak of an automated production system (aPS). The composition of a process automation system is depicted in Figure 3. Process automation systems contain three subsystems with close interaction between them. On the lowest level, the technical system that executes the technical process can be found. The technical system receives actuator signals for the control of the process from a supervisory computing and communication system. The technical system forwards sensor signals from the technical process to the computing and communication system. Humans interact with the computing and communication systems over human-machine interfaces (HMIs) for process control and get feedback on the process result [LG99b].

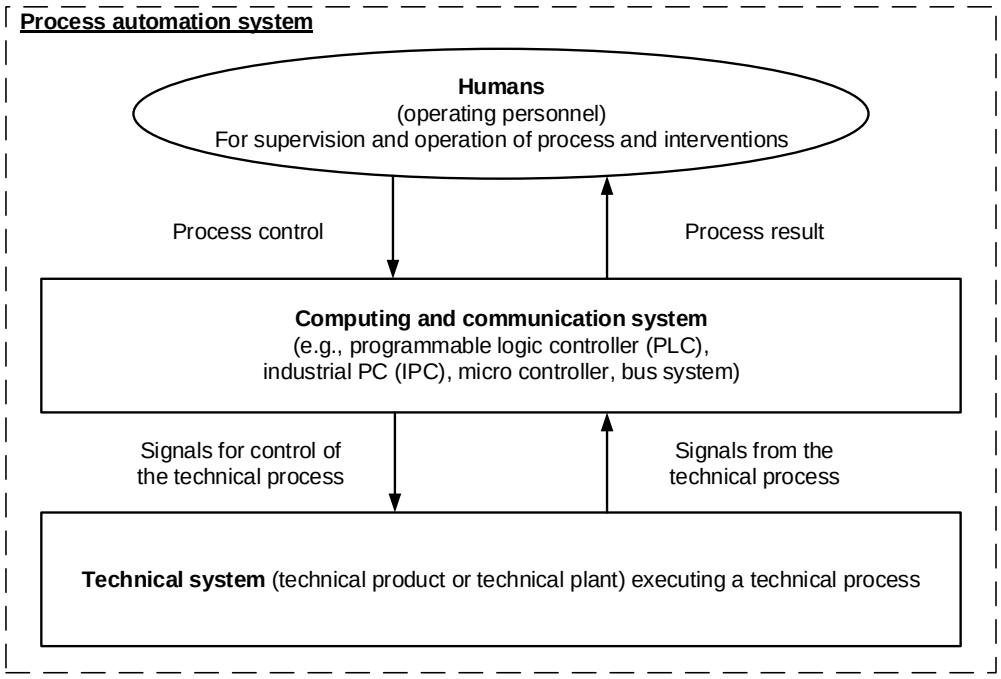


Figure 3: Structure of process automation systems (adapted from Lauber and Göhner [LG99b]).

Sensors and actuators are used for interaction with the technical process. While sensors can measure physical quantities and convert them to electrical or optical signals, actuators influence the physical quantities of the technical process. Different ways to realize the coupling between these sensors/actuators and the computer and communication system are depicted in Figure 4.

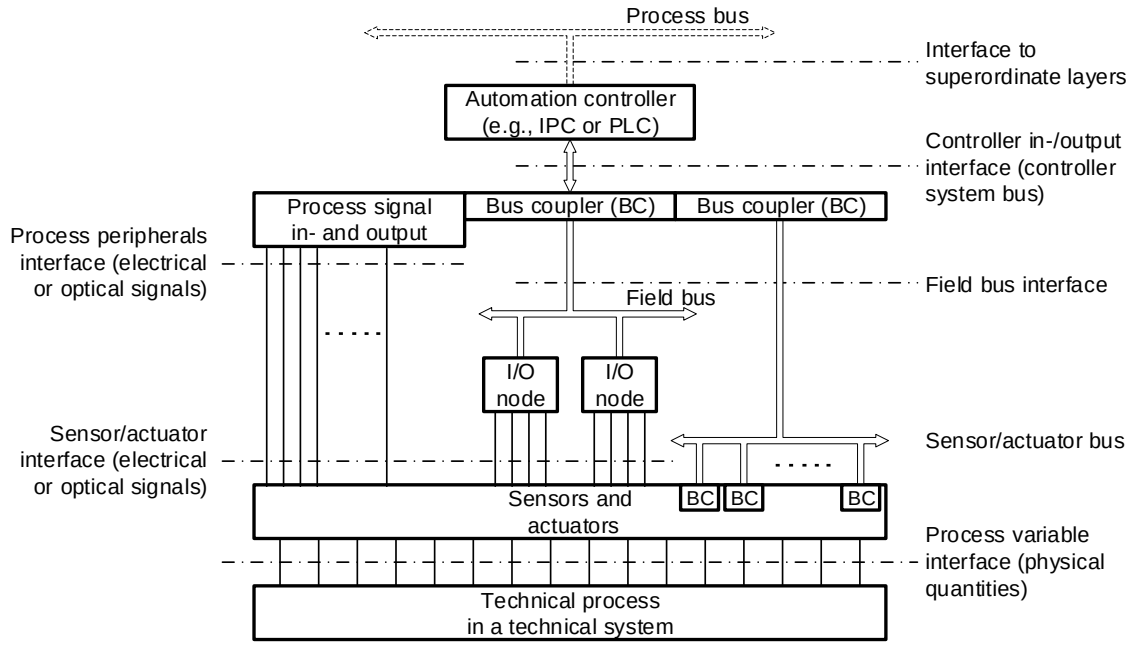


Figure 4: Installations for coupling an automation controller with a technical process, including relevant interfaces (adapted from Lauber and Göhner [LG99b] and VDI/VDE guideline 3687 [VDI3687]). A direct connection between sensors/actuators and automation controller (left), field bus connection using a bus coupler and decentralized I/O nodes (middle), and decentralized intelligent sensors with direct bus access (right).

In Figure 4, the computer and communication system consists of several parts: a central automation controller that controls the technical process, a communication system connecting the automation controller and the sensors/actuators, and a process bus that enables the communication between the automation controller and other systems. In aPS, the automation controller is typically realized either as a programmable logic controller (PLC) or as an industrial PC (IPC).

The simplest form of coupling the automation controller and the sensors/actuators is a direct connection using multiple cables. This type of interfacing is often found in small-scale aPS, where the automation controller is located close to the sensors and actuators, only requiring short cables [SHW99].

In larger aPS, a direct connection between the automation controller and sensors/actuators is often not feasible due to lengthy and expensive wires and increased risks of interferences. Here, a field bus with decentralized in- and output (I/O) nodes can be a solution [SHW99]. These I/O nodes have a direct connection to the respective sensors/actuators and communicate with the automation controller over a field bus. Various field buses with different feature sets and characteristics exist, which will be introduced in Section 2.3.1.

Intelligent sensors and actuators often include a bus interface. They allow a completely decentralized structure of the coupling between sensors/actuators and automation controllers. This type of coupling is often found in very large-scale aPS with the need for decentralized processing of signals, e.g., the process industry.

The process bus interface from Figure 4 allows the aPS to be embedded into larger automation systems. These include besides the aPS for controlling a technical process also enterprise functions needed for coordination and supervision of complex production processes. As the requirements in the application domains differ significantly (real-time control in aPS, large amounts of data in enterprise functions), a layered architecture, called the automation pyramid, is prevalent in industrial practice [SHW99]. The hierarchical automation pyramid structure is standardized in ANSI/ISA-95 [ISA95] / IEC 62264 [IEC62264]. A graphical representation of the automation pyramid with its levels and the related, divergent requirements is given in Figure 5.

On the field level, aPS with automation controllers (level 1 of the ISA-95 structure) automate and control a technical production process (level 0). Several aPS or large-scale aPS are often coordinated by a SCADA (supervisory control and data acquisition) system on level 2. The process level consists of manufacturing execution systems (MES, level 3) that monitor the production process, store historic data on the quality of manufactured products, and manage the distribution of open manufacturing orders to suitable aPS. On the highest level, the so-called operational level (level 4),

an enterprise resource planning (ERP) system coordinates the production across multiple production sites and calculates key performance indicators (KPIs) to assess the overall production performance. Furthermore, ERP systems include the long-term planning of inventories, as well as production schedules.

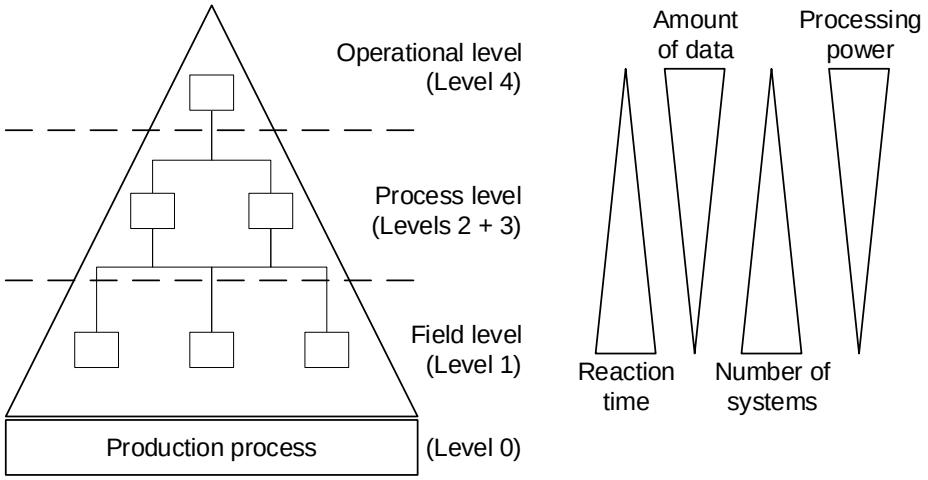


Figure 5: Automation pyramid structure and requirements for the communication and processing system (adapted and extended from Scherff et al. [SHW99]and Lauber and Göhner [LG99b]).

On the lowest levels of the automation pyramid, a technical production process is supervised and controlled. This requires a concise reaction in real-time. Furthermore, the number of systems is vast as multiple aPS can be part of a production site. The number of systems decreases on the higher levels as multiple subsystems are supervised and controlled from an upper level. However, this causes an increase in the amount of data processed by superordinate systems compared to the lower levels.

The field level is characterized by sensors/actuators and PLCs with a low computational performance that are connected via real-time field buses. The hardware and software systems on the lower levels are referred to as OT. This includes a large number of existing legacy systems with long life cycles, heterogeneous interfaces, and limited communication capabilities. In contrast, the upper levels are part of the IT and often consist of high-performance servers and office computers connected with Ethernet networks. The clear separation between the defined layers allows hardware and software providers to focus on the respective set of requirements. However, this separation also causes, by definition, that data is only allowed to be exchanged across two adjacent levels of the automation pyramid.

The rigid structure of the automation pyramid is increasingly questioned and extended due to new demands related to flexibility and decentralized intelligence. New trends, such as industrial Ethernet, middleware, or the concept of I 4.0, stimulate the evolution of the underlying system architecture towards higher flexibility and data availability [Rie+14b; Sau07; Sau10]. One example of

the evolution process already found in industry is the so-called automation diabolo introduced by Vogel-Heuser et al. [Vog+09] (cf. VDI guideline 5600 [VDI5600] and Section 2.3.2). Therefore, to include data collection from existing legacy systems that are still part of the classical automation pyramid is a challenge for the data collection process in industrial automation.

2.2. Industrie 4.0 and Industrial Internet of Things

The concept of I 4.0 was first introduced in 2011 by the German Industrie 4.0 working group as part of the strategic initiative Industrie 4.0 of the German government [Boy+18; KWH13; VH16]. It describes the idea of a fourth industrial revolution, after the initial mechanization (first revolution), the introduction of assembly lines (second revolution), and the digital automation by PLCs in the 1970s (third revolution). Industrie 4.0 incorporates the global leveraging of data and the full connectivity of systems for individualized production, optimized decision making, and increased resource efficiency. The following prerequisites were identified for the realization of I 4.0 concepts [KWH13]:

- horizontal integration of systems and data through value networks,
- end-to-end digital integration of engineering across the entire value chain, and
- vertical integration and networked manufacturing systems.

While the last point relates to improved integration across the levels of the automation pyramid, the two others include the closer cooperation of parties along whole value networks and an enhanced digital and integrated engineering of systems. This overall integration causes a convergence of IT and OT through new technologies. Two main enablers for the realization of I 4.0 are the Industrial Internet of Things (IIoT) and Cyber-physical Systems (CPS) [Ban+16; Mon+16].

The IIoT describes the industrial application of Internet of Things (IoT) technology found in consumer electronics. The term IoT encompasses an information network of physical objects that closely interact and cooperate to reach a common goal [Jes+17; Wor15]. The IIoT adapts these principles considering industrial requirements. It describes the seamless connectivity of all systems involved in the manufacturing process to create a digital or virtual factory. This increased connectivity offers new chances for data collection using IIoT technology.

2.2.1. Cyber-physical Systems

A CPS, in general, is a physical system that includes enhanced computing and communication capabilities to monitor, coordinate, control, and integrate operations [GB12; Raj+10; VBF12]. Lee et al. [LBK15] introduced the so-called 5C architecture that describes the system architecture of

Cyber-physical Production Systems (CPPS). The architecture defines the internal composition of CPPS and separates the distinct aspects into five layers:

- the smart connection layer that includes communication with sensors/actuators, as well as other systems;
- the data-to-information conversion level that includes monitoring of machine health and quality using data analytics;
- the cyber level that represents a digital twin including relevant data describing the system;
- the cognition level with enhanced functionalities related to visualization, decision making, or integrated simulations; and
- the configuration level that provides self-X (configuration, adjustment, optimization).

The integration of multiple CPS into a larger system is called a cyber-physical system of systems (CPSoS) [Fer+18]. Figure 6 gives a typical CPSoS network structure with technologies spanning from OT to IT and including a multitude of different communication links and systems, all cooperating to fulfill a given manufacturing task. While these new principles question the classical automation pyramid, the heterogeneous mix of technologies and systems is still characteristic for CPSoS.

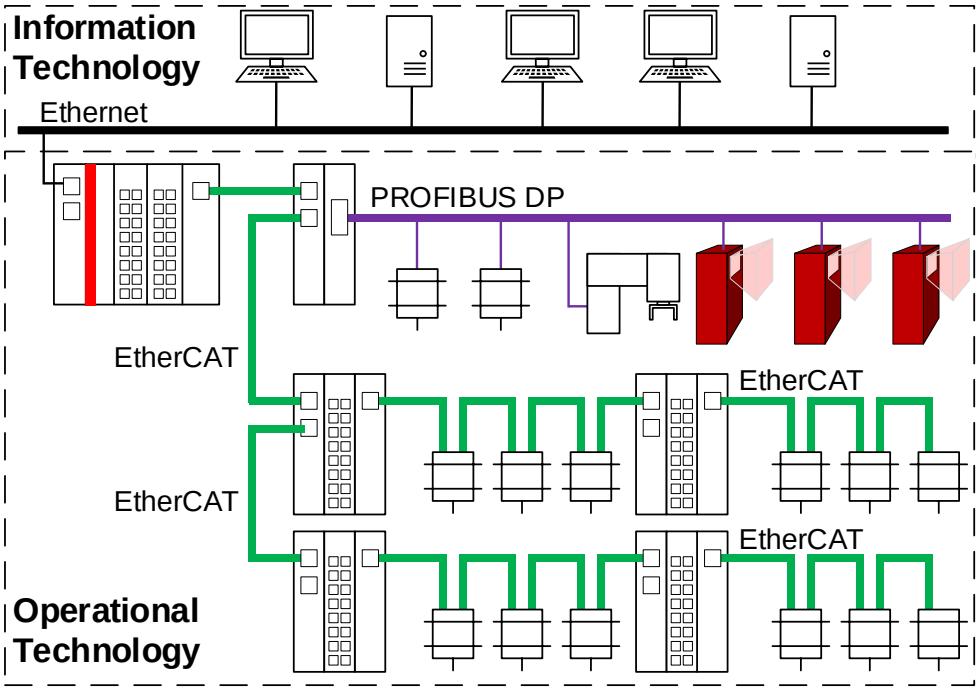


Figure 6: Simplified network layout of a typical CPSoS consisting of IT and OT domains with various types of connected devices and networks (Trunzer et al. [TWV20]).

Interoperability and connectivity of CPSoS, as well as system architectures that allow the integration of CPSoS, were identified as grand challenges for enterprises in the future [Pan+19].

2.2.2. Reference Architectures

Reference architectures describe an abstract view of a real system and give recommendations for a successful realization of system architectures. Furthermore, they include a common vocabulary as well as technology- and implementation-neutral basic guidelines for the design of architectures. In the course of I 4.0 and the (I)IoT, serval international and national standardization bodies and industrial consortia actively work on the definition of reference architectures.

For instance, the Reference Architecture Model Industrie 4.0 (RAMI 4.0) defined in DIN SPEC 91345 [DIN91345] describes a layered architecture along three axes. A visual depiction of RAMI 4.0 is given in Figure 7 with the three axes *life cycle and value stream*, the *hierarchical levels of the system* aligned with ISA-95/IEC 62264, as well as *the architecture layers*. RAMI 4.0 targets the industrial manufacturing and production domain.

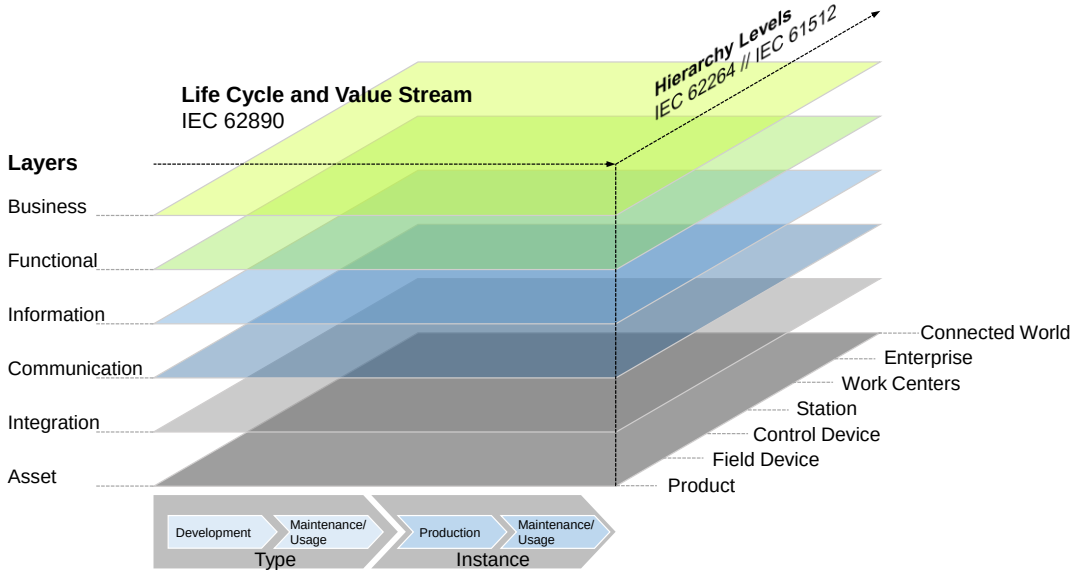


Figure 7: Graphical representation of the Reference Architecture Model Industrie 4.0 (RAMI4.0) (DIN SPEC 91345 [DIN91345]) (Trunzer et al. [Tru+20b]).

Other reference architectures exist with the American Industrial Internet Reference Architecture (IIRA) [Ind17b], the IEEE Architectural Framework for the Internet of Things [IEEE2413], as well as the Internet of Things Reference Architecture (IoT RA) standardized in ISO 30141 [ISO30141]. In contrast to RAMI 4.0, both reference architectures follow a general approach for multiple domains and are not limited to manufacturing and industrial automation [Ind17a].

The integration of existing systems is possible with all three reference architectures. However, the focus of all three lies in the abstract definition of architectures for I 4.0 and (I)IoT, and therefore

guidelines for the integration of legacy systems are lacking. While initial developments and deployments are characterized as greenfield scenarios, the consideration and need for the integration of existing systems are typical for so-called brownfield scenarios [Kag15]. With the NAMUR Open Architecture (NOA) (see Figure 8) [Cai+19; Cai18; Kle+17; NE175], a reference architecture proposal for brownfield scenarios in the field of chemical process industry exists. This domain is especially characterized by very long lifetimes of plants of up to 40 years and constant retrofitting and updating of the installed base [Bir+10; Vog+15].

Existing control systems for deterministic control from the automation pyramid are part of the *core process control*. In brownfield systems, this part is often already existing and controls a mission- and safety-critical production process. Additional functionalities for monitoring and optimization (M+O) of plants, e.g., by enhanced data analytics or dashboards, reside outside and separated from this core part. Furthermore, NOA foresees the retrofitting of plants with additional sensors to increase the monitoring capabilities. As these sensors are often not needed for the main control of the plant, they also reside outside of the core process control to prevent interference. The connection between M+O systems and the deterministic control systems is realized as a secondary communication channel that only allows the flow of data from the core process control to M+O systems. This retrofitted communication channel, therefore, completely separates the two domains and allows data exchange across multiple levels of the automation pyramid. Information can be sent back into the core process control via a distinct channel that includes a so-called verification of request to ensure the secure origin and intent of the input.

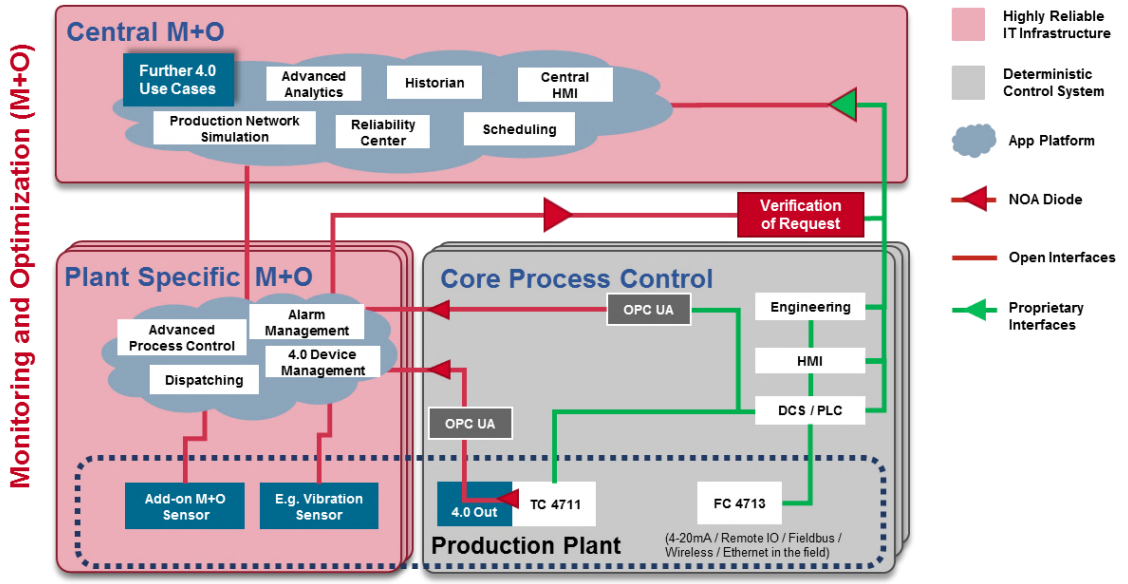


Figure 8: Concept of the NAMUR Open Architecture (NOA) as a supplement to the existing ISA-95 automation structure (NAMUR NE175 [NE175]).

2.3. Industrial Communication

A considerable heterogeneity characterizes the communication in the domain of industrial automation and IIoT [SHW99; VDI3687; WSJ17]. While on the field level, multiple, incompatible field buses and sensor/actuator networks for real-time communication evolved over the last decades and are still in operation [Neu07], the IT technology is mainly based on Ethernet networks [PN09; Sau07]. Still, heterogeneity on the IT level does not arise from the lower levels of the OSI model [ISO7498] as for the different field buses, but on the upper layers in the form of various communication protocols. Therefore, in the following Sections, an overview of communication on the field and superordinate levels is given.

Discrete-evet network simulators, such as OMNeT++ [Ope20; VH08] and Riverbed Modeler (former OPNET) [Riv19], allow the modeling of networked systems and their communication. Based on typical network components in libraries, the low-level network interaction can be modeled. Furthermore, the simulators allow the creation of individual models and the specification of behavior. The simulators can be used to investigate the performance of different protocols and topologies.

2.3.1. Field Level

On the field level, two types of wired communication networks can be differentiated: sensor/actuator networks and field buses. A typical example of a sensor/actuator network is the HART protocol. HART superimposes digital communication on existing analog 4…20 mA signals commonly found in the process industry [PN09]. Another example is IO-Link standardized in the IEC 61131-9 draft [IEC61131].

A large number of field buses are standardized in the IEC 61158 [IEC61158] and IEC 61784 [IEC61784] series. The field bus history is characterized by evolution over decades, as well as by incompatible physical layers and connectors. Some examples are Profibus DP or CAN. As these field buses are incompatible with the Ethernet-based superordinate levels of the automation pyramid, Ethernet-based field buses were developed for the field of industrial automation. Using the same physical layer simplifies the integration and convergence of OT and IT. However, as standard Ethernet is not capable of supporting real-time communication, significant modifications and adaptions on the protocol stack were necessary [Jas+09]. Consequently, several incompatible Industrial Ethernet networks evolved. Examples include Profinet, Modbus/TCP, or EtherCAT [Neu07; Sau07].

In recent years, the standardization efforts concerning a real-time capable standard Ethernet profile in the form of Time-Sensitive Networking (TSN) [IEC60802] and the introduction of wireless or

even cellular communication networks further diversified the landscape of industrial communication on the field level. Figure 9 reflects this diverse landscape of industrial communication on the field level in the form of market shares of field buses in the year 2019 [HMS19]. TSN and wireless 5G technology, however, provide the potential to unify industrial communication on a mid-term perspective and to simplify the integration with superordinate IT systems [Neu+18; Sau10; WSJ17].

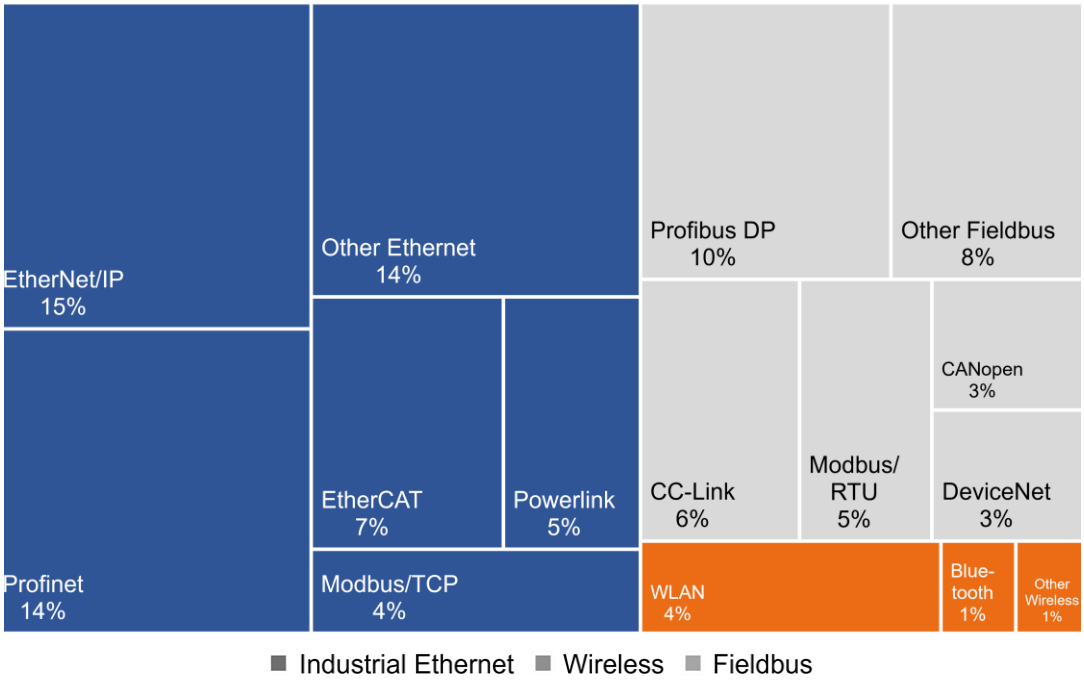


Figure 9: Industrial field bus and network market shares in 2019 (data from HMS Industrial Networks [HMS19]).

2.3.2. Superordinate Levels

On the superordinate IT levels, Ethernet is the predominant Physical Layer (cf. [ISO7498]), but a wide range of different network structures and protocols can be found. Furthermore, also, the IT levels are characterized by a large number of existing and mission-critical legacy systems [Bis+99]. Parts of this Section have been published as a German version in [Tru+20b].

With the need for better integration of systems (horizontal, vertical, and end-to-end) as a prerequisite for I 4.0, the number of interconnections between these systems is rising enormously. These connections are often engineered individually for each peer-to-peer (P2P) connection, causing a huge partially connected mesh network. As all P2P connections rely on a mutual understanding of the data in both systems, they are highly specific and cannot be reused for other P2P connections. Vogel-Heuser et al. [Vog+09] identified these challenges already before the introduction of the I 4.0 concept and proposed a common information model mediating between the systems as part of the automation diabolo (cf. Figure 10).

However, a common information model can only achieve a mutual, semantic understanding of data between all systems. The second aspect of heterogeneity, the multitude of legacy interfaces and protocols, is not solved by the introduction of an information model. Here, middleware concepts are the corresponding solution to unify protocols and communication.

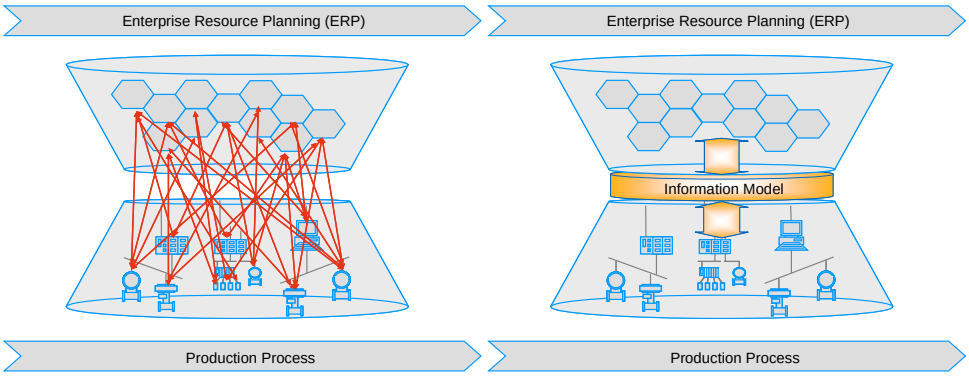


Figure 10: Information diabolo with individually engineered, direct P2P connections (left) and with a common information model (right) (Vogel-Heuser et al. [Vog+09]).

A middleware is a software solution that abstracts specific aspects of the underlying software and hardware systems with uniform interfaces. It is used as a link between heterogeneous systems and information representations [Izz09; VDI2657]. Middleware is a central aspect of IoT research, as can be seen in the surveys by Razzaque et al. [Raz+16] and Perera et al. [Per+14]. If all relevant systems are connected to the middleware (Figure 11 right), full connectivity as within a fully connected mesh network (Figure 11 left) can be achieved, but at a much lower number of necessary connections. A summary and comparison between the network types can be found in Figure 11.

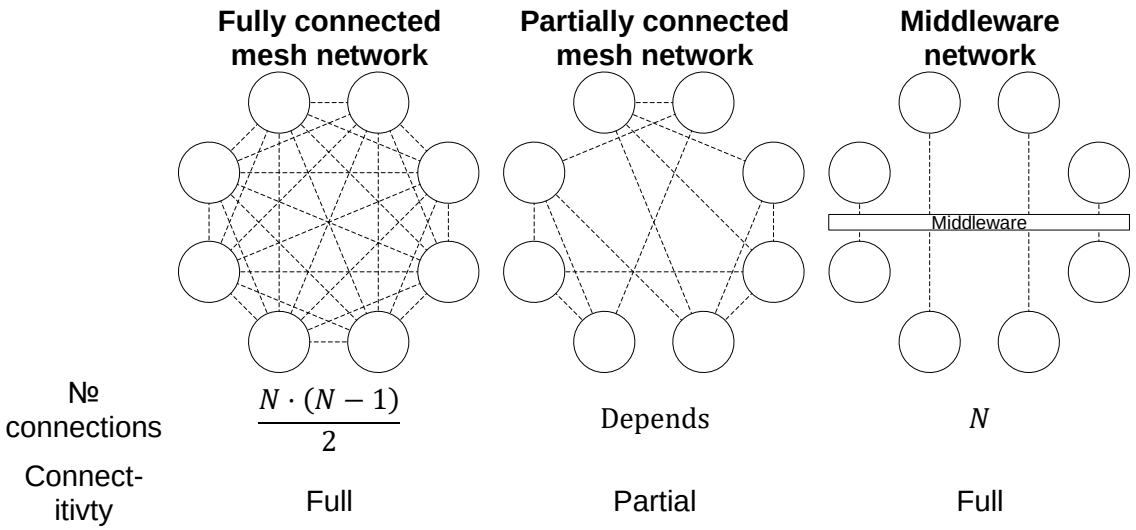


Figure 11: Comparison of different network structures with the number of connections depending on the number of systems N and connectivity [Haa97; Ind17c].

Uniform interfaces make the integration of systems and data much more manageable. Therefore, the specific interfaces, protocols, and information representations of legacy systems must be translated between the system-specific view and the common view at the middleware level. New systems can be created directly compatible with the middleware to reduce the additional integration effort. With middleware, two systems connected to the middleware no longer need to know the system-specific details of the other system.

However, a multitude of protocols and associated middleware is available for the realization of IIoT use-cases, all with their specific strengths and weaknesses [Al-+15; Str+18]. These include the underlying communication pattern, the used OSI layer 4 protocol (TCP or UDP), or support of quality of service (QoS) features such as message lifetimes or delivery guarantees [MKB07].

One group of protocols stems from the field of business integration. These protocols often follow the Enterprise Service Bus (ESB) concept presented by Chappel [Cha04]. ESB describes a central bus that acts as a message broker. All information is sent to this central broker and is then forwarded to the respective clients. The broker, therefore, completely decouples the communication between two connected systems. ESBs commonly provide additional functionalities for translation of information models, service orchestration, and message routing. Typical protocols encompass the Advanced Message Queuing Protocol (AMQP) [ISO19464] or the Representational State Transfer (REST) architecture style with the Hypertext Transfer Protocol (HTTP) [Fie00].

Additional protocols stem from the field of IoT. These protocols are characterized by their low computational complexity and relatively sparse support for QoS, which makes them feasible for distributed, low-cost IoT devices. Typical examples of this class are the Message Queuing Telemetry Transport protocol (MQTT) [ISO20922] or the Constrained Application Protocol (CoAP) [RFC7252].

Apache Kafka [Apa19] is a high-performance stream processing platform originating from the field of log file analysis. It is a common platform in scenarios where vast amounts of streamed data from a large number of data sources or systems have to be processed [Wan+15].

In the field of industrial communication, OPC Data Access (OPC DA) [OPC03] was the prevailing protocol. However, it is tied to the Windows platform as it is based on Microsoft COM technology. Its successor, the OPC Unified Architecture (OPC UA) [IEC62541], provides an operating system-independent communication platform with integrated capabilities for information modeling. However, as OPC UA is based in a server-client pattern, its scalability is limited. Therefore, with Part 14 of the OPC UA specification [OPC18], the publish/subscribe pattern was defined for OPC UA, and is supported in two ways: either decentralized using UDP broadcast messages

(UADP) or tunneling through AMQP or MQTT brokers. Other approaches exist with the MTConnect standard [MTC18] or with the Object Management Group's (OMG) Data Distribution Service (DDS) [OMG15]. Furthermore, OMG published the specification of an OPC UA/DDS gateway [OMG18] for the transparent interconnection of both protocols, and Pfrommer et al. [PGP16] propose a hybrid system.

A summary of relevant protocols can be found in Table 1, including their characteristics.

Table 1: Characteristics of relevant protocols (adapted from Trunzer et al. [Tru+19b; Tru+20b]).

Criterion	AMQP	CoAP	DDS	Kafka	MQTT	MT Connect	OPC UA			REST
							Standard	PS		
								UADP	Broker	
Messaging pattern	PS	RR	PS	PS	PS	PS & RR	RR	PS & RR		RR
OSI Layer 4 protocol	TCP	UDP	UDP	TCP	TCP	TCP	TCP	UDP	TCP	TCP
Architecture	C	DC	DC	D	C	DC	DC	DC	C	DC
QoS	+	+	++	-	+	-	-	+	-	-
Encryption	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Authentication	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Open-source implementation	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Number of suppliers	>10	>10	5-10	1	>10	>10	>10	1	0	/
Standard owner	ISO/IEC	IETF	OMG	Open	ISO/IEC	MTC	IEC	OPC Foundation		Open
Reference	[ISO19464]	[RFC7252]	[OMG15]	[Apa19]	[ISO20922]	[MTC18]	[IEC62541]	[OPC18]		[Fie00]

PS: publish-subscribe; RR: request-response; DC: decentralized; C: centralized; D: distributed

Overall, the multitude of different field buses and protocols with their specific characteristics complicates the task of data collection from CPSoS for big data applications in industrial automation. Furthermore, the heterogeneity and closeness of systems cause massive implementation efforts that make a wide-scale data collection often not feasible in industry.

2.4. Big Data in Automation

With the implementation of CPPS and IIoT in production, the rising number of connected systems, and the higher processing power in the field, more and more data (big data) become available [Che+18; KK19]. Big data is characterized by the so-called 4 V's [Has+15]:

- *volume* is the amount of data that is generated,
- *variety* corresponds to the heterogeneity of data that is collected,
- *velocity* refers to the speed of data generation, and

- *value* to hidden information and knowledge that is locked inside these vast amounts of heterogeneous, high-velocity data.

Other definitions with 3V's [Ber13] up to 7V's [KUG20147] exist, but all describe the idea that these characteristics render existing systems not capable of handling the data. Another description is given as the HACE theorem by Wu et al. [Wu+14] as "large-volume, heterogeneous, autonomous sources with distributed and decentralized controls" and the aim to "explore complex and evolving relationships among data."

To reveal and extract hidden information or knowledge from these massive amounts of data is the aim of data analytics and data mining [VH16; Wu+14]. Here, data-driven algorithms are often used to analyze the data. Significant interdisciplinarity and many involved disciplines characterize data mining projects in industrial automation [ITK19; YK15], as well as a large number of heterogeneous types of data that must be considered [RPC19].

Besides the main analysis of data, data staging was identified as one of the open research issues that have to be considered in the future [Has+15]. Data staging includes the collection and integration of data and is a crucial and challenging pre-step before being able to analyze the data.

2.5. Model-driven Development

Model-driven development (MDD) relates to a development paradigm that employs models not only for documentation purposes but as essential components during the development and engineering phase. Model transformations are used to leverage the modeled information to automate parts of the development process. Models and MDD play an essential role in industrial automation [Alv+18; BFS13; CFV20; Fay+15; Lie+18; Sch+02; Vog+14b; Vya13; WDF18]. In this Section, the fundamentals of modeling and MDD will be introduced. MDD can be used to decrease manual implementation effort and is, therefore, a candidate for a data collection architecture approach with manageable efforts.

Models are abstract representations of real-world objects. Modeling aims to capture and reflect relevant aspects of a system in an abstracted way. Models consist of model elements that describe distinct aspects of the real-world object. The higher the number of available model elements is, the more precise the real-world object can be described by the model. At the same time, models should be well-arranged and comprehensible. Balancing the level of detail that can be captured by models and their comprehensibility is, therefore, always a trade-off [LG99a; MJG11; Sta73].

Models have to conform to a metamodel that abstractly describes and defines the usable model elements. The OMG defines a metamodel as a "model that defines a modeling language and is also expressed using a modeling language" [OMG14]. As can be seen from the definition, recursion can occur when defining metamodels. Related to the given definition, a model that describes and defines the elements of the metamodel is called meta-metamodel. This recursion happens every time a model is defined. To solve the problem of a possible endless recursion, the OMG defined with its Meta Object Facility (MOF) [ISO19508; OMG16] a meta-metamodel that is capable of describing itself. Therefore, this model can serve as a root to define other metamodels and models. MOF introduced the concept of a so-called four-layered metamodel architecture, with MOF residing on the highest, called M3, layer. On the M2 layer, metamodels defined with MOF can be found. Therefore, models describing real-world systems are on the M1 layer. The lowest level, M0, corresponds to real-world systems and instances of the models.

The definition of a modeling language and its components is given in Figure 12 [BCW17; Rod15].

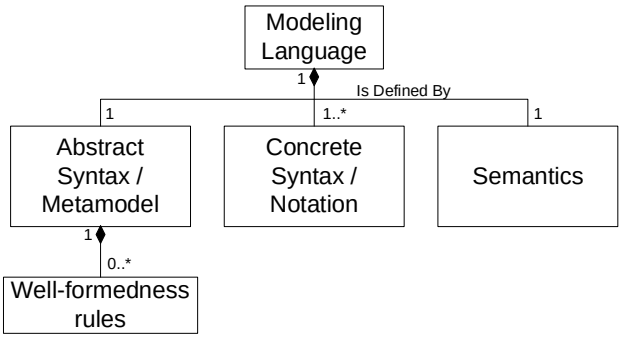


Figure 12: Definition of a modeling language according to Harel and Rumpe [HR00] (adapted and modified from Rodrigues [Rod15]).

The metamodel or abstract syntax of a modeling language describes the modeling elements, their names, and relations. It is an abstraction of the concepts of the modeling domain. The abstract syntax includes structural semantics that defines how model elements can be related under a set of constraints. These can either be formulated through specific languages, e.g., the Object Constraint Language (OCL) [OMG14], or informal, natural language. [HR00; Rod15]

The notation or concrete syntax refers to the usage of the modeling language by users. A modeling language can contain various notations, including textual or graphical ones [Rod15]. Notations should be designed to be understandable and, at the same time, provide enough expressiveness. Moody [Moo09] introduces with his "Physics of Notations" design principles for graphical (visual) notations. These include, for instance, the so-called perceptual discriminability, which describes how well symbols of the notation can be differentiated among each other, or semantic transparency, which suggests using symbols that directly reflect their meaning.

The semantics of a modeling language relates to the meaning of model elements and their relations. The semantics can be expressed as formalized models or in natural language describing their meaning [Rod15].

Modeling languages can be differentiated in General-Purpose Modeling Languages (GPMLs) and Domain-Specific Languages (DSLs). GPMLs can be applied to any domain for modeling, while DSLs are tailored for modeling specific applications or domains [BCW17]. A typical example of a GPML is the Unified Modeling Language (UML) [OMG17] on the MOF M2 level. UML defines a set of diagrams that can be used to specify distinct aspects of software systems. These include the structure of software systems (e.g., class diagram or component diagram) and their behavior (e.g., activity diagram or sequence diagram). An example that reflects aspects of a GPML, as well as characteristics of a DSL, is the Systems Modeling Language (SysML) [Bas+11; OMG19]. SysML is a UML-based modeling language for systems engineering applications. Therefore, it can be applied to a wide variety of different use-cases but provides more expressiveness and tailored model elements for specific aspects compared to UML. A further step towards a DSL is SysML4Mechatronics [Ker19] that extends SysML by additional features for modeling of non-software-related characteristics of mechatronic systems. An example of a DSL is the Palladio Component Model (PCM) [Reu+11; Reu+16] for the modeling of business software architectures.

MDD aims to leverage models that are modeled using a GPML or DSL for the software development process. A concrete proposal for the application of MDD is defined with the Model-Driven Architecture (MDA) [OMG14]. One central aspect of MDA is the usage of modeled information via model transformations. Model transformations generate one or more target models based on the information from one or more source models [MCV05]. Transformations can be used to transform models between different levels of abstraction or to a model based on another modeling language [OMG14]. Three general types of model transformations exist: model-to-model (M2M), model-to-text (M2T) and text-to-model (T2M) transformations. Model transformation can be implemented using general-purpose programming languages, for example, C# or Java, or based on specific model transformation languages [SK03].

A generic definition of a MOF-compatible M2M transformation language can be found with MOF Query View Transformation (QVT) [OMG16]. An example of a specific language for M2M transformations is the ATLAS Transformation Language (ATL) [ATL05]. Figure 13 gives an example of an M2M transformation using a QVT-compliant transformation language. The source (A) and target (B) metamodels both must be MOF-compliant. The M2M definition describes the mapping of model elements from metamodel A to model elements of metamodel B. Hence, it describes the transformation rules that can be executed by an M2M transformation engine. This engine reads an

instance of metamodel A (model A), executes the respective transformation rules, and outputs a model that is compliant to metamodel B (model B).

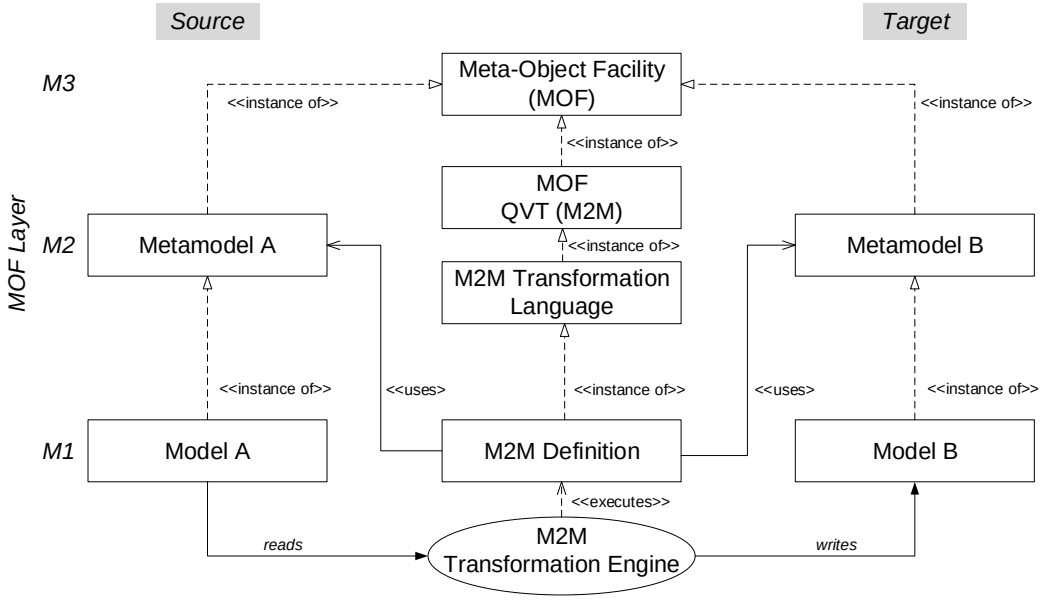


Figure 13: Principle of a model to model (M2M) transformation using a MOF QVT-compliant transformation language (following the conventions from Brambilla et al. [BCW17]).

With the specification of the MOF Model to Text Transformation Language (MOFM2T) [OMG08], a generic definition of an M2T language exists. A concrete realization can be found with the Acceleo transformation language [Ecl19g]. The working principle of an M2T transformation is depicted in Figure 14. Here, the target of the transformation does not conform to MOF. Hence, the M2T definition describes mappings between model elements from metamodel A and textual templates. These templates can be based on any textual representation, e.g., code written in any programming language, configuration files, or documentation.

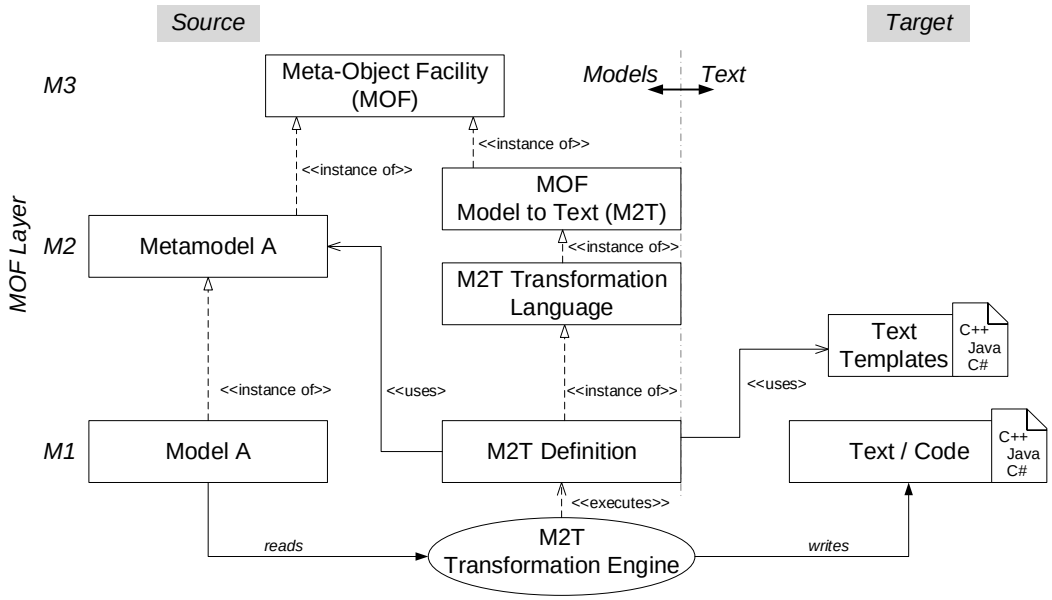


Figure 14: Principle of a model to text (M2T) transformation for text/code generation using a MOF M2T-compliant transformation language (adapted from Aicher [Aic18], as well as Schütz [Sch15] and extended, following the conventions from Brambilla et al. [BCW17]).

2.6. Virtualization and Containerization

Virtualization abstracts available hardware resources and can cover different aspects, e.g., the network or the computing resources. It offers the possibility to use available resources more efficiently and enables greater flexibility.

A common approach for virtualization is the virtualization of complete computers as so-called virtual machines, including storage, network, computing resources, and other in- and outputs. This is classically done using so-called hypervisors. Hypervisors allow multiple virtual machines to use the same hardware platform parallelly. Therefore, virtual machines contain complete installations of operating systems (OS) and run their own kernels. Access to the hardware is then managed through the hypervisor. A more recent approach is the so-called containerization or container-based virtualization. Instead of virtualizing the complete hardware and running multiple OS in parallel, containers are self-contained units that include only applications but share the same underlying operating system and hardware. Multiple containers then share the same host operating systems but are isolated from each other. Containers, therefore, do not contain complete OS installations and are more lightweight. Figure 15 summarizes the layers of both virtualization approaches and their differences visually. [Ber14; Pah15]

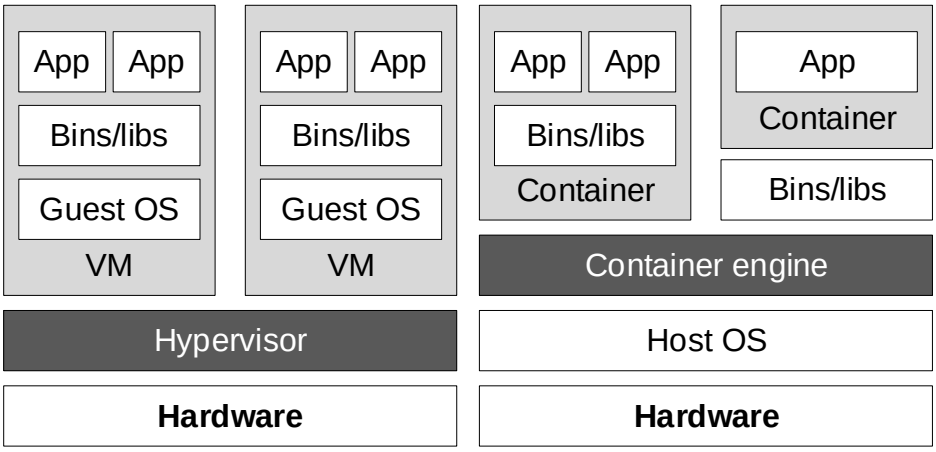


Figure 15: Comparison of virtualization architectures. Layers of hypervisor (left) and container (right) virtualization (adapted from Pahl [Pah15]).

A widely accepted container-software solution, and the current de-facto standard, is the open-source software Docker [Doc20d]. Docker provides a container engine for the execution of containers, as well as their management over a command-line interface. Additionally, Docker features the concept of repositories, where containers can be distributed and their dependencies managed.

While hypervisor-based virtualization suffers from measurable performance overhead due to the virtualization of resources and the scheduling of their concurrent usage, the overhead of container-based approaches is relatively small. Studies showed that their performance impact on computing, memory, and network performance is neglectable [Fel+15; Mor17; SLV19]. Recently, containerization has also found its way into real-time applications, which is still an active research field [SLV19; Str+20].

Multiple containers and their dependencies between each other can be managed by so-called orchestration engines. Orchestration engines allow the dynamic deployment of containers to so-called computing nodes (computers) and their monitoring. In the case of malfunctions or unexpected load peaks, the orchestration engines can react accordingly by restarting or redeploying distinct containers in the network. Multiple accepted orchestration engines for containers exist. A widely accepted, but complex solution, is Kubernetes [Clo20]. Among the alternatives is Docker swarm [Doc20b] that ships with Docker.

In summary, the concept of containers provides a lightweight and portable tool for sharing and deployment of software in distributed IT systems. Via orchestration engines, also large numbers of containers can be easily deployed, managed, and monitored.

3. Requirements on a Model-driven Approach for Data Collection System Architectures for Cyber-physical Systems of Systems

The concept of a model-driven approach for the automatic generation of data collection architectures must fulfill different requirements. The requirements are presented in the following. They can be derived from the state-of-the-art and/or industrial practice and represent the boundary conditions the concept must adhere to. Based on the field of investigation, these can be grouped into four categories, namely the requirements for the architecture concept itself (*Req-A*), the underlying software framework for industrial communication (*Req-SF*), the domain-specific language to model the data collection characteristics of CPSoS (*Req*-M) and the model-driven generation of the data collection architecture (*Req-G*).

3.1. Data Collection System Architecture (*Req-A*)

The approach should allow the collection of data from different levels of the existing automation pyramid [Dot+18]. Therefore, data from a multitude of distributed hard- and software systems must be collected and forwarded to other systems that analyze or store the data. These systems range from systems on the field level (e.g., smart sensors and PLCs) up to high-level IT applications (e.g., systems in a cloud environment) [Kle+17]. The approach, therefore, must bridge the OT and IT domains and allow data collection across all layers of the automation pyramid.

***Req-A*$_{ATP}$ – Data collection from different levels of the automation pyramid**
The approach should support data collection from different levels of the automation pyramid.

Both, the field level, as well as the superordinate IT systems, are characterized by a substantial heterogeneity when it comes to communication protocols and interfaces [BS15]. Numerous technologies and protocols exist with their related strengths and weaknesses that can be used to interconnect these distributed systems. The approach should take available technologies into account and include a technology-agnostic concept. This technology-agnostic concept must be realizable using distinct sets of technologies to account for the requirements of a specific application. As a result, the approach allows a simplified migration in the future if the constraints or the set of available technologies change over time.

***Req-A*$_{TAC}$ – Technology-agnostic concept**
Support different sets of technologies for concrete realizations of the architecture.

Systems in industrial automation are characterized by their long lifetime of often up to 40 years [BS15]. The organization of these systems often still strictly follows the layout of the hierarchical ISA-95 automation pyramid, which ensures timeliness and reliability of the systems control. Replacing this structure with a flexible communication mesh is a current trend in research [Del+17a; Vog+09], but costly and often infeasible for existing installations due to enormous implementation efforts. Therefore, the approach should also allow operation in parallel to the existing brownfield structures and the automation pyramid, not requiring a replacement of existing structures [ITK19; Kle+17]. This ensures broad applicability of the developed concepts.

***Req-A$_{POP}$* – Allow parallel operation to the automation pyramid**
The approach should be capable of being implemented as a coexisting extension of the automation pyramid. It should not require the replacement of existing structures.

An ever-increasing complexity and effort characterize the design and implementation phases of data collection system architectures for CPSoS. The reasons are the growing number of connected devices, the heterogeneity in protocols and technologies, as well as the multitude of involved experts from different fields [Str+18]. The approach should reduce complexity and the amount of manually programmed code to simplify initial implementations and configurations of the data collection system architecture.

***Req-A$_{Dep}$* – Simplified implementation and configuration (Deployment)**
A decreased effort for the implementation and configuration of the system architecture in terms of complexity and manually programmed code in comparison to manual programming.

Besides initial deployment, redeployments (migrations) are of significant interest for industrial applicability [Dot+18]. Currently, due to the excessive costs related to the migration from one technology to another, a vendor lock-in effect can be observed. Enterprises hesitate to migrate to newer, better-suited communication technology as the communication logic of all participants must be reimplemented. The approach should, therefore, support and simplify future migrations of architectures that are implemented under the developed approach.

***Req-A$_{ReDep}$* – Simplified migration between technologies (Redeployment)**
A decreased effort for the migration and reconfiguration of the system architecture from one communication technology to another in terms of complexity and manually programmed code.

3.2. Interoperability of Systems and Architecture Software Framework (*Req-SF*)

A practical realization of a data collection system architecture requires a software back end that allows the architecture concept to be implemented. The concept, therefore, must fulfill several additional requirements.

The considerable number of heterogeneous systems leads to massive efforts when interfacing these systems in order to collect data from them. Without a set of standardized interfaces that unify the communication and data collection from all involved systems, connections must be set up on a P2P basis. Additionally, installing a new participant requires the setup of multiple, independent interfaces depending on the number of needed connections and the available interfacing options of the communication peers. This practice increases the complexity of implementing and maintaining data collection architectures. The definition of standardized interfaces can decrease these integration efforts as one can rely on the interface definition and is not required to support a multitude of different interfaces and protocols [VDI2657].

***Req-SF*$_{API}$ – Standardized interfaces to minimize effort**
Definition of standardized interfaces for the integration of participants into the architecture.

A multitude of different technologies and protocols can be used for the realization of the system architecture. Often, the concrete technologies for implementation must be chosen in the early engineering phase. The implementation is then tailored to this specific set of technologies. This workflow drastically increases the costs for migration when a change of technology is needed, for instance, because new and better-suited technology is available or old technology not any longer available on the market. Encapsulating and abstracting the specifics of different technologies through the standardized interface (see *Req-SF*$_{API}$) can decrease the dependence on a specific set of technologies, prevent vendor lock-in, and simplify the migration to other technologies in the future [VDI2657].

***Req-SF*$_{ACP}$ – Abstraction of technology-specific properties of communication**
An abstraction of the specifics of distinct communication technologies and protocols behind the standardized interface.

The implementation of data collection system architectures in the automation domain always has to take legacy brownfield systems into account [Ind17c; Jha+14; Kle+17]. Currently, most of the relevant data is available from brownfield systems, and concepts for the interfacing of these

sources are needed. Therefore, when implementing the architecture in parallel to the automation pyramid, existing legacy systems must be interfaced. Concepts are needed which support the integration of these legacy systems for a co-existence with greenfield systems [AIM10; Ban+16; Che+18; KBD09].

***Req-SF$_{Leg}$* – Support of legacy systems**
Concepts for supporting data collection from legacy systems in brownfield environments.

3.3. Requirements on the Domain-specific Language for Architecture Modeling (*Req-M*)

Data collection architectures deal with various heterogeneous hardware devices and related software in the IT and OT domains that are connected through diverse types of networks and fieldbuses. Collecting and analyzing data from such distributed and networked systems of systems is challenging because of the considerable number of connected systems (up to several hundred) with often more than 1,000 in- and outputs per system. Moreover, the complexity of the underlying constraints (e.g., acceptable latency, transmission rates of networks, constrained computing power) needs to be considered. The design and operation of data collection architectures involve stakeholders from different domains. For instance, a data analyst may have requirements on the amount of historical data needed for training a new analysis model, while a control engineer is concerned about the maximum acceptable latency between data generation in a machine and a calculated decision in the cloud. A domain-specific language with a visual notation can support experts during early design phases to capture requirements as well as during the operation of the architecture. Besides enhancing the communication between the involved experts, such models can serve as a documentation of the running system [Pan+19; Pen+17; Str+09].

A domain-specific language must provide the means to model a multitude of devices, networks, and software functionalities from the IT and OT domains. This encompasses the field layer with sensors, actuators, field buses, and PLCs, as well as superordinate IT systems, Ethernet-based networks, and sophisticated software functionalities (e.g., data analysis, storage, visualization). This so-called system viewpoint of the modeling language should describe the available hard- and software, as well as the available network connections between and within the systems of systems.

***Req-M$_{Sys}$* – System viewpoint**
Viewpoint for modeling of hardware, software, and networks from the field level up to superordinate IT systems.

For systems with a substantial number of connected devices and analyses, the data flow becomes extraordinarily complex. Describing and capturing the flow of data is crucial for various stakeholders [Ran+18]. Additionally, relevant information about the type of data (e.g., integer or float), the state of data (batch or streamed data), as well as the type of system's interaction with the data (forwarding, modification) must be represented. For instance, without proper modeling of the data flow through the systems, it is almost impossible for an IT architect to correctly size hardware nodes. Additionally, a data analyst may not be able to determine influences on the quality of data.

***Req-M*$_{DF}$ – Data flow viewpoint**
Viewpoint for the modeling of flows of data through distributed hardware and software systems, including representation of additional information such as type and state of data.

To enable the DSL's usage during the engineering and operation of a system, the requirements for the system and its properties need to be represented. Experts from different domains (e.g., control engineers, automation specialists, IT architects, or data analysts) have distinct types of requirements, which the data collection architecture should fulfill. For instance, while a data analyst can define the required sampling rate f_S of a variable, an IT architect is concerned about the security of data transmission. Furthermore, to evaluate the performance of a system in operation, its actual properties must be compared to the defined requirements. Hence, the means of stating requirements and properties should be part of the modeling notation.

***Req-M*$_{PropReq}$ – Annotations for properties and requirements**
Additional elements for adding annotations to the system and data flow viewpoints that capture requirements and properties of the system.

Visual notations or modeling languages are widely applied in software engineering and can communicate complex information often more intuitively than textual representations [Moo09]. As experts from different domains, who lack a mutual understanding of each other's domain-specific terminology, participate in the process of developing a data collection architecture for CPSoS, a graphical modeling notation can improve the exchange of ideas between these disciplines and experts.

***Req-M*$_{Graph}$ – Graphical modeling notation**
The domain-specific language includes a graphical notation for describing CPSoS using the system and data flow viewpoints with additional annotations.

3.4. Requirements on the Model-driven Generation of Data Collection Architectures (*Req-G*)

One significant complexity when implementing a data collection architecture is the development of the software communication interfaces and their usage in every connected participant. A model-driven approach for the generation of a data collection architecture can decrease this manual engineering and programming efforts. Therefore, a model-driven approach should automate the generation of the communication interfaces per participant to configure the communication architecture automatically. An evaluation with adequate metrics is needed to prove the improved efficiency of the approach, as only a minor fraction of published approaches capture this aspect in the field of modeling languages in Industrie 4.0 [Wor+20].

***Req-G*$_{Com}$ – Model-driven generation of communication interfaces**
Automatic, model-driven generation of the communication interfaces for the participants.

3.5. Focus of the Thesis

The approach of this thesis covers the modeling of data collection from CPSoS and the model-driven generation of the underlying communication architecture itself. However, specific problems within this domain are out of the scope of this work. These problems are considered as prerequisites for this thesis and are summarized as follows:

firstly, the data collection architecture should function as a bidirectional but passive system. CPSoS should not conduct any control interactions over the communication channels of the data collection architecture. The data collection architecture, therefore, does not have the additional requirement of meeting hard real-time communication for the collection and communication of data. Instead, control interaction is carried out over existing, real-time-capable communication links, possibly using OT technology. Furthermore, this explains the positioning as an extension of the existing control structure in the form of the automation pyramid. Still, a parallel operation to modern CPS with an internal mesh structure is also possible.

Secondly, no quantitative evaluation of the visual quality of the graphical notation (e.g., according to the principles of Moody [Moo09]) is carried out. For this thesis, the DSL with the graphical modeling notation should function as a tool to summarize and structure the knowledge and expectations of the involved experts. Therefore, it has to provide the relevant viewpoints and means for expressing the specific problems of the automation domain. The graphical quality of the notation is, therefore, not in the scope of this thesis.

At last, no system behavior nor the mechanical hardware components of the system should be modeled. The approach is tailored to data collection in complex CPSoS but neglects the internal composition of these networked systems and their dynamic behavior. Participating CPS should be treated as black boxes that produce and consume data.

4. State-of-the-Art

The following Chapter surveys and evaluates state-of-the-art approaches that address a similar field of investigation concerning the requirements from Chapter 3. The survey is divided into three parts: The first part (Section 4.1) covers concepts and realizations of generic system architectures for interoperability in CPSoS as well as specialized data collection system architectures. The second part (Section 4.2) investigates textual and graphical modeling languages for system architectures. The last part (Section 4.3) considers approaches for the model-driven generation of communication system architectures in general and industrial automation. This Chapter closes with a derivation of an identified research gap (Section 4.4). Table 2 presents the rating scheme per requirement for the state-of-the-art comparison that will be used to assess the existing approaches.

Table 2: Summary of the rating scheme per requirement for the state-of-the-art comparison. Most criteria are graded with + (fully fulfilled), ○ (partially fulfilled), and - (not fulfilled), with additional criteria where applicable.

Data Collection System Architectures (*Req-A*)	
	***Req-A*$_{ATP}$ – Data collection from different levels of the automation pyramid**
	+ Consideration of data collection from different levels of the automation pyramid.
	○ Data collection without proper consideration of the automation pyramid levels.
	- No data collection.
	***Req-A*$_{TAC}$ – Technology-agnostic concept**
	+ Focus on a technology-agnostic concept for the architecture.
	○ No consideration of technology-neutrality, but the concept can be applied using different technologies.
	- Concept is not technology-agnostic.
	***Req-A*$_{POP}$ – Parallel operation to pyramid architecture**
	+ Concept is designed to be operated in parallel to the existing infrastructure.
	○ Concept can be implemented in parallel but lack of special considerations.
	- Concept replaces existing infrastructure.
	***Req-A*$_{Dep}$ – Simplified implementation and configuration (Deployment)**
	+ Manual implementation effort for the initial implementation and deployment of the system architecture is decreased.
	- Manual implementation effort for the initial implementation and deployment of the system architecture is increased.
	? Not evaluated.
	***Req-A*$_{ReDep}$ – Simplified migration between technologies (Redeployment)**
	+ Manual implementation effort for the migration and redeployment of an existingsystem architecture is decreased.
	- Manual implementation effort for the migration and redeployment of an existing system architecture is increased.
	? Not evaluated.

Software Framework (*Req-SF*)		***Req-SF*$_{API}$ – Standardized interfaces to minimize effort**
	+	Definition of standardized interfaces for communication.
	-	Peer-to-peer architecture without standardized interfaces.
		***Req-SF*$_{ACP}$ – Abstraction of technology-specific properties of communication**
	+	Abstraction of technology-specific properties of communication.
	-	No abstraction of technology-specific properties of communication.
	∕	Not relevant, e.g., no standardized interface defined.
		***Req-SF*$_{Leg}$ – Support for legacy systems**
	+	Integration of legacy systems is considered, and concepts are presented (brownfield deployment).
	○	Integration of legacy systems is only considered (conceptual brownfield deployment).
	-	No integration of legacy systems (greenfield application).
Architecture Modeling Language (*Req-M*)		***Req-M*$_{Sys}$ – System viewpoint**
	+	Full support for the system viewpoint.
	○	Partial support for the system viewpoint with relevant elements missing.
	-	No system viewpoint.
		***Req-M*$_{DF}$ – Data flow viewpoint**
	+	Flow of data through the system and additional information (type of data, state of data) is fully captured.
	○	Partial coverage of the data flow and its characteristics.
	-	No data flow viewpoint.
		***Req-M*$_{PropReq}$ – Annotations for properties and requirements**
	+	Modeling framework includes annotations for both viewpoints allowing the statement of requirements and properties.
	○	Annotations only include requirements or properties but lack consideration of both.
	-	No possibility of stating properties nor requirements.
		***Req-M*$_{Graph}$ – Graphical modeling notation**
	+	Graphical modeling notation for both viewpoints and annotations.
	○	Partial graphical modeling notation for either viewpoint.
	-	Lack of a graphical modeling notation.
Model-driven Gen. (*Req-G*)		***Req-G*$_{Com}$ – Model-driven generation of communication interfaces**
	+	Automatic generation of communication interfaces.
	○	Partial generation of the communication interfaces.
	∧	Concept for the generation of communication interfaces presented.
	-	No automatic generation of communication interfaces.

4.1. System Architectures

The following section summarizes and reviews state-of-the-art system architectures from the literature. The overview differentiates between generic architectures for interoperability and connectivity and dedicated data collection system architectures. Every section starts with a presentation of isolated approaches. Afterward, it is followed by a summary of research projects in the field. Concerning system architectures, only *Req-A*- and *Req-SF* categories are evaluated. Categories *Req-M* and *Req-G* do not apply as the approaches do not include any modeling notation or model-driven approach for their generation. The presented approaches all fulfill the *Req-*A_{API} requirement as this provides the basis for a configurable data collection architecture.

With the Apache PLC4X project [Apa20], a software framework for unified access to heterogeneous protocols is actively developed. The framework offers a shared programming interface that abstracts the specifics of the underlying protocols (*Req-*SF_{ACP}). Currently, several protocols, including low-level PLC protocols, such as Beckhoff ADS [Bec19c] or Siemens S7 ISO-on-ISO [RFC1006], are supported. The integration of existing legacy devices is the focus of the project (*Req-*SF_{Leg}). However, the project does not include an architecture concept.

4.1.1. Generic System Architectures

Kim and Youm [KY13] present a machine-to-machine platform for integrating data and services. Their approach targets generic IoT devices like smart sensors for consumer use. Therefore, aspects like data collection from various levels of the automation pyramid (*Req-*A_{ATP}) are not considered but could be fulfilled with adaptions. A standardized interface for communication is provided but does not abstract different communication protocols as only a single proprietary protocol is used for communication (*Req-*SF_{ACP}). Legacy devices are not considered (*Req-*SF_{Leg}). The concept is theoretically technology-neutral, but detailed considerations of this aspect are not in the scope of the approach (*Req-*A_{TAC}). A parallel operation is not foreseen (*Req-*A_{POP}).

Fiaschetti et al. [Fia+18] describe an implementation of a monitoring and control architecture for energy distribution systems. They combine a DDS for the field level and the AMQP message broker ActiveMQ for communication. As the application domain differs from industrial automation, a parallel operation is not foreseen, and direct control of systems is executed over the architecture (*Req-*A_{POP}). Moreover, the integration of existing legacy devices is not considered (*Req-*SF_{Leg}). The presented approach is tailored to the specific set of technologies used for its implementation and not applicable to other technologies (*Req-*A_{TAC}, *Req-*SF_{ACP}).

Longo et al. [LNP19] developed a concept with a central ESB and participants that communicate using REST web services. They aim to create a platform for digital twin applications. Multiple

distributed ESB instances communicate with each other over CoAP. The approach considers the complete range of systems in industrial automation (*Req-A_{ATP}*) and is implemented for a parallel operation to existing systems (*Req-A_{POP}*). Due to their focus on REST web services, the approach is not technology-neutral and does not abstract specific properties of communication (*Req-A_{TAC}*, *Req-SF_{ACP}*). The integration of legacy systems is mentioned, but the actual interfacing of these systems is not conducted and demonstrated (*Req-SF_{Leg}*).

A hybrid peer-to-peer/middleware architecture for retrofitting existing automation systems is described by Ismail and Kastner [IK16; IK17; Ism18]. The approach is used for vertical integration of data (*Req-A_{ATP}*) and is based on a combination of OPC UA and approaches for service discovery and orchestration. Gateways are proposed for interfacing legacy systems (*Req-SF_{Leg}*); however, the technology-specific properties are not abstracted as only OPC UA is considered as the transport protocol (*Req-SF_{ACP}*). Their concept allows a parallel operation to the automation pyramid (*Req-A_{POP}*) and can be interpreted as theoretically technology agnostic (*Req-A_{TAC}*). However, special consideration and demonstration of this aspect are not included.

The architecture by Sola et al. [SGL15] (projects ComVantage and FITMAN) aims at enhancing the interoperability between field level and superordinate systems by data collection and exchange (*Req-A_{ATP}*). The architecture concept is formulated with technology-neutrality in mind (*Req-A_{TAC}*); however, it is not clear if a parallel operation to the automation pyramid is allowed (*Req-A_{POP}*). Standard interfaces abstract the specifics of communication protocols (*Req-SF_{ACP}*). Additionally, the integration of legacy systems is considered (*Req-SF_{Leg}*).

The Line Information System Architecture (LISA) by Theorin et al. [The+16] proposes an ESB for flexible data integration and control in factories. Data adapters allow the integration of existing legacy systems (*Req-SF_{Leg}*), ranging from the field level up to ERP systems (*Req-A_{ATP}*). A parallel operation to the existing control structure is demonstrated in an industrial application of the architecture (*Req-A_{POP}*). While the concept itself is technology-neutral with no particular focus on this aspect (*Req-A_{TAC}*), the communication and programming interfaces are not, as only AMQP over ActiveMQ is supported (*Req-SF_{ACP}*).

The MAYA project [Cia+17] conceptualizes a microservice architecture for digital twins in production. Participants communicate with each other using web services. The concept includes bidirectional communication between systems on the field level and a simulation framework. Superordinate systems are not mentioned, but the architecture could also be applied for data collection from these (*Req-A_{ATP}*). While the concept itself is not based on a specific technology, special considerations of this aspect are not part of MAYA (*Req-A_{TAC}*). Due to the nature of a distributed

control logic and bidirectional interaction for control of connected systems, a parallel operation to the existing automation pyramid is not possible (*Req-A_{POP}*). The standardized interface does not abstract communication with different communication technologies as it is focused on web services (*Req-SF_{ACP}*). Also, the integration of legacy systems is not part of the MAYA approach (*Req-SF_{Leg}*).

The Manufacturing Service Bus (MSB) [Grö+16; Kas+17; Mín12] is an ESB-realization for data integration over the product lifecycle. It aggregates data from engineering as well as from different automation levels during the operation of production plants (*Req-A_{ATP}*). Legacy applications can be retrofitted to support the standardized interface using adapters (*Req-SF_{Leg}*). The programming interface abstracts specific properties of communication, and the concept itself is technology agnostic (*Req-SF_{ACP}*, *Req-A_{TAC}*). A practical realization is demonstrated by Schel et al. [Sch+18], where REST web services, OPC UA, and MQTT are used for communication with participants. Parallel operation to existing infrastructure is not considered but could be implemented (*Req-A_{POP}*). An approach that builds on top of the MSB is Virtual Fort Knox [Hol+13]. Virtual Fort Knox encompasses a cloud platform, where an MSB and additional applications are hosted in a cloud environment, to enable small and medium-sized enterprises (SMEs) to benefit from the MSB concept.

BaSys 4.0 [EGW18; Kuh+18; Tru+19c] is a German project that is dedicated to the vision of a generic Industrie 4.0 middleware that follows the RAMI 4.0 principles, including administration shells. The conceptualized middleware is called Virtual Automation Bus and offers two distinct communication channels: one for real-time communication and one for non-real-time communication. Different technologies for communication are considered and can be used for the implementation (*Req-A_{TAC}*). Their specifics are abstracted with a standardized interface (*Req-SF_{ACP}*). Legacy systems can be integrated using special adapters that translate between Virtual Automation Bus and the legacy systems (*Req-SF_{Leg}*). BaSys 4.0 replaces the existing control structure of plants with a flexible, new architecture (*Req-A_{POP}*) and cannot be operated in parallel. The implementation results of BaSys 4.0 are collected in the Eclipse BaSyx project [Ecl19d] that provides a framework for future applications of the platform.

Lastra et al. [FL17a; FL17b; Moc+12] focus on web services-based architectures for data integration in industrial automation systems. Their approaches include concepts for retrofitting and interfacing existing legacy systems (*Req-SF_{Leg}*). However, their aim is always to replace the existing hierarchical structure of automation systems (*Req-A_{POP}*). Additionally, as only web services are considered as means of communication, their concepts are not technology agnostic, nor are their standard interfaces abstracting the specifics of different protocols (*Req-A_{TAC}*, *Req-SF_{ACP}*).

Lastra et al. [Fer+17; Ift+18; Qur+17] are also involved in the C2NET project, which implements a real-time data collection architecture to optimize the supply networks of SMEs. An architecture for the collection of data from an ERP system and related data sources (relational databases, spreadsheets) is implemented by Qureshi et al. [Qur+17]. The approach employs an ActiveMQ broker, which is operated in coexistence to the control infrastructure (*Req-A_{POP}*). In parallel, cloud-based architectures that replace existing infrastructure with a web service-based approach are presented. These concepts also consider data collection from field devices (*Req-A_{ATP}*) [Fer+17; Ift+18]. Gateways allow the integration of existing legacy systems into the architecture (*Req-SF_{Leg}*). The presented approach comprises the idea of a standard and abstracted interface that encapsulates the technology-specific properties of communication protocols (*Req-SF_{ACP}*). However, the architecture concept is not fully technology-agnostic due to the focus on web services for communication (*Req-A_{TAC}*).

In the scope of his dissertation, Leitão presented ADACOR (ADAptive holonic COntrol aRchitecture) [Lei04; LR06]. ADACOR is a multi-agent system for agile and adaptive control of manufacturing. Every system is represented by an agent that communicates with the other agents in a distributed architecture. These agents encapsulate the legacy interfaces of systems (*Req-SF_{Leg}*) and replace the existing control infrastructure (*Req-A_{POP}*). Leitão also considers superordinate systems, but the focus of ADACOR is on the field level (*Req-A_{ATP}*). The concept itself is technology agnostic (*Req-A_{TAC}*). Leitão et al. [LCR05] also present an implementation of the architecture using JADE, a software framework for agent development in Java. A refined version, called ADACOR2 [Bar+15], incorporates the idea of agent evolution to support self-adaptivity over time.

The IMC-AESOP project [Del+11; Kar+14; LCK16] realized a cloud-based and service-oriented architecture for plant control applications that aims at replacing existing control systems (*Req-A_{ATP}*, *Req-A_{POP}*). Local clouds enable the communication of systems over standardized web services. Additionally, migration approaches to retrofit and integrate legacy systems are discussed (*Req-SF_{Leg}*). While the concept itself is technology agnostic (*Req-A_{TAC}*), the API is tailored to web services and does not abstract between different technologies (*Req-SF_{ACP}*).

The SOCRADES project [KBD09; LCK16] developed an architecture for a so-called next-generation industrial automation architecture meant to replace the existing automation pyramid (*Req-A_{ATP}*, *Req-A_{POP}*). Webservices are used for communication of the systems, including mechanisms for service discovery and orchestration. Gateways and mediators interface legacy systems and enable their incorporation into the new automation architecture (*Req-SF_{Leg}*). Due to the strong focus on web services and related technologies for service discovery, neither the concept nor the implementation are technology-neutral (*Req-SF_{ACP}*, *Req-A_{TAC}*).

The Arrowhead project [Car17; Del+17a; Del+17b; Der+15; Var+17] provides a framework for distributed, cloud-based interaction of systems. Arrowhead is built on top of IMC-AESOP and SOCRADES. Additionally, it enables realtime capable communication if necessary, but replaces the existing control architecture with the new paradigm of cloud-based, federated CPSoS (*Req-A_{POP}*). Protocol translators allow the integration of legacy devices into the cloud environment [DED17] (*Req-SF_{Leg}*). Furthermore, the Arrowhead framework allows the usage of different protocols for communication and abstracts their specific properties (*Req-SF_{ACP}*).

The ARUM project [LCK16; Lei+13; Lei+15] proposes an agent-based architecture with an ESB acting as middleware between the different systems. The developed service-oriented architecture aims at minimizing the response time to unexpected events during the ramp-up phase of plants. Legacy devices are incorporated using gateways (*Req-SF_{Leg}*), but the existing control infrastructure is replaced by ARUM (*Req-A_{ATP}*, *Req-A_{POP}*). Additionally, ARUM describes an ecosystem consisting of architecture and many advanced tools for simulation, scheduling, and planning. Agent communication follows the FIPA specifications; see, for instance, the FIPA specification for HTTP as transport medium [FIPA02], which limits the technology-neutrality of the concept and its implementation (*Req-A_{TAC}*, *Req-SF_{ACP}*).

An architecture for flexible reconfiguration of CPS is developed within the PERFoRM project [Gos+17; Lei+16]. PERFoRM includes a middleware component for communication across multiple layers of the automation hierarchy (*Req-A_{ATP}*) and can be operated in parallel to the existing control infrastructure (*Req-A_{POP}*) [PER16a]. The PERFoRM middleware supports various middlewares (for instance, Apache Camel [Gos+18] and Apache Service Mix [Cha+17; Gos+17]). Furthermore, it abstracts the specifics of protocols and systems with a standardized interface (*Req-A_{TAC}*, *Req-SF_{ACP}*) and a supplementary information model called PML [PER17]. Detailed concepts are derived for the integration of legacy systems into the architecture and the application of the PERFoRM concept to different use-cases (*Req-SF_{Leg}*) [Lei+17; PER17].

4.1.2. Data Collection System Architectures

A dedicated data collection architecture is presented by Gama et al. [GTD12]. The architecture is conceptualized to collect data from distributed RFID readers. Due to the different application domain, the architecture does not consider different hierarchical levels in industrial automation, but could also be applied in this domain (*Req-A_{ATP}*). Webservices are employed for the communication of the distributed systems and a central, mediating ESB-component. Hence, the architecture is neither technology agnostic (*Req-A_{TAC}*), nor does the interface abstract the specific communication logic of web services (*Req-SF_{ACP}*). Additionally, the integration of existing legacy systems is not considered (*Req-SF_{Leg}*).

Liu and Jiang [LJ16] present an architecture for data collection from various levels of the automation pyramid (*Req-A_{ATP}*). It is based on big data components, such as Apache Hadoop, a software framework for the processing of big data. Technology-neutrality of the concept is not discussed, but could theoretically be achieved (*Req-A_{TAC}*). Nevertheless, the standard interface does not abstract the specific properties of different technologies (*Req-SF_{ACP}*). Due to the nature of a data collection architecture, a parallel operation to the existing infrastructure is possible (*Req-A_{POP}*). However, it remains unclear how to interface legacy systems (*Req-SF_{Leg}*).

Kirmse et al. [Kir+18] propose an architecture for data collection and integration from CPSoS. They also consider data that is distributed over multiple companies and data collection from multiple levels of the automation hierarchy (*Req-A_{ATP}*). The implementation of their architecture uses OPC UA for communication, while their concept could be applied using different technologies (*Req-A_{TAC}*). As OPC UA is used as the standard communication protocol, the defined interface does not abstract the specifics of communication but is based on the functionalities of OPC UA directly (*Req-SF_{ACP}*). Legacy system integration is considered but not demonstrated (*Req-SF_{Leg}*).

Liu et al. [Liu+16] use a commercial OSIsoft PI [OSI19] system for data collection and integration in the domain of power systems. They use a model-driven approach to automate the generation of information models based on the Common Information Model defined in EN 61968 [Eur13]. A parallel operation is not discussed but typical for OSIsoft PI systems (*Req-A_{POP}*). Data is collected from the field levels as well as higher-level systems (*Req-A_{ATP}*). Additionally, the concept considers the integration of legacy systems over data adapters (*Req-SF_{Leg}*). Still, due to the limitation on OSIsoft PI, neither the concept nor its implementation is technology agnostic (*Req-A_{TAC}*, *Req-SF_{ACP}*).

The big data cloud platform AMCoT [Lin+17; Liu+18] is developed for the domain of semiconductor manufacturing. Individual systems are interfaced using so-called cyber-physical agents, which can also integrate legacy systems into the architecture (*Req-SF_{Leg}*). Communication is conducted using REST or SOAP web services. MATLAB is used for executing analysis in a connected cloud environment. Parallel operation to the existing infrastructure is not explicitly mentioned but possible (*Req-A_{POP}*). Neither the concept nor its specific implementation of communication is technology-neutral due to restriction of only supporting web services for communication (*Req-A_{TAC}*, *Req-SF_{ACP}*).

Fleischmann et al. [FKF16a; FKF16b; Fle+16] present an architecture for aggregating data from different levels of the automation pyramid for condition monitoring applications (*Req-A_{ATP}*). Their

concept is realized using web services, but could theoretically be applied using other sets of technologies (*Req-A_{TAC}*). However, technology-specific aspects of communication are not abstracted in the standardized API (*Req-SF_{ACP}*). Albeit the integration of legacy systems is mentioned, it is not demonstrated (*Req-SF_{Leg}*).

Peres et al. [Per+18] propose the IDARTS framework, a hybrid multi-agent/Apache Kafka-based architecture for data collection and analysis in industrial automation. Component Monitoring Agents collect the data from field level systems but could theoretically also be used to acquire the data of higher-level systems (*Req-A_{ATP}*). The collected data is forwarded to a central Apache Kafka instance, which mediates between the field level agents and the analysis part of the architecture. The concept is developed without a specific technology in mind, however special considerations on technology-neutrality are missing (*Req-A_{TAC}*). Parallel operation to the existing control infrastructure is foreseen (*Req-A_{POP}*). The definition of so-called Generic Data Collection Interfaces and Generic Data Output Interfaces unify the in- and outputs of the system. Nevertheless, the implementation of IDARTS is tailored for the JADE agent framework and Apache Kafka. Therefore, it does not abstract further (*Req-SF_{ACP}*). The integration of legacy systems is mentioned, but specific concepts are lacking (*Req-SF_{Leg}*).

In the scope of the COCOP project [COC18a; COC18b; HKV18], an architecture for plant-wide monitoring applications is developed. Data is collected from different hierarchical levels of the automation systems (*Req-A_{ATP}*), including legacy systems that are interfaced using adapters (*Req-SF_{Leg}*). Different broker technologies are compared, and an actual implementation using RabbitMQ is presented. Therefore, the concept can be seen technology-neutral, but this is not in the scope of the COCOP project (*Req-A_{TAC}*). A parallel operation is foreseen for the monitoring of existing plants (*Req-A_{POP}*). Additionally, the standardized interfaces take communication over AMQP, REST, and OPC UA into account. Nevertheless, it remains questionable if the actual implementation of the COCOP architecture encompasses this feature of a protocol-agnostic interface (*Req-SF_{ACP}*).

Table 3 summarizes the requirement fulfillment of all presented generic and data collection system architectures. As can be seen from Table 3, none of the approaches evaluated if decreased implementation or redeployment efforts can be observed with the architecture. Only the PERFoRM architecture fulfills all other requirements of categories *Req-A* and *Req-SF*. The PLC4X plays a special role as it does not include an architecture concept but is limited to a software framework.

Table 3: Evaluation of relevant approaches in the field of system architectures and data collection system architectures. See Table 2 for the rating scheme.

Approach	Requirements							
	Req-A_{ATP}	***Req-A_{TAC}***	***Req-A_{POP}***	***Req-A_{Dep}***	***Req-A_{ReDep}***	***Req-SF_{API}***	***Req-SF_{ACP}***	***Req-SF_{Leg}***
ADACOR	○	+	-	?	?	+	-	+
AMCoT	+	-	○	?	?	+	-	+
Arrowhead	+	+	-	?	?	+	+	+
ARUM	+	○	-	?	?	+	/	+
BaSys 4.0	+	+	-	?	?	+	+	+
C2NET	+	○	+	?	?	+	+	+
COCOP	+	○	+	?	?	+	+	+
Fiaschetti et al.	○	-	-	?	?	+	-	-
Fleischmann et al.	+	○	+	?	?	+	-	○
Gama et al.	○	-	○	?	?	+	/	-
IMC-AESOP	+	○	-	?	?	+	-	+
Ismail and Kastner	+	○	+	?	?	+	-	+
Kim and Youm	○	○	-	?	?	+	-	-
Kirmse et al.	+	○	+	?	?	+	/	○
Lastra et al.	+	-	-	?	?	+	-	+
Liu and Jiang	+	○	+	?	?	+	-	-
Liu et al.	+	-	○	?	?	+	/	+
Longo et al.	+	-	+	?	?	+	-	○
MAYA	○	○	-	?	?	+	-	-
MSB	+	+	○	?	?	+	+	+
Peres et al.	○	○	+	?	?	+	-	○
PERFoRM	+	+	+	?	?	+	+	+
PLC4X						+	+	+
SOCRADES	+	-	-	?	?	+	/	+
Sola et al.	+	+	○	?	?	+	+	+
Theorin et al.	+	○	+	?	?	+	-	+

4.2. Modeling Languages

In the following Section, relevant modeling languages are summarized and reviewed. At first, AutomationML, as a universal data exchange format is considered. Afterward, UML profiles are presented, followed by a review of graphical modeling notations. As none of the approaches includes an architecture concept nor the model-driven generation of system architectures, only requirements of category *Req-M* are considered in the following review.

The Automation Markup Language (AutomationML or AML) [Dra+08; IEC62714] is a data exchange format for the domain of industrial automation. AutomationML aims to provide a vendor-neutral, XML-based data format that can be used to exchange data between heterogeneous engineering systems and tools. AutomationML combines existing standards and accepted exchange formats to describe topology, geometry, and kinematics, as well as control software of automation systems. Furthermore, AutomationML provides the possibility of enhancing the modeling capabilities with so-called role class libraries. A role class library for communication is described in several publications [Aut14; DLH13; RD18; Rie+14a]. The basic AutomationML libraries, in conjunction with the communication library, can be used to model communication networks in control systems. They include elements to describe the hardware and networks of the system, as well as simple software functionalities related to control of the system. The library separates between a logical data processing view and a physical hardware view. The views are mapped to reflect which software functionality is executed on which hardware. Besides, a simple mechanism for the description of data exchange exists. However, it is mostly limited to the field level. Therefore, *Req-*M_{Sys} and *Req-*M_{DF} are both partially fulfilled, as only a description of complex software functionalities and a more complete data flow viewpoint are not considered. Nevertheless, a possibility to annotate the models with properties and requirements (*Req-*$M_{PropReq}$) and a graphical representation of the model (*Req-*M_{Graph}) are not part of AutomationML.

4.2.1. UML-profiles

The OMG specified MARTE [OMG11], a UML profile for the Modeling and Analysis of Real-Time Embedded Systems. MARTE differentiates between a design model for the design of systems and an analysis model for the analysis of schedulability and performance of designed systems. As MARTE is a UML 2 profile, its graphical modeling capabilities are limited and restricted to additional icons and simple symbols in combination with the graphical elements of UML (*Req-*M_{Graph}). The system can be described in terms of available hardware and resources, I/O signals, and communication interfaces, including networks. Therefore, *Req-*M_{Sys} can be considered as entirely fulfilled. MARTE does not include an explicit data flow viewpoint. Instead, the flow of data is directly modeled within the hardware elements. Therefore, following the flow of data

and its manipulations through the systems is only implicitly possible (*Req-M_{DF}*). An annotation of modeling elements with additional properties and requirements is possible. The specification of MARTE foresees many relevant items for timing and scheduling, but additional items, e.g., for the specification of protocols or encryption requirements, are missing (*Req-$M_{PropReq}$*).

UML-RT [Gro+99; Sel98] is a UML profile for event-driven, distributed real-time software based on the ROOM (Real-Time Object-Oriented Modeling) language [SGW94]. UML-RT relies on the graphical elements of UML but does not provide a graphical notation, including any symbols itself (*Req-M_{Graph}*). The modeling language features elements for the definition of so-called capsules, which can communicate with other capsules over ports. These ports implement a specified protocol behavior and can be connected with connectors. UML-RT does not include elements for the explicit modeling of hardware systems (*Req-M_{Sys}*). The flow of data is modeled implicitly with the description capsules and ports/connectors and provides only basic information (*Req-M_{DF}*). For instance, the type of data handling cannot be seen directly, but the internal behavior of a capsule needs to be investigated. UML-RT provides no way of stating any properties or requirements (*Req-$M_{PropReq}$*).

Katzke and Vogel-Heuser defined the so-called UML-PA (process automation) profile [Kat08; KV05a; KV05b]. UML-PA is a tailored profile for the modeling of software in industrial automation systems. UML-PA introduces additional graphical symbols but relies mainly on the graphical notation provided by UML (*Req-M_{Graph}*). The modeling language differentiates between hardware and software of a system. Signals that are connected to a hardware unit are mapped to software signals. Additionally, networks can be defined. *Req-M_{Sys}* is, therefore, entirely fulfilled. On the other hand, data flows can only be modeled implicitly without special consideration of types of data flow and data manipulations (*Req-M_{DF}*). UML-PA provides a basic set of properties and requirements that can be used to annotate models. These are mainly related to bus capacities and time constraints but fail to capture additional characteristics (*Req-$M_{PropReq}$*).

The Service-oriented architecture Modeling Language (SoaML), which is specified by the OMG [OMG12], contains a metamodel and a UML profile for modeling and design of service-oriented architectures. Therefore, SoaML includes stereotypes for the interaction of services. For instance, data flows can be captured in UML Sequence Diagrams and describe the interactions and roles of services. A possibility to describe the type of flow (batch/stream) is missing (*Req-M_{DF}*). Moreover, SoaML includes basic stereotypes for the description of data types and signals. Besides, the software functions of the systems can be modeled as services. As SoaML is designed to describe service-oriented architectures, elements for the hardware and network de-

scription are not included (*Req-*M_{Sys}). SoaML does not include elements that characterize the properties and requirements of system architectures (*Req-*$M_{PropReq}$). Finally, as SoaML is designed as a UML profile, a separate graphical modeling notation is not part of the specification (*Req-*M_{Graph}).

Another UML profile exists with the OMG's SysML [ISO19514; OMG19], a modeling language for system engineering. Compared to UML, SysML introduces additional diagrams and extends the available UML model elements. The structure of the system can only be modeled using abstract *block*-elements. More detailed symbols and differentiation of components of industrial automation systems are not available (*Req-*M_{Sys}). Communication between systems can only be modeled implicitly using ports without additional information concerning the type of data handling (*Req-*M_{DF}). SysML foresees requirements specifications but lacks separate mechanisms for property specification in relation to the requirements (*Req-*$M_{PropReq}$). The graphical model elements are limited to the elements available in UML (*Req-*M_{Graph}).

With the SysML-vAT (SysML for distributed automation systems) [Fra14], Frank adapted and extended the SysML for the modeling of distributed automation systems. Therefore, new stereotypes to further specify hardware components of the systems are introduced. However, detailed modeling of networks and variables is missing (*Req-*M_{Sys}). SysML-vAT extends the modeling of ports with directed ports for in- and outputs, but as with SysML, data flows can only be captured implicitly (*Req-*M_{DF}). With the introduction of colors and additional symbols, the graphical capabilities of SysML are extended slightly but still very limited (*Req-*M_{Graph}). Vogel-Heuser et al. [Vog+14a] demonstrate the model-driven generation of IEC 61131-3-compliant code from SysML-vAT models but do not cover non-automation software layers.

4.2.2. Graphical Notations

The AADL (Architecture Analysis and Design Language) [Fei+05; FG13; FLV06] is a modeling language for real-time applications and embedded systems standardized by SAE International (Society of Automotive Engineers) in SAE standard AS5506C [SAEAS5506C]. AADL puts significant effort into an exhaustive and formal specification of embedded systems in performance-critical applications. The language features constructs for a detailed description of software and hardware systems, as well as in- and output signals (*Req-*M_{Sys}). The data flow through the system is modeled together with the system aspects and is included implicitly (*Req-*M_{DF}). A basic set of properties can be stated, but additional aspects, such as protocols, encryption, and semantics are not considered due to the intense focus on embedded systems. Also, mechanisms for stating requirements are not taken into account (*Req-*$M_{PropReq}$). AADL provides a textual and graphical syntax for system modeling (*Req-*M_{Graph}).

The Open Group, an industry consortium to develop and foster open, vendor-neutral standards, specifies the ArchiMate modeling language [LPJ10; Ope19]. The scope of ArchiMate is the modeling of enterprise architectures and their evolution over time, but without a particular focus on the automation domain or Cyber-physical Systems. ArchiMate features a simple, but powerful graphical modeling notation with support for various symbols and different icons (*Req-M*$_{Graph}$). Besides, ArchiMate supports the definition of individual viewpoints. Considering the description of the system, the notation allows the modeling of host systems, software applications, and networks. However, the aspect of signals and master/slave networks is not considered (*Req-M*$_{Sys}$). The flow of data (*Req-M*$_{DF}$) can only be described implicitly, without a separate viewpoint, without the possibility to model distinct types of data and data manipulations. ArchiMate provides a set of basic annotations that can be used for the definition of requirements and properties. Nevertheless, only high-level annotations are defined in ArchiMate, which could be extended and customized if necessary (*Req-M*$_{PropReq}$).

Greifenender and Frey [GF07; Gre07] developed a graphical notation called DesLaNAS for the description of networked systems (*Req-M*$_{Graph}$). The focus of the approach lies in the modeling of communication-based delay between connected systems. Therefore, the flow of information and its delay through systems and networks can be modeled, but a possibility to describe the type of flow (stream/batch) is missing (*Req-M*$_{DF}$). Besides, the notation does not differentiate how a system influences the data. A system viewpoint, as well as the possibility to annotate models with properties and requirements, are not considered (*Req-M*$_{Sys}$, *Req-M*$_{PropReq}$).

Lewin et al. [LVF17] developed an adapted value stream analysis for information flows in Industrie 4.0 scenarios. Their approach does not capture the structure of the system (*Req-M*$_{Sys}$) but considers the flow of data and information between connected systems. The value stream analysis is focused on superordinate systems. Furthermore, it includes the direction of flows and a basic description of actions that are conducted with the data, e.g., data processing or data analysis. Still, a differentiation of data flows (stream/batch), mapping on the system viewpoint, and the description of the transmitted variables and values are missing (*Req-M*$_{DF}$). A possibility to annotate the models with properties and requirements is not considered (*Req-M*$_{PropReq}$). The concept encompasses a graphical notation for the description of data and information flows but no symbols for the system viewpoint (*Req-M*$_{Graph}$).

The group of Vogel-Heuser et al. developed a graphical modeling notation for decentralized control systems (DCS). The first version of this notation, which is inspired by UML-PA, was presented by Witsch and Vogel-Heuser [WV08] and provides graphical elements for modeling of control hardware, related networks, and in-/outputs (*Req-M*$_{Graph}$, *Req-M*$_{Sys}$). Additionally, models

can be annotated with relevant properties and requirements (*Req-*$M_{PropReq}$). The approach by Witsch and Vogel-Heuser includes an underlying metamodel to structure the modeled information. An extended version of the notation was later presented by Vogel-Heuser et al. [Had+12; Has+13; Vog+11; Vog+12] and encompassed an extended set of properties and requirements. The focus of all four contributions is dedicated to the network architecture of DCS and the related time behavior. Therefore, a consideration of superordinate systems is not included (*Req-*M_{Sys}). Vogel-Heuser and Ribeiro [VR18] introduce an adapted version of the notation for fog computing on the field level. The approach introduces additional elements to the notation, such as elements for data frames. Another work by Sollfrank et al. [STV19; SVF17] reflects the adaption of the notation for safety applications. Supplementary elements for safety-related hardware, properties, and requirements are presented and used. With the DSL4hDNCS [Vog+20], the group of Vogel-Heuser further unified the separate versions of the graphical modeling notation and extended the graphical modeling notation with a metamodel to yield a DSL. Nevertheless, none of the works by Vogel-Heuser et al. can be used to capture related software functions executed on the hardware (*Req-*M_{Sys}). Additionally, an explicit viewpoint for the flow of data through the system is not considered (*Req-*M_{DF}).

Table 4 summarizes the reviewed modeling approaches. As can be seen, only AADL, MARTE, and UML-PA can model the relevant aspects of the system architecture. None of the presented approaches captures the characteristics of the data flow entirely (*Req-*M_{DF}), which is especially important for data collection and analysis in CPSoS. The only approaches that allow a free definition of properties and requirements (*Req-*$M_{PropReq}$) are the works by Vogel-Heuser et al. Some of the reviewed approaches provide basic sets of annotations that are limited to specific aspects of the systems. Others are only capable of describing either properties or requirements, but not both. Additionally, the requirement of a graphical modeling notation (*Req-*M_{Graph}) is only thoroughly addressed by AADL, ArchiMate, Greifeneder and Frey, as well as Vogel-Heuser et al. All languages can describe specific aspects of the systems (*Req-*M_{Sys}) but at various levels of detail, ranging from detailed to not captured at all. In summary, none of the approaches can fulfill all requirements of the *Req-M* category.

Table 4: Evaluation of relevant approaches in the field of modeling languages for system architectures. See Table 2 for the rating scheme.

Approach	Requirements			
	Req-M$_{Sys}$	***Req-M$_{DF}$***	***Req-M$_{PropReq}$***	***Req-M$_{Graph}$***
AADL	+	○	○	+
ArchiMate	○	○	○	+
AutomationML	○	○	-	-
DesLaNAS	-	○	-	+
Lewin et al.	-	○	-	○
MARTE	+	○	○	○
UML-PA	+	○	○	○
UML-RT	○	○	-	-
SoaML	○	○	-	-
SysML	○	○	○	-
SysML-vAT	○	○	○	-
Vogel-Heuser et al.	○	-	+	+

4.3. Model-driven System Architectures

In the following section, model-driven approaches for the generation of system architectures are surveyed and evaluated. The overview is divided into two parts: the first part summarizes generic approaches for the generation of system architectures. In contrast, the second one is focused on system architectures for industrial automation. Concerning the generic approaches, only requirements of categories *Req-M* and *Req-G* are evaluated as no specific architecture for industrial automation is part of the approaches. Contributions in the second part of this section are evaluated concerning the requirements of all categories.

4.3.1. Generic Architectures

Benaben et al. [Ben+17] present an approach for model-driven engineering of middleware systems. Their approach focuses on the domain of enterprise integration between different companies. The system viewpoint of the underlying metamodel includes some elements that can be applied to industrial automation. However, the focus on enterprise integration leads to strong attention on the modeling of services but not hardware, software, and networks (*Req-M$_{Sys}$*). Moreover, the approach does not comprise a data flow viewpoint (*Req-M$_{DF}$*). Data flows are only implicitly modeled using the activities *invoke*, *receive*, and *reply*. Properties and requirements of the systems are

not captured at all (*Req-$M_{PropReq}$*). Furthermore, the modeling approach is only based on a UML representation but does not consider any graphical modeling constructs (*Req-M_{Graph}*). Benaben et al. use the model and transform it to configure an ESB and related web services for communication (*Req-G_{Com}*).

The groups around Broy and Schätz et al. developed AutoFOCUS [Ara+15; Bau+05; Bro+08; Hub+96], a modeling concept for distributed embedded systems. AutoFOCUS is based on FOCUS [Bro+93] and extends it with a graphical modeling notation (*Req-M_{Graph}*). While the graphical modeling notation only offers a minimal set of shapes, in the system viewpoint (system structure diagram), these can be used to model networks of hardware components. The system viewpoint is limited to data exchange among these hardware components and does not capture network structure or other the type of hardware systems. A second viewpoint for the modeling of data flows can be used to model and characterize modifications and usage of data (*Req-M_{DF}*). However, following the flow of data through systems is challenging due to the very detailed modeling on an embedded hardware level and the implicit formulation of data flows. The modeling notation allows the modeling of timing requirements, such as cyclic execution, but does not differentiate between requirements and properties (*Req-$M_{PropReq}$*). Based on the models, the software code for various hardware platforms can be generated that includes the interactions between the modeled components (*Req-G_{Com}*).

Dorn et al. [DWD14] developed an approach for the automatic generation of message-oriented communication systems. Therefore, they adapt the extensible Architecture Description Language (xADL) by Dashofy et al. [DvT01] with additional elements. These elements describe message-oriented communication in general and their respective implementations for the ESBs ActiveMQ and Mule. Their description of the system architecture is only focused on communication interfaces and the abstract description of components (*Req-M_{Sys}*). The data flow is modeled with more detail in comparison to other approaches (*Req-M_{DF}*). However, only endpoints, communication channels, and the direction of information flows can be modeled. Neither the description of properties/requirements nor a graphical modeling notation is considered (*Req-$M_{PropReq}$*, *Req-M_{Graph}*). For the development process of the communication systems, tool support is provided. A model transformation can then automatically generate runtime configurations for the middleware but no client code (*Req-G_{Com}*).

Ebeid et al. [Ebe+15; EFQ15] extend the UML MARTE profile with network-related aspects and introduce a model-driven generation of runnable configurations for distributed embedded systems. Their extension to UML MARTE captures aspects of QoS and defines abstract data channels that are used to transport data from one system to another. Concerning the *Req-M* requirements, the

approach shares the same characteristics as the UML MARTE profile itself. Ebeid et al. consider an automatic generation of communication interfaces but restrict the approach to the model-driven generation of configurations for a simulation environment (*Req-G_{Com}*).

With the so-called ThingML, Harrand et al. [Har+16] present a textual, domain-specific modeling language for embedded IoT devices (*Req-M_{Graph}*). ThingML encompasses basic system and data flow viewpoints but focuses on embedded devices and low-level interactions between them (*Req-M_{Sys}*, *Req-M_{DF}*). ThingML does not provide language constructs for adding properties or requirements to the model (*Req-$M_{PropReq}$*). Based on the models, basic communication interfaces are automatically generated by model transformations (*Req-G_{Com}*).

Issarny et al. conceptualized different model-driven communication architectures, for instance, the extensible service bus (XSB) [Geo+13] and the eVolution Service Bus (VSB) [Bou+19; Bou17; Iss+16]. The focus of both approaches lies in the domain of the Internet of Things. Therefore, the service buses use web services. The necessary communication interfaces are automatically generated based on the model (*Req-G_{Com}*). The underlying models do not encompass a system viewpoint that can describe hardware, software, and networks (*Req-M_{Sys}*). Data flows are implicitly modeled with so-called mash-up graphs (*Req-$M_{DF,}$ Req-M_{Graph}*). These graphs show the path of data through the system and the dependencies between components. Nevertheless, no differentiation of the specific roles of components is made. Bouloukakis et al. [Bou+19] extend the approach by introducing a uniform software framework that abstracts the functional properties of specific IoT protocols.

Petrasch [Pet17; Pet18] presents a model-based approach for the development of microservice architectures. The system viewpoint is limited to a basic description of the system related to enterprise integration (*Req-M_{Sys}*). A data flow viewpoint (*Req-M_{DF}*) does not exist. Furthermore, it is not possible to annotate the models with properties and requirements (*Req-$M_{PropReq}$*). The approach does not include a graphical modeling notation and is based on UML (*Req-M_{Graph}*). The work by Petrasch includes the concept of a model-driven generation of communication interfaces, but implementation and demonstration of this functionality are missing (*Req-G_{Com}*).

Pusztai et al. [PTD19] propose a model-based approach for the development of IoT applications. Based on a new UML profile, the modeling of heterogeneous IoT devices is possible. The introduced stereotypes are tailored for embedded IoT hardware and include basic components such as CPUs. However, the modeling of networks and specific devices relevant for industrial automation is not in the scope of the approach (*Req-M_{Sys}*). Data flows can be modeled implicitly using activity diagrams (*Req-M_{DF}*). The modeling approach does not include model elements for the capturing of properties and requirements (*Req-$M_{PropReq}$*), nor does it include a graphical notation

(*Req-M_{Graph}*). Based on the models, the code for communication between the systems over REST is generated (*Req-G_{Com}*).

Tekinerdogan et al. [TÇK18] present an approach to simulate and find optimized deployment scenarios for DDS systems. Various deployment scenarios are automatically generated and tested for the fulfillment of specific requirements using simulation. Therefore, they developed an approach that allows the modeling of applications and physical resources, for instance, available memory and processing power, but no actual hardware devices (*Req-M_{Sys}*). Furthermore, the approach includes the possibility to annotate the models with requirements (*Req-$M_{PropReq}$*) that a deployed architecture must fulfill. Nevertheless, no actual properties are considered. Also, a data flow viewpoint (*Req-M_{DF}*), a graphical notation (*Req-M_{Graph}*), and a model-driven generation of communication interfaces (*Req-G_{Com}*) are not part of the approach.

Terzić et al. [Ter+18] developed the model-driven tool MicroBuilder. MicroBuilder includes a framework for the automatic generation of REST microservices for e-commerce applications. Therefore, they automatically set up communication interfaces based on a specified model (*Req-G_{Com}*). The underlying modeling language is a mixture of graphical notation and textual, domain-specific language (*Req-M_{Graph}*). Due to the different domain, no system viewpoint, including elements for industrial automation, is considered (*Req-M_{Sys}*). Simple data flows can be modeled using the graphical notation (*Req-M_{DF}*). However, no annotation with properties or requirements is considered (*Req-$M_{PropReq}$*).

4.3.2. System Architectures for Industrial Automation

The only implemented model-driven system architecture for data collection is presented by Mazak et al. [Maz+18]. Their approach is based on an extended version of AutomationML. Therefore, they add the description of data dependencies between systems. The extended model is then used to automatically set up OPC UA servers as data providers and a data collection architecture. The collected data is finally stored in a time-series database. While their approach is focused on data collection from the field level, it can be used to collect data from various levels (*Req-A_{ATP}*). As the architecture is a data collection architecture, a parallel operation to the existing control infrastructure is foreseen (*Req-A_{POP}*). Due to the focus on OPC UA, neither the concept is technology-agnostic (*Req-A_{TAC}*), nor a standardized interface that abstracts between different communication technologies is defined (*Req-SF_{ACP}*). The integration of legacy systems is not considered (*Req-SF_{Leg}*). While Mazak et al. claim decreased costs for re-engineering during the evolution of CPPS, the contribution does not evaluate or measure a decreased effort during initial deployment (*Req-A_{ReDep}*) nor re-deployment (*Req-A_{ReDep}*). As the approach is based on AutomationML, the same limitations concerning the systems viewpoint (*Req-M_{Sys}*), properties/requirements

(*Req-$M_{PropReq}$*), and the non-graphical representation (*Req-M_{Graph}*) apply. The flow of data (*Req-M_{DF}*) is implicitly modeled over the introduced dependencies, but it cannot be followed through the system or over a multi-stage process. The approach automatically generates all necessary communication interfaces (*Req-G_{Com}*), including the OPC UA server, the respective client components, and the database connection.

Hufnagel et al. [HFV13; HV15] present a concept facilitating the collection of distributed and heterogeneous data based on ESB-principles. The proposed, technology-agnostic architecture (*Req-A_{TAC}*) uses data mapping and adapters to integrate near real-time and batch data from different systems, including legacy systems (*Req-SF_{Leg}*). Data collection from various levels of the automation pyramid (*Req-A_{ATP}*) and parallel operation (*Req-A_{POP}*) are not considered but could be applied in theory. The modeling approach includes elements for modeling the system architecture but only related to the communication with the common data backbone, not the individual systems themselves (*Req-M_{Sys}*). The approach does not foresee the modeling of any other aspects (*Req-M_{DF}*, *Req-$M_{PropReq}$*) nor provide a graphical modeling notation (*Req-M_{Graph}*). The underlying model-based development of the data exchange does encompass a concept for the model-driven generation of the architecture itself (*Req-G_{Com}*). However, no publication of the concept's practical implementation is available.

Based on the UML4IoT metamodel [TC16], Thramboulidis and Christoulakis [TVS18] conceptualized a model-driven generation of microservice architectures for CPPS. Their developed architecture aims at replacing the existing infrastructure (*Req-A_{POP}*) and focuses on the field level but could be applied to interface superordinate systems as well (*Req-A_{ATP}*). The Lightweight Machine to Machine protocol (LwM2M) [Ope18] is used, which makes the concept tailored to this specific communication technology (*Req-A_{TAC}*) and prevents a further abstraction by the standardized interface (*Req-SF_{ACP}*). The concept does not consider the integration of existing legacy systems into the architecture (*Req-SF_{Leg}*). Concerning the included metamodel, the system viewpoint focuses on resources and their information exchange but lacks consideration of hardware, software, and networks (*Req-M_{Sys}*). The metamodel does not comprise a data flow viewpoint (*Req-M_{DF}*) nor the possibility of annotations with properties and requirements (*Req-$M_{PropReq}$*). All modeling is non-graphical (*Req-M_{Graph}*). The actual model transformation into a deployable architecture is only developed conceptually but not demonstrated (*Req-G_{Com}*).

Table 5 summarizes the evaluation of model-based system architectures. Only the approaches by Mazak et al. [Maz+18] Hufnagel et al. [HFV13; HV15], and Thramboulidis et al. [TC16; TVS18] were evaluated concerning the requirements of categories *Req-A* and *Req-SF*, as they are the only approaches that generate a system architecture in the field of industrial automation. Furthermore,

none of the approaches that encompass a model-driven generation of the architecture features a complete domain-specific language, including a graphical modeling notation for data collection architectures.

Table 5: Evaluation of relevant approaches in the field of model-driven system architectures. Non-relevant criteria are grayed out. See Table 2 for the rating scheme.

Approach	**Requirements**												
	Req-A$_{ATP}$	***Req-A$_{TAC}$***	***Req-A$_{POP}$***	***Req-A$_{Dep}$***	***Req-A$_{ReDep}$***	***Req-SF$_{API}$***	***Req-SF$_{ACP}$***	***Req-SF$_{Leg}$***	***Req-M$_{Sys}$***	***Req-M$_{DF}$***	***Req-M$_{PropReq}$***	***Req-M$_{Graph}$***	***Req-G$_{Com}$***
Benaben et al.									○	-	-	-	+
Broy and Schätz									○	○	○	+	+
Dorn et al.									○	○	-	-	○
Ebeid et al.									+	○	○	○	∧
Harrand et al.									○	○	-	-	+
Hufnagel et al.	○	+	○	?	?	+	-	○	○	-	-	-	∧
Issarny et al.									-	○	-	○	+
Mazak et al.	○	-	+	?	?	+	/	-	○	○	-	-	+
Petrasch									○	-	-	-	∧
Pusztai et al.									○	○	-	-	+
Tekinerdogan et al.									○	-	○	-	-
Terzić et al.									-	○	-	○	+
Thramboulidis et al.	○	-	-	?	?	+	-	-	○	-	-	-	○

4.4. Research Gap in Model-driven Development of Data Collection System Architectures

The reviewed approaches and their respective fields of contribution are summarized in Figure 16. As can be seen, a large number of approaches that consider system architectures exist. On the other hand, several distinct modeling notations for distributed systems were identified. Nevertheless, only five approaches exist that encompass a modeling approach (*Req-M*) for industrial automation systems, namely AutomationML, as well as the approaches by Mazak et al., Hufnagel et al., Thramboulidis et al., and Vogel-Heuser et al. (light gray and dark gray areas in Figure 16).

Based on the *Req-G* requirements, the model-driven generation of system architectures was considered. The majority of identified approaches are either dedicated to REST web services or the field of system architectures for consumer IoT devices. Only Mazak et al., Hufnagel et al., and

Thramboulidis et al. (dark gray area in Figure 16) present approaches that apply to the model-driven generation of system architectures for the industrial automation domain (*Req-G*). However, the three approaches only consider parts of a data collection architecture and lack an explicit data flow description and a domain-specific language with a visual notation. Additionally, Mazak et al., as well as Thramboulidis et al., support only the usage of a single communication protocol.

Moreover, the evaluation of model-driven approaches should include suitable metrics and at least semi-industrial use-cases, as identified by Wortmann et al. [Wor+20]. However, none of the surveyed approaches proofed a reduction of implementation efforts using suitable metrics.

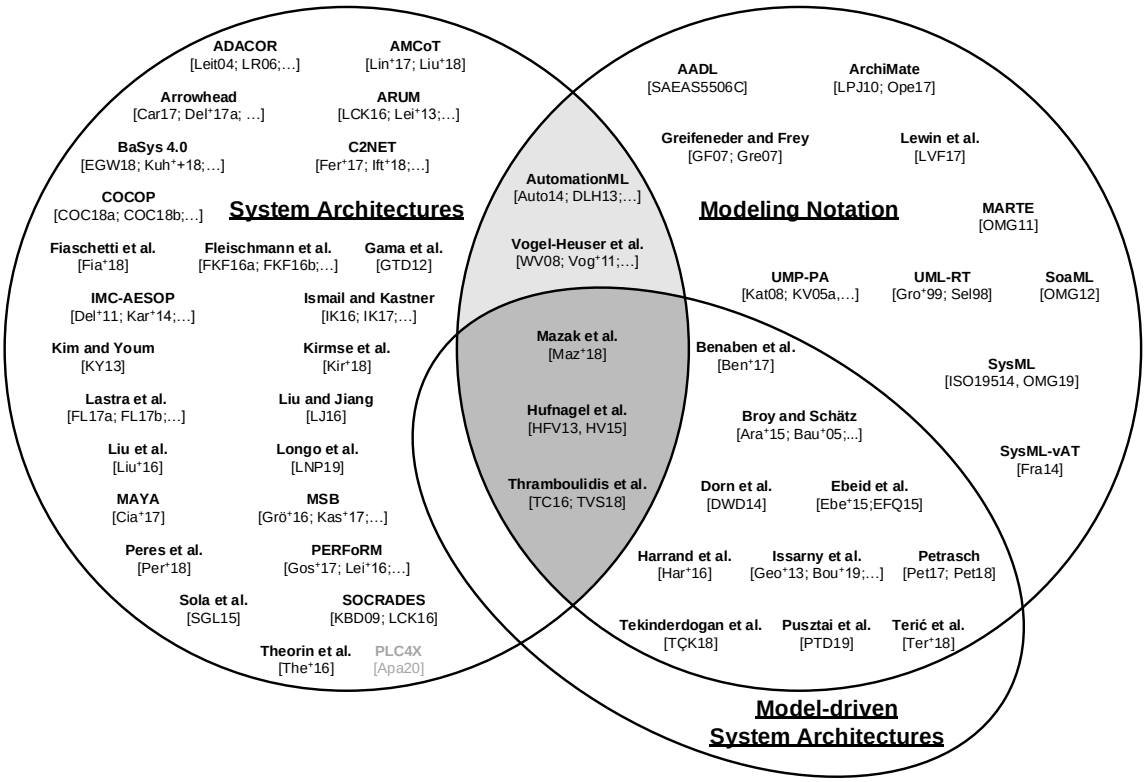


Figure 16: Overview of relevant state-of-the-art contributions, their field of contribution, and identified research gap. The research gap is highlighted in gray.

Therefore, the research gap that is addressed within this thesis is identified as:

Research gap

A model-driven approach for data collection based on a domain-specific language with a visual notation for the formal description of systems and associated data flows does not exist in the domain of industrial automation. None of the surveyed approaches provides the means to generate necessary communication interfaces for data collection automatically. Furthermore, support for multiple communication protocols is lacking.

5. Approach for Model-driven Development of Data Collection Architectures

This Section describes the concepts for modeling and model-driven generation of data collection architectures. First, an overview of the entire approach and its building blocks is given. Subsequently, a detailed description of each sub-concept is presented in the following subsections.

To address the research gap identified in the previous Chapter, the approach is constituted of four sub-concepts, which address the different requirement categories from Chapter 3. These sub-concepts are the generic architecture concept for interoperability and connectivity (1, *Req-A*), the graphical modeling notation and the underlying metamodel that constitute the domain-specific language (2, *Req-M*), the software framework to abstract the specific properties of different communication technologies (3, *Req-SF*), and the model-driven generation of the data collection architecture (4, *Req-G*). Figure 17 illustrates how the separate building blocks depend on each other.

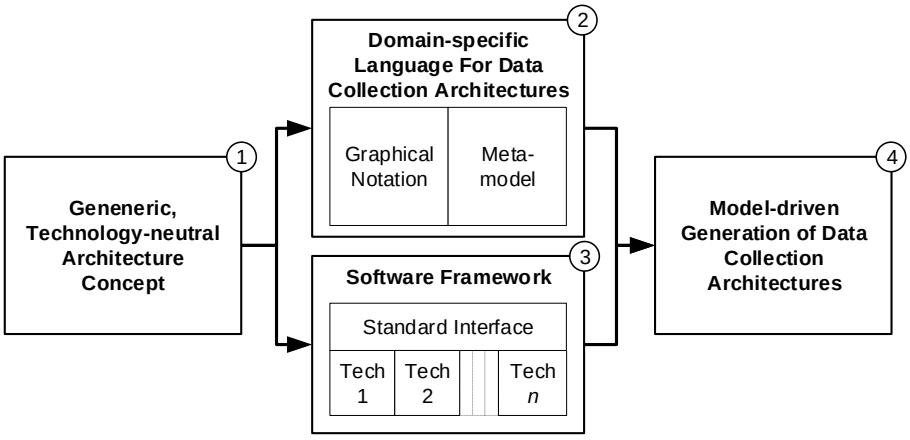


Figure 17: Building blocks of the concept. Generic, technology-neutral architecture concept (left,1), the domain-specific language for data collection architectures including the graphical notation and the metamodel (top, 2), the software framework for different communication technologies (bottom, 3), and the model-driven generation of the data collection architecture (right, 4).

The architecture concept (see Section 5.1) describes the overall, technology-neutral concept of the data collection architecture for industrial automation. Therefore, it is designed with a focus on the domain of industrial automation to ensure the fulfillment of the requirements *Req-A* category.

The domain-specific language (see Section 5.2) that addresses requirements from the *Req-M* category includes a graphical modeling notation (concrete syntax) for the description of data collection architectures in industrial automation. It is based on an underlying metamodel (abstract syntax) which describes the basic concepts and rules. Furthermore, it formalizes and structures the modeled information.

A supplementary software framework (see Section 5.3) is conceptualized to support various communication technologies for industrial data collection architectures. The specifics of the technologies are abstracted and unified by a standard communication interface. It is focused on the requirements from category *Req-SF*.

As the last building block, the model-driven generation of the data collection architecture is included (see Section 5.4). It uses formalized information from architecture descriptions that are based on the DSL. Furthermore, it employs the functionality of the software framework to unify and abstract the communication code for data collection and manipulation. The requirements from the *Req-A* and *Req-G* category are particularly crucial for this sub-concept.

Figure 18 reflects the proposed integrated, model-driven workflow. Based on an existing CPSoS (brownfield) or a conceptualized system (greenfield), a suitable data collection architecture is designed by an interdisciplinary expert team. This team is made up of automation engineers, process experts, IT architects, data analysts, and programmers. Following the guidelines of the architecture concept, the architecture is described. For this purpose, the vocabulary, rules, and graphical elements of the DSL are used. After incremental refinement of the conceptualized data collection architecture by the experts, a final architecture description is established. This description serves as the basis for the model-based generation of the data collection architecture. In an M2T transformation step, a preconfigured architecture is generated, which is based on code templates from the software framework. The preconfigured architecture encompasses the configured communication part of the architecture and placeholders for custom code (OSI layer 7). This preconfigured architecture is, in the next step, extended with custom code fragments to add the specific functionalities of the architecture before it is deployed to the CPSoS.

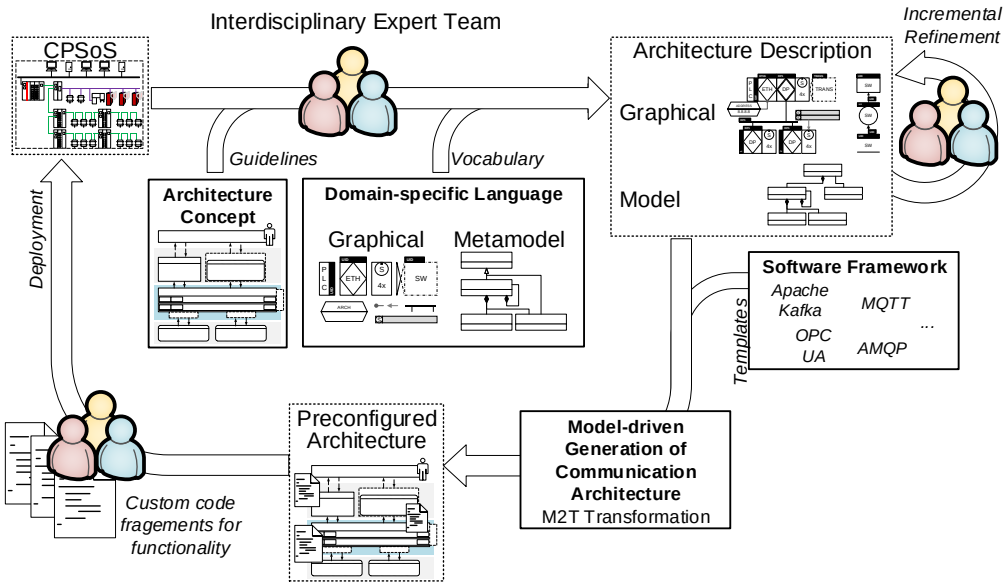


Figure 18: Workflow for model-driven development of data collection system architectures.

5.1. Technology-neutral Architecture Concept

The concept of the data collection architecture is based on previous work by the author and extends the published concepts [TLV18; Tru+17; Tru+19b; Tru+19c].

Industrial automation systems are characterized by a multitude of heterogeneous systems situated on different levels of the automation pyramid. The architecture needs to support the data collection process from these distributed systems (*Req-*A_{ATP}). To minimize the impact of the data collection process on the control of the system, parallel operation of the data collection architecture to the existing automation pyramid is desirable (*Req-*A_{POP}).

One of the major obstacles for the implementation of data analysis projects in industrial processes is the significant effort for interfacing the multitude of heterogeneous systems [Bis+99; Pei19]. Existing legacy systems with proprietary interfaces further complicate the task. The results are often ad-hoc implementations for specific data analysis projects that result in hard to maintain meshed P2P communication networks. Modifications or updates concerning information models, communication interfaces, and available communication protocols on one of the connected systems result in the need to update all related communication interfaces. The architecture concept should, therefore, decrease the necessary implementation efforts for the initial deployment of a data collection and analysis infrastructure (*Req-*A_{Dep}), as well as for redeployment or migration scenarios (*Req-*A_{ReDep}).

As the implementation effort for data collection architectures is strongly related to the number of necessary communication channels, the application of middleware concepts can be beneficial. The middleware mediates between all connected systems and allows transparent data access (see Figure 19). The definition of a standardized interface for the connection of systems unifies the data collection process. Legacy systems that are not compatible with the newly introduced standard interface must be interfaced using data adapters. These data adapters translate between the legacy systems and the standardized interface. Greenfield applications that are implemented following the standard interface do not need any further mediating step. They are compatible with the middleware out-of-the-box. Figure 19 reflects the data collected from different levels of the automation pyramid, as well as a parallel operation not interfering with the existing infrastructure for control. Legacy systems, especially the existing automation systems (e.g., legacy PLCs) residing in the automation pyramid, are interfaced using data adapters. The middleware acts as a mediating bus that allows transparent data access from all connected systems, called participants. The participants that are connected to the architecture are systems that include hardware as well as software functionalities.

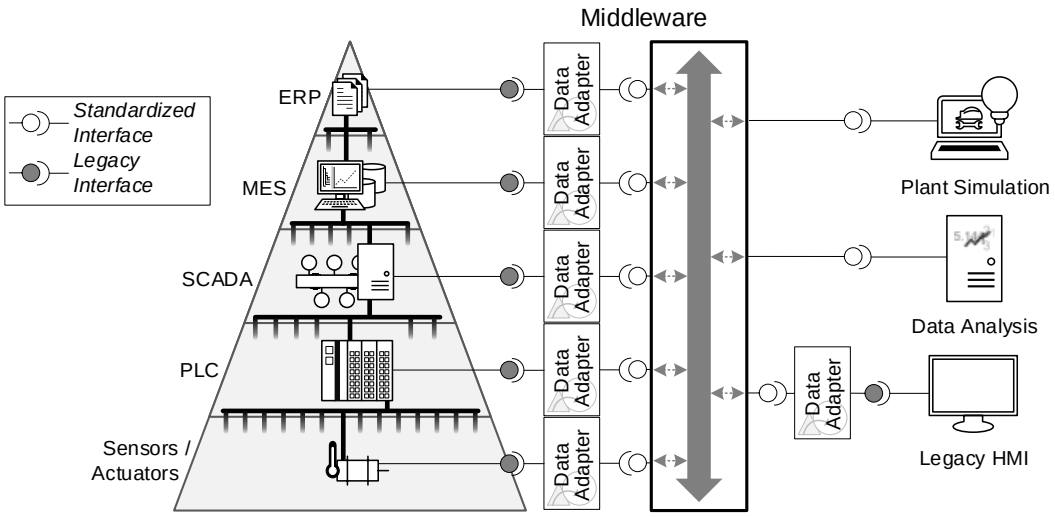


Figure 19: High-level concept of the data collection architecture.

Therefore, for *n* participants *N* communication channels have to be implemented for transparent data access in comparison to $\frac{N\cdot(N-1)}{2}$ interfaces for a completely connected P2P mesh (see Section 2.3.2). A comparison between the necessary communication channels for transparent data access across all systems is depicted in Figure 20 as a function of the connected systems *n*. As can be seen, the number of communication channels for the middleware approach is significantly decreased if more than three systems are connected.

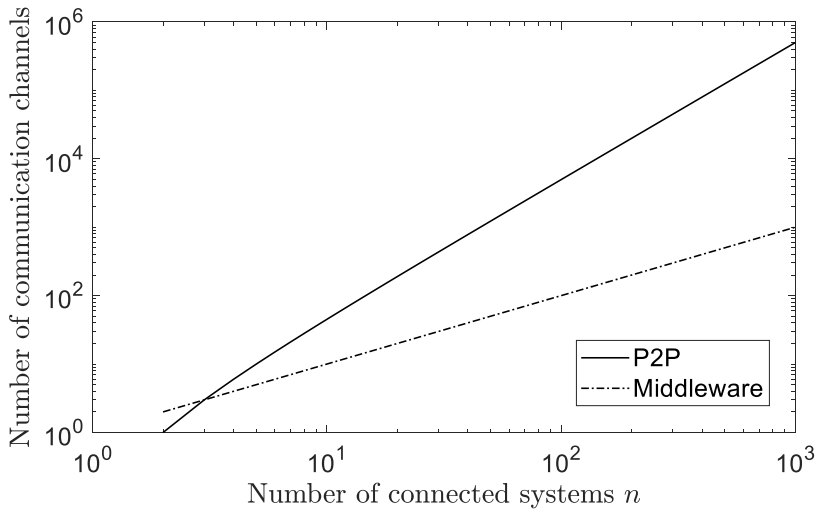


Figure 20: Number of necessary communication channels for transparent data access as a function of the number of connected systems n for a fully connected mesh (P2P) and a middleware network.

The middleware acts as a secondary communication channel following the NOA concept [Kle+17; NE175]. It allows the vertical and horizontal integration of data from the automation pyramid: systems on different levels of the hierarchy can be interfaced and their data made accessible; besides, transparent data exchange is possible for participants on the same hierarchical layer.

The detailed middleware concept is depicted in Figure 21. Distinct functional layers are introduced to increase the modularity of the concept, namely *Data*, *Integration*, *Analysis*, and *Dashboard*. The architecture's heart is the *Data Management and Integration Broker* in the *Integration* layer. It acts as the middleware component of the architecture and mediates between the participants.

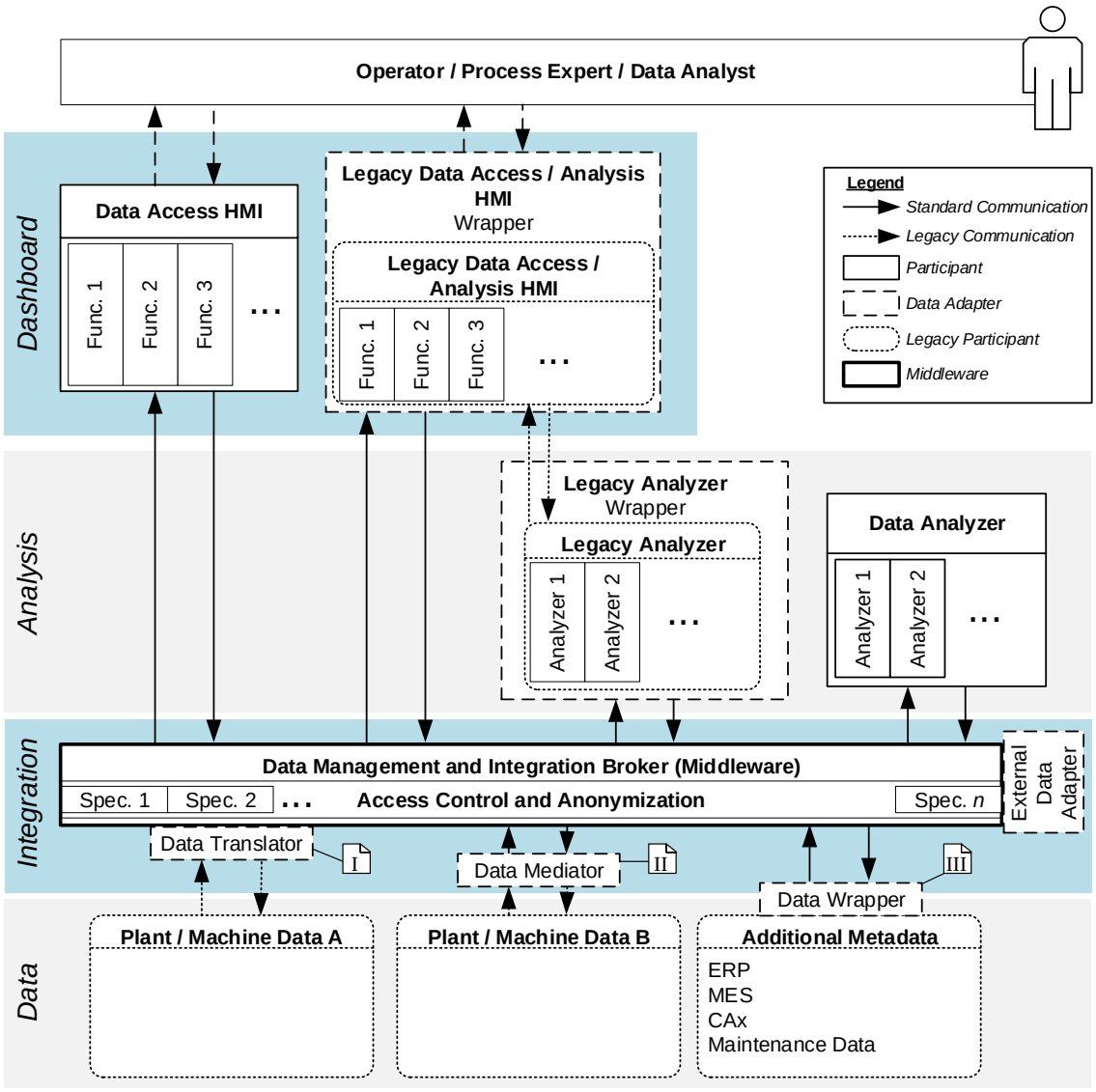


Figure 21: Detailed concept of the data collection architecture (adapted from [Tru+19c]).

Data is received by the broker and distributed to all participants, which are interested in this specific piece of data. The *Data Management and Integration Broker* features central rights management (*Access Control and Anonymization*). Access rights to datasets are managed and controlled by the middleware itself, ensuring that no sensitive information is leaked to non-authorized participants. The anonymization component can anonymize data before it is distributed to clients. For instance, data could be normalized, artificial noise could be introduced, or the sampling rate of the data could be decreased. This lowers the information content of the data and prevents the leaking

of sensitive information. Centralizing these functionalities on the *Integration* layer minimizes redundancies in the architecture as the participants can rely on the middleware. Furthermore, as data processing by the participants is out of the limits of the central *Integration* layer, it could be compromised and hence not trustworthy. The trust in the system can be increased by keeping this functionality at a central instance. An *External Data Adapter* allows the connection of multiple instances of the broker for a separation of concerns, for instance, across different production sites or even companies (inter-enterprise data exchange).

The *Data* layer includes systems that function as data sources and may receive processed information. Participants that are part of the automation pyramid always reside in the *Data* layer.

The third layer, the *Analysis* layer, includes systems that provide advanced functionalities executed on the data. Typically, any data analysis, simulation, and optimization belong to this layer.

The last layer is the *Dashboard* layer, which is used to communicate with humans. Operators, experts, or data analysts can visualize data from the *Data* layer and results from the *Analysis* layer.

All participants communicate over the central *Data Management and Integration Broker* without direct P2P connections (see solid lines in Figure 21). Therefore, a standard interface is used that allows all participants to communicate in a unified way. The standardized interface defines how data can be accessed and forwarded. The principles of the standardized interface and data adapters are illustrated in Figure 22.

The standardized interface provides the necessary functionalities for communication with other systems and can be realized with different technologies (*Req-*A_{TAC}). Existing connections between legacy systems can be left in place if access to the transported data is not necessary for other systems outside the legacy connection (the dotted connection between *Legacy Analyzer* and *Legacy Data Access / Analysis HMI* in Figure 21). Keeping these existing connections helps to minimize the development effort, as existing connections can be retained.

While newly developed participants can make direct use of the standardized interface and implement it for communication (see Figure 22 (left), I), existing legacy participants are interfaced using data adapters that translate the protocol (syntax, OSI layers 4 to 7) and understanding (semantic, information model) between legacy and standard representation. Different concepts for data adapters exist, depending on the location where the translating logic (*Translator*) is executed. These are, from left to right in Figure 22:

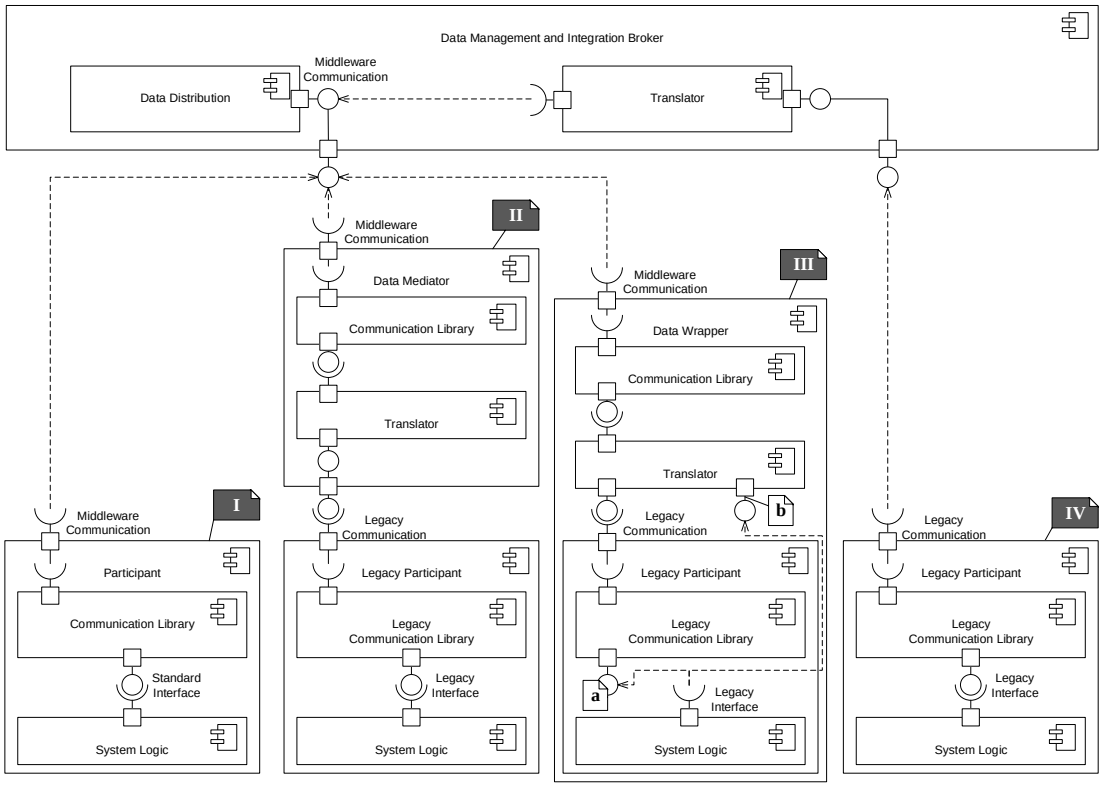


Figure 22: Principle of the technology-neutral, standardized interface to integrate greenfield and brownfield participants. Greenfield participants (left, I) need no adapter. Legacy participants need adapters: independent data mediator (second from left, II), integrated data wrapper (third from left, III), and data translator in Data Management and Integration Broker (right, IV).

- II: data mediators that constitute independent systems. These systems receive the legacy data over an interface that is provided by the legacy component. The *Translator* subsystem translates syntax and semantic between the incompatible legacy and standard representations. Communication with the broker uses the standard communication library;
- III: data wrappers that form integrated systems with the legacy systems they wrap. From the outside, only the standard interface-compliant wrapper is visible to the architecture. The legacy system is entirely wrapped inside the wrapper. Translation and communication with the middleware follow the same principle as in the data mediators. Communication between *Translator* and the legacy system's *System Logic* can either be handled through the legacy communication library (III, a) or direct access (III, b);
- IV: data translators on the broker-level as an integral part of the *Data Management and Integration Broker*. For this kind of data adapter, the middleware must also provide the legacy interface for communication.

Throughout this thesis, all three kinds of adapter concepts are summarized under the term data adapter. The distinct concepts have their strengths and weaknesses. Depending on the specific use-case, one may choose a suitable data adapter concept. As the retrofitting of existing legacy systems

with data adapters is a very time-consuming task, a step-wise deployment and refinement of the architecture are proposed. The initial deployment should focus on systems that heavily depend on each other to benefit from the decreased number of interfaces. Over time, more and more systems can be migrated in small, manageable steps. This step-wise approach minimizes the effort for initial deployment at the tradeoff of incomplete data access. [Bis+99; Cal+17]

While Figures 19 to 22 all illustrate a central *Data Management and Integration Broker*, they only refer to the function of the component, not its physical location. The broker can be implemented using different sets of technologies, centralized or distributed. The presented concept can be adapted and implemented for a wide variety of use-cases (*Req-*A_{TAC}) regardless of actual middleware technology and communication protocol used (see Section 2.3.2), the mix of programming languages for implementation, or the actual realization of the adapters. The architecture concept serves as a basis for practical implementations and their description using the DSL.

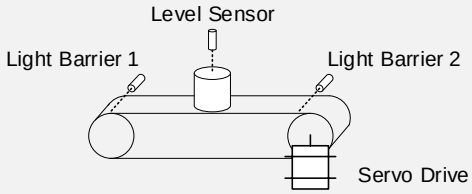
5.2. Domain-specific Language for Data Collection Architectures

The DSL for data collection architectures developed in this thesis, as the definition of the term modeling language requires [Rod15], consists of a metamodel (abstract syntax) and a graphical notation (concrete syntax). Both are introduced in the following two subsections. Experts can model information using the graphical modeling notation (*Req-*M_{Graph}). The modeled information is then structured as instances of the metamodel, which can be used for the subsequent model-driven generation of the data collection architecture. Class and interface names that are part of the metamodel are highlighted in *italic* in the text.

Throughout the following parts of this Chapter, a simple application example indicated by grey boxes and the caption “AE.Part” will be used to introduce the application of the DSL. After each explanation of a subpart, the presented concepts are applied to the application example to reflect their specific usage. However, due to the simplicity of the example, not every introduced concept can be found, or the depth of modeling is limited. The introduction of the example can be found in AE.Part 1. A list of all references to the application example can be found in Chapter 13.

AE.Part 1: Introduction of the physical application example.

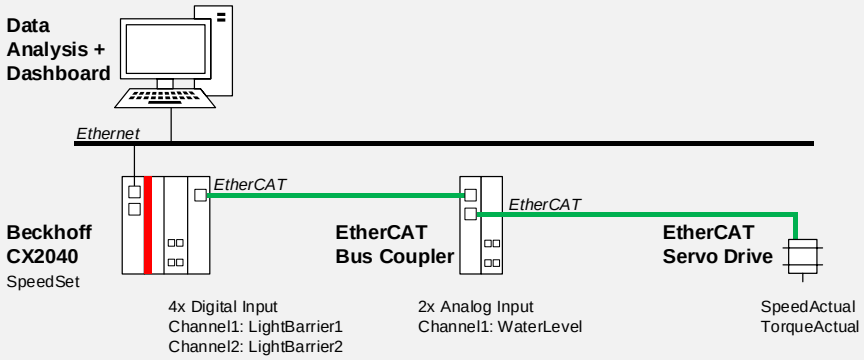
The application example contains a conveyor belt driven by a servo drive. Light barriers are located on both ends of the conveyor belt and can detect workpieces entering or exiting the conveyor belt. A level sensor is located over the center of the belt and measures the filling level of water inside the workpieces. See AE.Figure I for a schematic drawing:



AE.Figure I: Schematic drawing of the physical setup.

The conveyor belt is subject to constant wear, which leads to unexpected downtimes. A new monitoring application is to be set up to monitor the condition of the belt. The monitoring is based on an anomaly detection algorithm that indicates the probability of an abnormal situation as a so-called anomaly score. A dashboard is to be installed that displays the calculated anomaly score to the operating personnel. Therefore, a PC executing the analysis function is added to the system. This PC is connected to the PLC of the conveyor belt via Ethernet.

The belt is controlled by a central Beckhoff CX2040 PLC that executes the control logic. Inside this PLC program, the set speed (variable *SpeedSet*) of the conveyor belt is calculated. A four-channel digital input terminal is directly attached to the PLC and connected to the two light barrier sensors (channel 1 is connected to *LightBarrier1* sensor on the left of the belt and channel 2 to *LightBarrier2* sensor on the right of the belt, the other two channels are not connected). Additional signals are interfaced over an EtherCAT bus connected to the PLC. The first bus coupler in the bus has a two-channel analog input terminal attached, which is connected on channel 1 to the level sensor (*WaterLevel*). The bus is finally terminated in the servo drive that is directly connected to the EtherCAT bus. The servo drive includes internal control logic and provides signals for the actual speed of the drive (*SpeedActual*) as well as the measured torque (*TorqueActual*). A schematic view of the network, the connected devices, as well as the sensor and actuator signals, are depicted in AE.Figure II:



AE.Figure II: Schematic drawing of the hardware components and input/outputs.

5.2.1. Communication Architecture Metamodel

The metamodel describes basic concepts of data collection architectures in the domain of industrial automation (abstract syntax). It structures the modeled information and makes it accessible for a subsequent model-driven generation of the data collection architecture. An overview of the base elements of the metamodel is given in Figure 23. These base elements and the associated classes are separately explained in the following. The base element of the metamodel is a concrete *Architecture* that is described by an *ArchitectureDescription* (compare ISO 42010 [ISO42010]). This *ArchitectureDescription* comprises a *ConfigurationContainer,* which in turn includes the modeled *SystemConfiguration* of the described architecture. The configuration of the system is divided into distinct categories:

- *SoftwareContainer* which describes the software functionalities that are part of the system and the flow of data/information between them;
- *PhysicalContainer* which describes the hardware components of the system;
- *RelationContainer* which maps software functionalities, networks connections, and data elements from the *SoftwareContainer* to hardware elements in the *PhysicalContainer*; and
- *AnnotationContainer* which includes and structures properties, requirements, and additional annotations.

The introduction of these sub-containers facilitates a strong separation of concerns when modeling and annotating complex, and intercorrelated systems constituted of hard- and software. A reduced version of the metamodel's general structure was published by the author as part of the DSL4hDNCS [Vog+20].

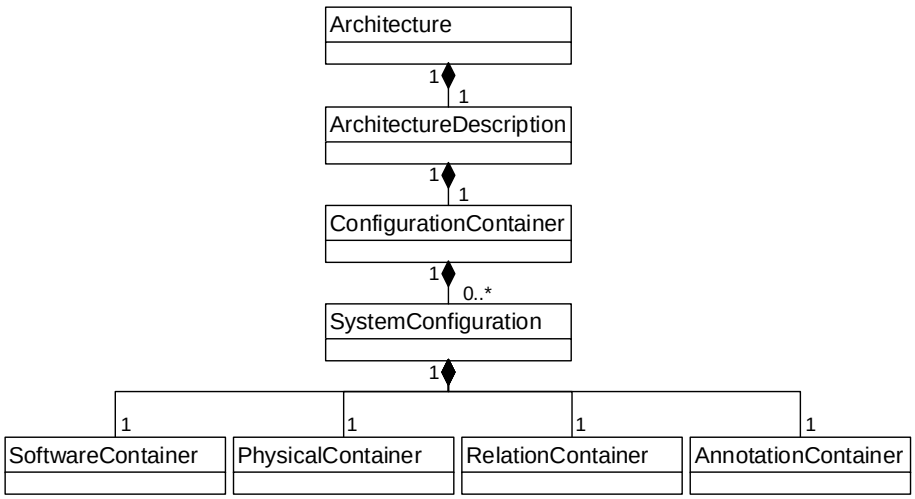


Figure 23: Overview of the general structure of the metamodel.

SoftwareContainer

Software functionalities are an essential part of data collection architectures. They describe the logic that generates, manages, modifies, processes, and transmits the collected data. The execution of a software functionality always requires a hardware processing unit associated with it (see Subsections PhysicalContainer and RelationContainer). Software functionalities in connected systems communicate with each other and exchange data and information. This data exchange between software functionalities can either be local between software functionalities on the same physical system or via a network. The structure and interaction of software functionalities are formalized and described in the *SoftwareContainer*, which is depicted in Figure 24. The *SoftwareContainer* bridges the system viewpoint (*Req-*M_{Sys}), and the data flow viewpoint (*Req-*M_{DF}).

All configuration elements that are aggregated by the *SoftwareContainer* derive from the base interface *ISoftwareConfigurationElement*. The metamodel differentiates between a platform-independent configuration (*IPlatformIndependentElement*) and a platform-specific configuration (*IPlatformSpecificElement*) (compare OMG MDA [OMG14]). While the platform-independent branch describes the abstract roles of software functionalities, data flows between them, and the exchanged data, the platform-specific branch describes the concrete technologies and configurations. This separation allows the definition of the software part of the architecture on two levels and increases the reusability of the information in the platform-independent branch. Furthermore, it reflects the workflow during the engineering of software systems with abstract descriptions in the beginning and their mapping to technologies during the workflow.

Every connected software system is a so-called *SoftwareFunctionality* and can process and communicate data. *SoftwareFunctionalities* include custom code for actions carried out on data (*ApplicationSpecificLogic*),e.g., data manipulations and calculations, and communication services (*IService*) for communication with other *SoftwareFunctionalities*. A service can either produce (*IProducer*), consume (*IConsumer*), or consume and produce data (*IConsumerProducer*, derived from *IProducer* and *IConsumer*) [OMG12]. Producers and consumers are connected over a *DataFlow* that describes data exchange between two *SoftwareFunctionalities*. As *SoftwareFunctionalities* can aggregate multiple services, complex interactions are possible; for instance, consumption of several data flows and offering of the processed data over two different *DataFlows*. This kind of complex interactions is often found in industrial automation, for instance, SCADA system that aggregate data from various PLCs and forward part of the data to a dashboard while another part of the data is provided for a superordinate MES system.

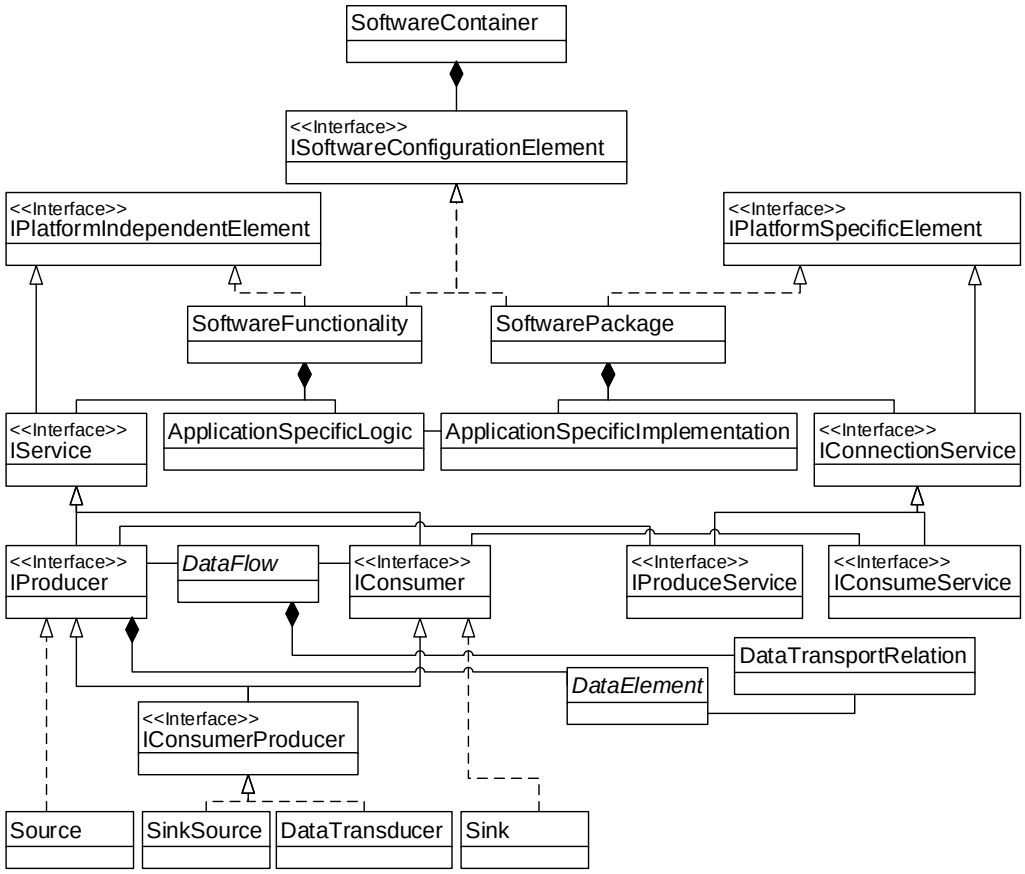


Figure 24: Detail of the metamodel's SoftwareContainer for the description of the software. Platform-independent part (left) and platform-specific part (right) allow the description at distinct levels of abstraction.

As introduced in *Req-M_{DF}*, the flows and manipulations of data are highly relevant in industrial data analysis. Thus, the metamodel needs to encompass elements to describe these aspects, and the concrete classes *Source*, *SinkSource*, *DataTransducer*, and *Sink* are introduced. These describe the roles of *SoftwareFunctionalities* related to the flow and manipulation of data (see Table 6).

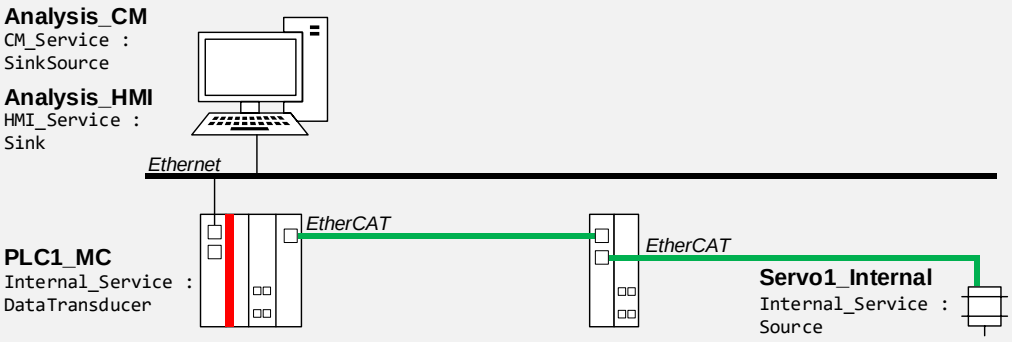
Table 6: Types of data manipulation considered in the metamodel.

Class	Derived from	Description
Source	*IProducer*	Origin of a *DataFlow*. *Sources* cannot consume any data but, instead, provide data to other *SoftwareFunctionalities*.
SinkSource	*IProducer* and *IConsumer*	*SinkSources* consume and produce data at the same time. Data that flows in can be processed or altered but is not automatically available on the producing side of the *SoftwareFunctionality*. In other words, inflowing *DataFlows* are terminated by a *SinkSource*, and only explicitly defined data is offered to other *SoftwareFunctionalities*.
DataTransducer	*IProducer* and *IConsumer*	*DataTransducers* consume and produce data at the same time. Data that flows in can be processed and is transparently and unaltered available on the producing side of the *SoftwareFunctionality*. Additionally, calculated or measured data can be made available as well.
Sink	*IConsumer*	The end of a *DataFlow*. *Sinks* only consume data but cannot produce or forward any data.

AE.Part 2 demonstrates the modeling of the software functionalities and the data flows for the application example.

AE.Part 2: Modeling SoftwareFunctionalities and DataFlow.

The application example contains different software functionalities and data flows that should be described and formalized using the metamodel. Therefore, the classes of the metamodel are instantiated as concrete objects representing the information to be modeled. Every software component of the system can be modeled as a *SoftwareFunctionality* (see Figure 24), which is composed of the internal programming logic of the functionality (*ApplicationSpecificLogic*) and a description of the function inside the data flow (*IService*). AE.Figure III reflects the mapping of the *SoftwareFunctionalties* to the respective hardware systems, as well as the definition and role of *IService* instances.



AE.Figure III: Mapping of the SoftwareFunctionalities to the components and IServices.

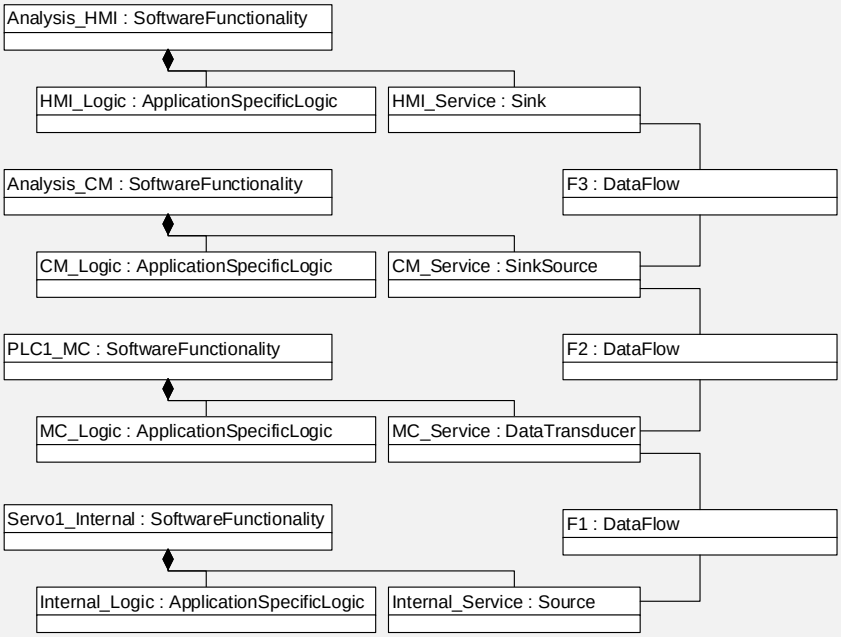
Starting from the field level, the internal control logic of the servo drive is instantiated as an instance of *SoftwareFunctionality* with the name *Servo1_Internal*. This software functionality aggregates the internal logic of the servo drive (*Internal_Logic*) and the communication part (*Internal_Service*). From an analysis point of view, the servo drive only sends data (the actual speed, *SpeedActual*, and torque, *TorqueActual*, values) but does not receive data. Hence, the *IService* instance *Internal_Service* is of type *Source*.

This data is sent to the communication part (*IService*) of the *SoftwareFunctionality* running on the PLC (*PLC1_MC*). Therefore, an instance of *DataFlow* (*F1*) is associated with both *IServices*. The software component *PLC1_MC* needs to forward the data from the servo drive and add the other signals from the bus (*WaterLevel*, *LightBarrier1*, and *LightBarrier2*) as well as the internal variable from the PLC logic (*SpeedSet*). As data is flowing into and out of the software functionality, its *IService* must be of type *IConsumerProducer*. Moreover, the original data from the servo drive is entirely forwarded, which specifies the *IService* as a *DataTransducer*.

In contrast, the analysis function for condition monitoring (*Analysis*_CM) inside the connected computer should consume the raw data from the PLC but not forward it any longer to the dashboard. Instead, only the calculated anomaly score is sent. Consequently, while the *IService* of the analysis software is still an *IConsumerProducer*, the concrete realization is a *SinkSource* that does not forward the original data any longer.

Finally, the anomaly score must be received by the dashboard software functionality (*Analysis_HMI*). The *ApplicationSpecificLogic* of the functionality (*HMI_Logic*) has to display the data to the user, while the communication part (*HMI_Service*) is a *Sink* for the data flow as no data is sent from here.

AE.Figure IV reflects the modeled instance. It captures the *SoftwareFunctionalities* and the *DataFlows* as part of the *SoftwareContainer* (not shown in the Figure).



AE.Figure IV: Example of SoftwareFunctionality and DataFlow modeling.

Across the connected systems, several types of data need to be processed and communicated by a *SoftwareFunctionality*. The abstract class *DataElement* describes these (see Figure 25). *DataElements* are differentiated by their type of information (*PrimitiveDataElement* or *ComplexDataElement*):

- *HardwareDataElement*, derived from *PrimitiveDataElement*, describes measured values that can be referred to as a measured hardware signal from a sensor or actuator (see Subsections HardwareContainer and RelationContainer). Examples are digital values from light barriers;

- *SoftwareDataElement*, derived from *PrimitiveDataElement and DerivedDataElement*, represents data that is calculated by a software functionality (pure software information). It may be based on other *DataElements* (e.g., *Hardware*) but does not correspond to the measured variable directly: instead, it can refer to original *DataElements* over the reference inherited by *DerivedDataElement*. Typical examples are values calculated in a PLC based on sensor values (*HardwareDataElement*) such as temperatures that are measured over resistance or operating modes;

- *ModelDataElement*, derived from *ComplexDataElement* and *DerivedDataElement*, describes complex trained or parametrized models for analysis and computation. The inheritance from *DerivedDataElement* allows to refer *DataElements* that were used to train the model; and

- *CompositeDataElement, derived* from *ComplexDataElement*, describes tuples of other *DataElements*, for instance, multi-dimensional data or structures.

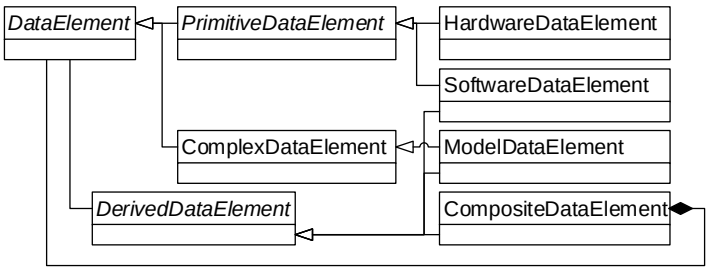
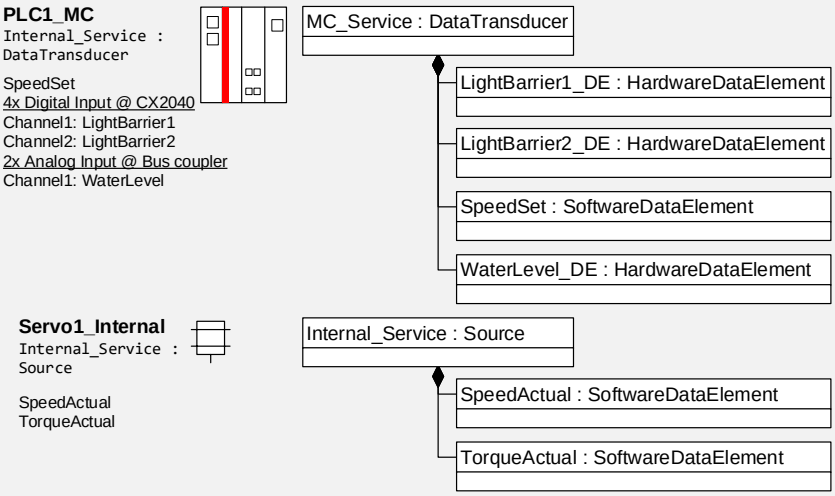


Figure 25: Detail of the metamodel for the description of DataElements.

DataElements and their way through the system need to be traced. Therefore, *DataElements* are aggregated by their original producers and referenced by the *DataFlows* that transport the specific *DataElement* (cf. Figure 24). As data is used to calculate and derive other data or information from it, the abstract class *DerivedDataElement* reflects this direct dependency of calculated data inside the architecture through a reference back to the original *DataElements*. Only *SoftwareDataElements* and *ModelDataElements* can include this reference (inheritance from *DerivedDataElement*). Therefore *HardwareDataElements* must always reflect raw and unaltered data from the field level. *DataElements* can change their name throughout their way through the systems. Therefore, the *DataTransportRelation* relates the unique *DataElements* to the transported *DataElements* and their system-specific names. AE.Part 3 gives a usage example of *DataElements* for modeling.

AE.Part 3: Modeling of DataElements.

In addition to the software functionalities, the data elements should be modeled as part of the *SoftwareContainer*. DataElements are aggregated by the respective *IServices* where they are first transmitted. As an example, the model of *DataElements* of the servo drive and the PLC is given in AE.Figure V. The sensor values *LightBarrier1*, *LightBarrier2*, and *WaterLevel* are of type *HardwareDataElement* as they are directly measured. In contrast, the *SpeedSet* variable is a *SoftwareDataElement* as it reflects an internal variable from the PLC logic without a direct correspondence to an output. The same applies to the servo drive's variables, which reflect information calculated from other data inside the servo drive control logic. The mapping of the *DataElements* to the respective *DataFlows F1* to *F3* from AE.Part 2 will be shown as part of the *RelationContainer* in AE.Part 6.



AE.Figure V: Example of DataElement modeling.

The platform-specific part of the *SoftwareContainer* (cf. Figure 24) describes and adds the concrete technologies and roles for a realization of the configuration. The elements *IProduceService* and *IConsumeServices*, derived from *IConnectionService*, refer to their abstract representations in the platform-interdependent part and enhance the modeled level of detail. The same applies to *ServiceDataFlow*, which details a *DataFlow* and connects the *IProduceServices* and *IConsumeServices*. *ApplicationSpecificImplementation* corresponds to the concrete realization of the logic defined in *ApplicationSpecificLogic*. *IConnectionServices* and the *UserImplementation* form so-called *SoftwarePackages*, the platform-specific counterpart to *SoftwareFunctionalities*.

In conjunction with the *RelationContainer*, the *SoftwareContainer* aims at addressing the data flow viewpoint of the modeling language (*Req-*M_{DF}).

Physical Container

The *PhysicalContainer* of the metamodel collects the descriptions of the hardware and network elements of the system architecture. In conjunction with the software functionalities mentioned above and a mapping between the two containers (see Subsection RelationContainer), it addresses the system viewpoint (*Req-M$_{Sys}$*). The *PhysicalContainer* was published by the author as part of the DSL4hDNCS [Vog+20].

Figure 26 reflects the elements of the *PhysicalContainer* with the base elements *IPhysicalConfigurationElement*, *IHardwareCapability*, and *IHardwareComponent*, which are characterized in the following:

- *IHardwareCapability* addresses the capabilities that specific hardware elements offer, such as converting electrical signals to data (*IConvertable*), being connectable to a network (*IConnectable*), and allowing the execution of higher software functionalities with application-specific code (*IProcessable*). Basic signal conversion and networking logic does not require an *IProcessable* (e.g., bus couplers with internal firmware, but no possibility of execution of custom logic);

- *IHardwareComponent* describes the elementary building blocks of hardware systems, in this case, *CPUs*, *NetworkInterfaces*, and *IOTerminals*; and

- *IPhysicalConfigurationElement* groups the separate hardware elements to physical systems. Furthermore, it defines the rules of their compositions. This includes (bus-)*Couplers*, *PLCs*, *Computers*, and *Clouds*, which are constituted of the elementary hardware components.

NetworkInterfaces aggregate elements of type *INetworkConfiguration* (not shown in Figure 26), which describe the actual configuration of an interface, including types of networks (e.g., Profibus, EtherCAT, or Ethernet) and the role in the network (master, slave, and regular participant).

IOTerminals aggregate *IOSignals* (in- and outputs, I/Os) from the field level. These signals are differentiated by their type of signal, namely digital information (*IOSignalDigital*) or analog information (*IOSignalAnalogue*). Additionally, sensors that serve as inputs (*IOTypeSensor*) are distinguished from the outputs of a control system, the actuators (*IOTypeActuator*). The particular types of *IOSignals* derive from these abstract superclasses, for instance, a *DigitalSensor* as an *IOSignalDigital* and *IOTypeSensor*. AE.Part 4 reflects the usage of the introduced elements to model the physical structure of the application example. The *IOSignals* can be mapped to *HardwareDataElement* as part of the *RelationContainer* (cf. Section RelationContainer).

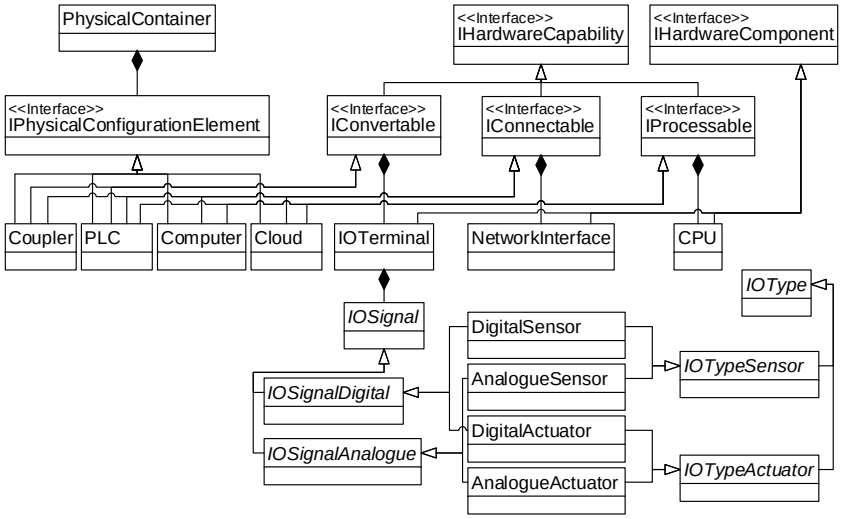


Figure 26: Detail of the metamodel's PhysicalContainer for a description of the system. Physical systems (left) are composed of distinct components (right). IOTerminals may encompass signals (bottom).

AE.Part 4: Modeling the physical configuration of the system (PhysicalContainer).

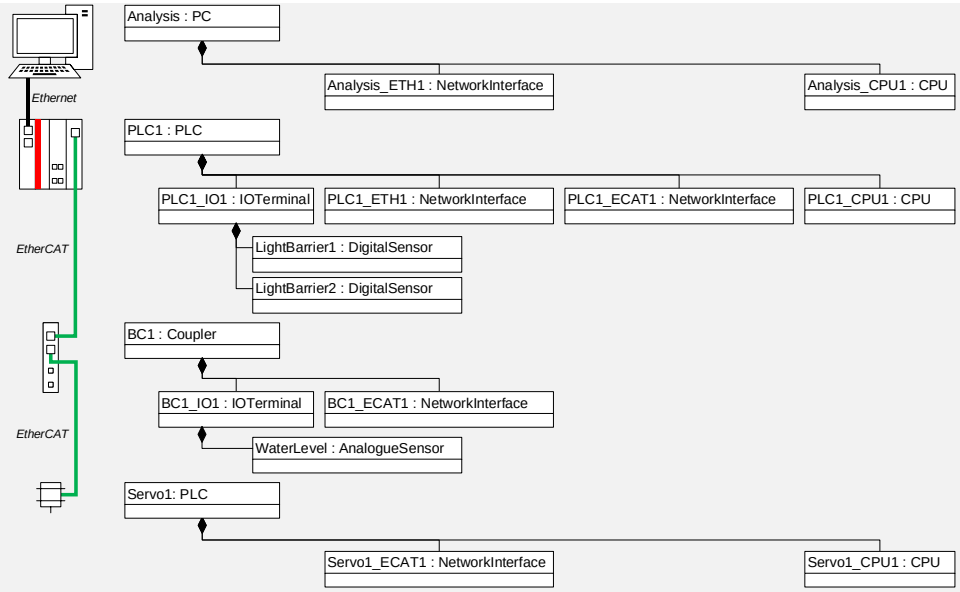
The physical configuration of an architecture is modeled as part of the *PhysicalContainer*. This container aggregates the separate hardware systems that form the architecture. For the application example, these are the servo drive with its internal logic, the EtherCAT bus coupler, the PLC, and the PC that hosts the analysis and dashboard functionalities. Each of these systems belongs to a specific category of *IPhysicalConfigurationElement* and is constituted of one or several *IHardwareComponents* (see AE.Figure VI).

For instance, the analysis computer (named A*nalysis* in this example) is a *PC* and aggregates a *NetworkInterface* (Analysis_ETH1) for network connectivity and a central processing unit (*CPU*) for the execution of *SoftwareFunctionalities*. The Beckhoff CX2040 *PLC* is composed of a *CPU* for the execution of the control program, two *NetworkInterfaces*, and an *IOTerminal*. While the first *NetworkInterface* is connected to the local Ethernet, the second *NetworkInterface* is the bus master interface of the EtherCAT bus of the conveyor. The *IOTerminal* corresponds to the four-channel digital input terminal directly attached to the *PLC*. It aggregates the two connected light barriers, which are both *DigitalSensors* (base type *IOSignal*).

In contrast to the *PLC*, the bus coupler *BC1* of type *Coupler* lacks an own *CPU* and is therefore not able to execute any *SoftwareFunctionality*. It can be regarded as a passive component. Its data has to be read from another active system with a *CPU*. Still, it contains an *IOTerminal* with the analog *WaterLevel* sensor attached and a *NetworkInterface* for EtherCAT connectivity.

Servo1, which is directly connected to the EtherCAT network, is represented as a *PLC* with a *NetworkInterface* and a *CPU* for the execution of the internal control logic.

The model instance to describe the physical system is shown in AE.Figure VI. The mapping of *IOSignals* to the *DataElements* is shown as part of the *RelationContainer* in AE.Part 6.



AE.Figure VI: Example of modeling the physical configuration.

AnnotationContainer

The *AnnotationContainer* holds information on annotations of the model elements with particular properties and requirements (*Req-M*$_{PropReq}$). The general structure of the AnnotationContainer was published by the author as part of the DSL4hDNCS [Vog+20] but in contrast to this thesis, including annotations for safety-related properties and requirements. Annotations can be assigned to distinct elements of the metamodel. The structure inside the *AnnotationContainer* is depicted in Figure 27. Annotations are grouped into so-called *AnnotationGroups*, which can describe several aspects of another element. Annotations can have the types (*AnnotationType*)

- *Requirement* to describe requirements a distinct system must fulfill; and
- *Property* that describes the actual value inside a deployed or simulated architecture.

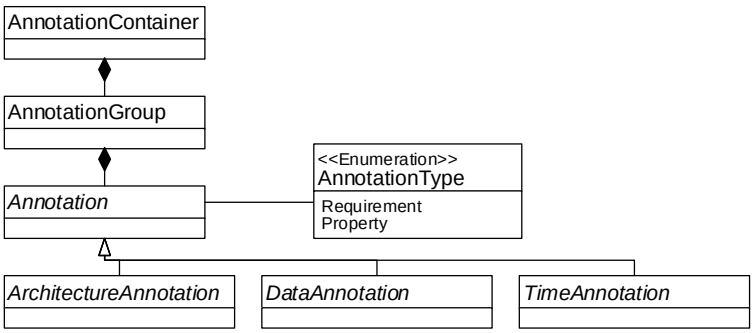


Figure 27: Detail of the metamodel's AnnotationContainer for description and categorization of annotations.

After deployment, *Requirements* and *Properties* can be compared to judge on the fulfillment of formulated requirements. Alternatively, if simulation models are available, feasible deployment scenarios can be simulated and assessed for requirement fulfillment. *Annotations* can be grouped into distinct categories. Following the original approach by Vogel-Heuser et al. [Vog+11], the following categories are used:

- architecture: annotations related to software or hardware systems;
- data: annotations related to the data that is communicated between participants; and
- time: annotations related to the time-behavior of systems. Directly based on the original approach [Vog+11], but extended by additional possible annotations.

A list of annotations included in the metamodel and the mappings to the respective categories are given in Table 7. Users may declare additional annotations and add them to the metamodel if needed for a use-case.

Table 7: List of annotations contained in the metamodel. Categorization (A) Architecture, (D) Data, (T) Time.

Type	Name	Description
A	ADDRESS	Address of a system. For instance, the IP address of an Ethernet interface or the Profibus station address of a bus coupler.
A	FLOW_TYPE	Specification on the type of a specific software functionality, e.g., stream, batch, or hybrid analysis/database.
A	HW_MANUF	Manufacturer specification of a hardware component, e.g., Siemens.
A	HW_TYPE	Type specification of a hardware component, e.g., S7-1513-1 PN.
A	HW_VER	Version specification of a hardware component.
A	N_SAMPLES	The ability of a system to buffer or store n samples.
A	REDUNDANCY	Information on redundancy/duplication of systems in order to improve reliability.
A	SCALABILITY	Represents the number of similar configurations connected to the same network, while only giving one example.
A	SW_NAME	Product name of a specific software representing a *SoftwareFunctionality*.
A	SW_PROVIDER	Provider of a specific software representing a *SoftwareFunctionality*.
A	SW_VER	Product version of a specific software representing a *SoftwareFunctionality*.
A	VLAN	VLAN identifier giving the VLAN (Virtual Local Area Network, IEEE 802.1Q [IEEE802]) an Ethernet network interface belongs to.
D	AUTH	The authentication mechanism for establishing communication or data transfer, e.g., password-based or certificate-based.
D	ENCRYPT	The encryption used for securing a data transfer, e.g., AES (Advanced Encryption Standard).

Type	Name	Description
D	PORT	Port used for communication as a combination of transport protocol and port, for instance, TCP:1883 as the standard port for MQTT.
D	PREPROCESS	Distributed preprocessing actions on involved systems, e.g., averaging or resampling.
D	PRIVACY	Privacy level of the transmitted data. This includes, for instance, normalization, resampling, or the introduction of arbitrary noise.
D	PROTOCOL	The underlying communication protocol used for communication.
D	SEMANTIC	Description of the underlying data semantic during transmission.
T	CYCLETIME	Cycle time of a system. Often used for Machine Control (MC) functionalities for cyclic execution of the control code.
T	JITTER	Information on jitter σ_f^2 for data transmission from source to destination.
T	LATENCY	Latency t_L description for data transmission from source to destination or data processing inside a system.
T	PROCESS	Time for processing t_{Proc} inside a system, for instance, analysis or translation of semantics.
T	SAMPLE_RATE	Sample rate f_S of a component to scan data.
T	SAMPLE_TIME	Sample time t_S of a component to scan data.

The mapping between annotations and other model elements is realized using mappers and inheritance of the mappers to the respective model elements. The mapper concept allows easy extension of additional dependencies and decreases the number of individual relations in the metamodel. An excerpt from the association logic is illustrated in Figure 28 for the *Annotations VLAN*, *Address*, *FlowType*, *Jitter*, and *Latency*. For instance, *Jitter* is the only *Annotation* that can refer to *Annotations*, in this case, other *TimeAnnotations*. The reason is that all other *TimeAnnotations* (e.g., *Cycletime* or *Latency*) can carry jitter information with them.

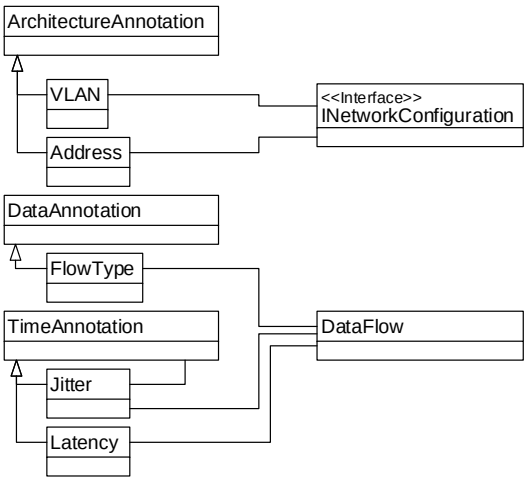
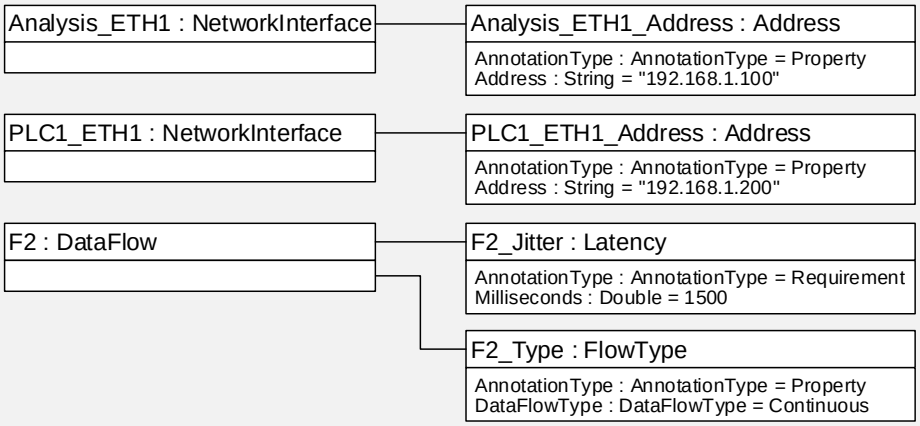


Figure 28: Excerpt of the metamodel for annotations.

An example of the usage of the annotations from Figure 28 is depicted in AE.Part 5.

AE.Part 5: Modeling of Annotations (AnnotationContainer).

As an example, the introduced types of annotations are used for adding additional information to the model (see the instance of the model in AE.Figure VII). For instance, the addresses of network interfaces can be specified. For the example, the IP addresses of the analysis computer, as well as the PLC, shall be specified as properties. Also, *DataFlow F2* (between PLC and analysis PC) has an associated latency requirement of 1500 ms, and its flow type is specified as continuous.



AE.Figure VII: Example of Annotations modeling.

RelationContainer

The *RelationContainer* includes the description of the mapping between software information and hardware platforms. It links the elements from the other containers and relates the modeled information. For instance, it states which software is running on which hardware device and what timing requirements have to be fulfilled. Every element which is aggregated by the *RelationContainer* is derived from the interface *IRelationContainer* (see Figure 29):

- *NetworkRelation* references *NetworkInterfaces* that are part of the same physical network and can communicate directly;
- *NetworkBindingRelation* maps a *DataFlow* to a concrete *NetworkInterface* and therefore describes the actual network that is used for communication;
- *HardwareSoftwareRelation* describes which processing unit (*CPU*) is associated with specific software (*SoftwareFunctionality*) and serves as an execution environment;
- *IOSignalRelation* which relates an *IOSignal* measured by an *IOTerminal* to its representation as a transferable data element (*HardwareDataElement*);
- *DataFlow* (shown in Figure 24) as a relation between *IProducer and IConsumer*; and

- *DataTransportRelation* from Figure 24, which maps the unique *DataElements* to the respective *DataFlows*.

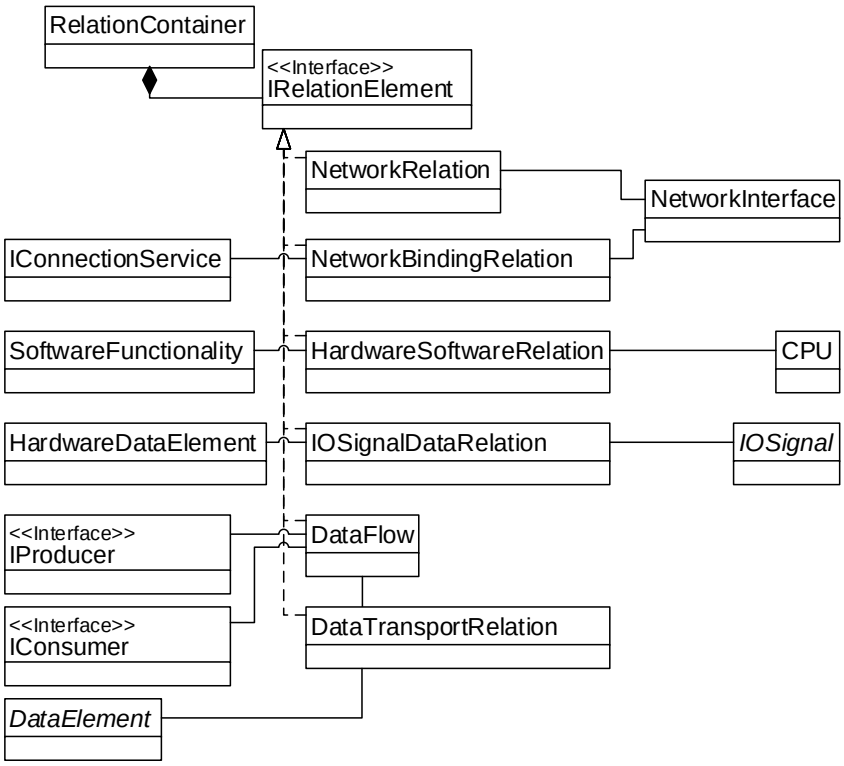


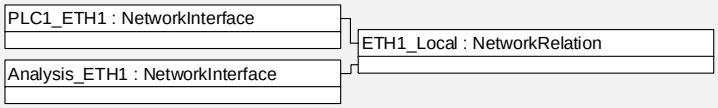
Figure 29: Detail of the metamodel for mapping software (left) and system (right) description with IRelationElements.

AE.Part 6 reflects the usage of *IRelationElements* to model relations between the elements of the metamodel.

AE.Part 6: Modeling relations between the elements and containers.

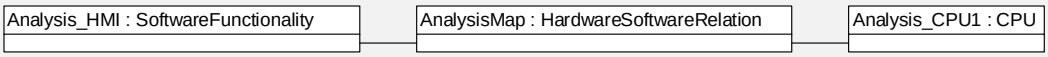
This part of the application example captures samples of the usage of *IRelationElements* and reflects the intended usage of these elements. However, full modeling of relations is beyond the scope of this application example and too exhaustive.

Networks are described as *NetworkRelations* that connect the related *NetworkInterfaces*. As an example (see AE.Figure VIII), the model of the Ethernet network (called *ETH1_Local* in this example) connecting the *NetworkInterfaces* of the PLC (*PLC1_ETH1*) and the analysis computer (*Analysis_ETH1*) is given below.



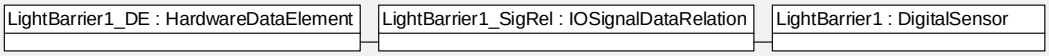
AE.Figure VIII: Example of modeling a network.

Via *HardwareSoftwareRelations* (see AE.Figure IX), the execution platform of a *SoftwareFunctionality* can be specified. As part of the example, the dashboard functionality (*Analysis_HMI*) is hosted on the analysis computer, more specifically, its *CPU* (*Analysis_CPU1*):



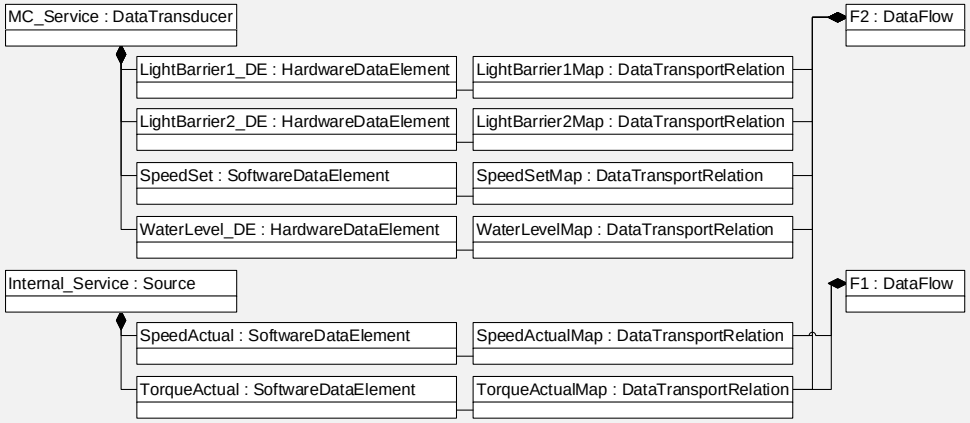
AE.Figure IX: Example of mapping SoftwareFunctionalities to CPUs.

The metamodel differentiates between the digitized information from in-/outputs (*HardwareDataElement*) and the sensor/actuators (*IOSignal*). Therefore, they need to be mapped to their software representations using *IOSignalRelations* (see AE.Figure X). For instance, the *LightBarrier1* as a *DigitalSensor* is mapped to the corresponding *HardwareDataElement* (*LightBarrier1_DE*):



AE.Figure X: Example of mapping HardwareDataElements to corresponding IOSignals.

Transport of a specific *DataElement* as part of a *DataFlow* is modeled with a *DataTransportRelation* (see AE.Figure XI). The corresponding *DataFlows* aggregate these. The example below contains an excerpt, where the *DataElements* of *Internal_Service* are transported as part of *DataFlow F1*. Additionally, *F2* transports the same data and includes data from *MC_Service*.



AE.Figure XI: Example of associating DataElements to DataFlows.

5.2.2. Graphical Modeling Notation

This section describes the graphical notation that builds on top of the metamodel and visualizes the modeled information. As stated by requirements *Req-M_{Sys} and Req-$M_{DF,}$* it distinguishes between the viewpoints *system* and *data flow*. Annotations that describe properties and requirements (*Req-$M_{PropReq}$*) can be used in both viewpoints and will be introduced after a presentation of the system and data flow viewpoints. Parts of this chapter have been published as [TWV20].

The system viewpoint is based on the graphical notation presented by Vogel-Heuser et al. [Vog+11]. The original approach and its modeling capabilities are extended by

- software functionalities,
- additional types of signals,
- a unique labeling system for identification of systems and other elements, and
- supplementary symbols.

The unique labeling system is a necessity for mapping the additional viewpoints of the notation and is therefore not part of the original approach. Throughout the following Section, the following letters indicate the relation of graphical model elements and concept to the original source of Vogel-Heuser et al. [Vog+11]:

- (I) included in the original source and used as is,
- (A) adapted and extended from the original source (modifications are mentioned).

If not mentioned differently, the graphical model elements are newly introduced as part of this contribution.

Table 8 summarizes generic graphical symbols that are consistent over both viewpoints. It includes the elements for the definition of *DataElements* and *IOSignals*, the graphical symbols used for the indication of *IOType* and *IOSignal*, as well as the drawing frame, which limits the drawing area of the graphical models.

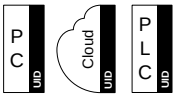
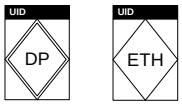
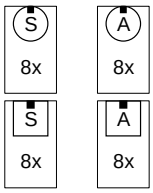
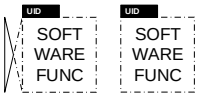
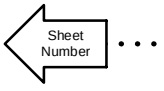
Table 8: Generic notation elements for both viewpoints of the graphical modeling notation.

Symbol(s)	Description
S UID	Signal or information element (*DataElement* and *IOSignal*) that is related to a system or data flow. The left field indicates the type and form of data (see below); the right side gives the UID of the variable or signal for identification across all systems. It reflects the available signals/information in a system or data flow. Mapping tables (see Table 13) give the mapping of the UID to the system-specific naming of this signal/information. (A) with a UID label.
 	Indicator for the type of related signal/information. Shape indicates the form of information (*IOSignal*, (I)): • circle analog signal, and • square digital signal. Tag indicates type of information (*IOType* and *DataElement*), • S sensor (*IOTypeSensor*, *HardwareDataElement*, (I)), • A actuator (*IOTypeActuator*, *HardwareDataElement*, (I)), • V variable (calculated, *SoftwareDataElement*), • C composite (*CompositeDataElement*), and • M model (parameterized or trained, *ModelDataElement*).
	Reference arrow (I) that connects the *DataElements*/*IOSignals* to the related software (then a *DataElement*) and hardware (then an *IOSignal*) system.
SheetName.SheetNumber	Drawing frame of a drawing sheet, as well as name and number of the sheet. Diagrams can span multiple sheets. Every sheet needs a unique combination of *SheetName* and *SheetNumber*.

System Viewpoint

The system viewpoint includes graphical items for elements from the *PhysicalContainer* as well as *DataElements* and *SoftwareFunctionalities* from the *SoftwareContainer*. Table 9 lists and describes these symbols. The convention for the system viewpoint is, if graphically possible, a hierarchical layout with superordinate systems at the top of the drawing sheet and the field level at the bottom. Network lines run horizontally from left to right with vertical connection lines to the associated *NetworkInterfaces*.

Table 9: Notation elements for the system viewpoint of the graphical modeling notation.

Symbol(s)	Description
	A processing unit (left part of each symbol, *IProcessable*) and a unique identifier (UID, (A)) of a system (right side, rotated). Processing units enable the execution of software functionalities. Differentiation of *Computers* (PC, left, (I)), cloud environments (*Cloud*, middle), PLCs/industrial PCs (*PLC*, right, (I)). *PLC* enables the combination with field terminals (*IConvertable*). If present, the first element of a system on the left.
	Bus coupler unit (*Coupler*) visualized by UID label. It does not contain a processing unit and can, therefore, not host any software functionality. The addition of a *NetworkInterface* to the right of the element is mandatory. Field terminals can be connected to the right of the bus coupler (*IConvertable*). (A), as the original approach implicitly models bus couplers as network interfaces without a processing unit.
	Communication interfaces of a system with UID of the interface. Differentiation of master interfaces for master/slave field buses (left) and slave/non-master fieldbuses/networks (right). Label inside rhombus indicates the type of communication interface, for instance, ETH (Ethernet), ECAT (EtherCAT), DP (Profibus DP), PA (Profibus PA), PN (Profinet), or CAN. (A) UID labels.
	Field terminals (*IOTerminals*, (I)) for in-/output of signals (*IOSignals*). The number below specifies the number of I/O channels that can be connected via the terminal. Field terminals follow a processing unit or a communication interface. Typically connected with signal elements (*IOSignal*).
	*SoftwareFunctionality* that is executed on a hardware system with UID of the specific functionality. *SoftwareFunctionalities* can only be executed if the related hardware system contains a processing unit (*IProcessable*). The first software functionality of a system is connected to the hardware with two triangles (left, *NetworkBindingRelation* and *HardwareSoftwareRelation*), additional functionalities are added on the right side of existing functionality (right). A concrete *SoftwareFunctionality* replaces the placeholder SOFTWAREFUNC (see Table 10 for a list of defined labels and their associated description).
	*Network* (bold, *NetworkRelation*) and connection lines to *NetworkInterfaces* (thin) with identifying UID label on the network. Connected interfaces determine the type of network. (A) UID label.
	Off-page connector for networks spanning multiple drawing sheets. The direction is always outwards from the connected network. Networks spanning multiple sheets need a consistent UID label on every sheet. Label SheetNumber gives the number of the sheet, where the continuation of the network is found.

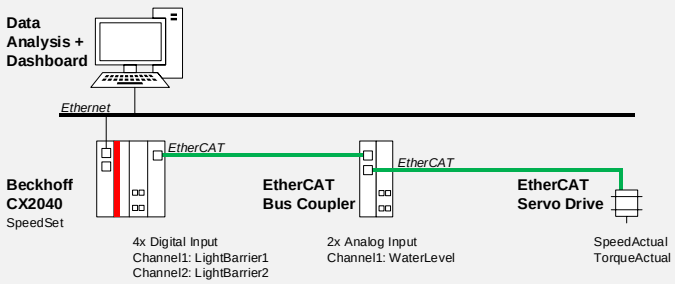
As mentioned in Table 9, *SoftwareFunctionalities* are graphically depicted by a special symbol and an associated label. This label describes the function that is executed. Labels and their explanations are summarized in Table 10. Finally, AE.Part 7 reflects the application of the graphical elements from the system viewpoint to the application example.

Table 10: Non-exhaustive list of possible software functionalities.

Functionality	Description
AGGR	Transparent aggregation of data from various sources without changes in protocol, format, and semantic.
DA	Data analysis functionality for extracting information and knowledge from data. May calculate variables and models.
FORW	Software functionality to transparently forward data to another system without modifications in format and semantic.
LEG	Existing legacy software components with an internal logic that may generate, consume, or manipulate data. Examples are MES and ERP systems, as well as other proprietary systems. If a legacy component can be decomposed into other software functionalities, these may be used instead of the LEG label.
MC	Machine control, typically a control application, running on a PLC or PC. May calculate variables from measurement signals.
ROUT	Message routing functionality to enable communication between heterogeneous systems. Typically, a middleware component.
STOR	Storage functionality to buffer or store, as well as providing data, information, and models.
TRANS	Translation between different data protocols, formats, and semantics. Used to adapt incompatible and legacy systems.
VISU	Visualization of data for users (human-machine interface, dashboards).

AE.Part 7: Graphical model in system viewpoint.

The system viewpoint follows the schematic view given in AE.Part 1 (see AE.Figure XII).



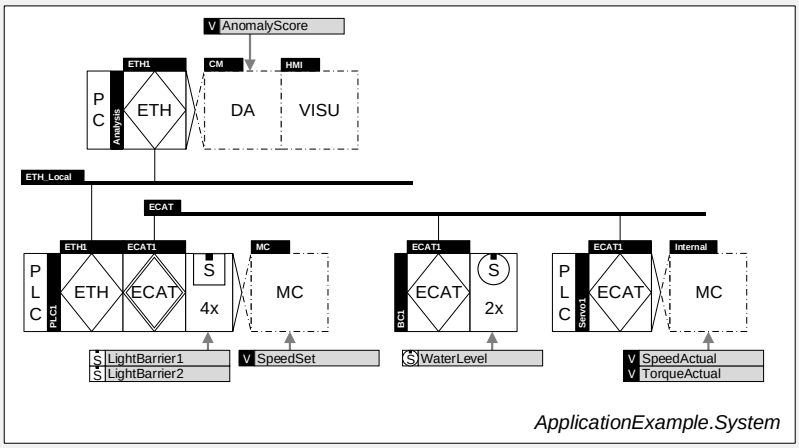
AE.Figure XII: Schematic drawing of the physical setup.

Starting from the field level, the servo drive (see lower right part ofAE.Figure XIII) is modeled as a *PLC* with the name *Servo1*, one EtherCAT slave interface (name *ECAT1*), and a *SoftwareFunctionality* that reflects the internal control code (*MC*, name *Internal*). The two corresponding internal variables (*SoftwareDataElements*) are associated with the *MC SoftwareFunctionality* where they are calculated.

This servo drive is connected to the EtherCAT field bus *ECAT*. The master interface of this bus is part of the central Beckhoff PLC (name *PLC1*). The PLC furthermore has a second network interface for Ethernet connectivity (ETH1), the four-channel digital input module with the two light barriers connected, and the machine control *SoftwareFunctionality* (name *MC*) with the software variable *SpeedSet*. The bus coupler *BC1* is part of the *ECAT* EtherCAT field bus as well and has a two-channel analog input signal connected with the *WaterLevel* sensor.

As already mentioned, *PLC1* is part of a second network (*ETH_Local*) that connects it to the analysis *PC* (name *Analysis*). This computer hosts the two *SoftwareFunctionalities* for data analysis (*DA*, name *CM*), where it calculates the *AnomalyScore* as a software variable, and the dashboard (*VISU*, name *HMI*).

The graphical model is completed by a drawing frame and the unique ID of this drawing sheet (*ApplicationExample.System*). Please refer back to AE.Parts 2, 3, and 4 for the corresponding instances of the metamodel.

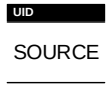
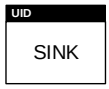
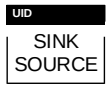
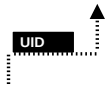


AE.Figure XIII: Example of the application example in the system viewpoint.

Data Flow Viewpoint

The data flow viewpoint is inspired by data flow diagrams (DFDs) introduced by DeMarco [DeM79] in his specification of the structured analysis (SA) and also used by Hatley and Pirbhai in their SA/RT for real-time systems [HP88]. The graphical notation adapts the method and terminology of modeling data flow diagrams and extends it with additional symbols for the specific application. The nodes of the DFD are mapped to *SoftwareFunctionalities*. Their shape can distinguish the influence on the DataFlow. Table 11 summarizes the notation elements for the data flow viewpoint and describes their meanings. The concrete function of the *SoftwareFunctionality* replaces the labels inside the elements (see Table 10). The convention for drawing data flow diagrams is a vertical flow from the bottom of a drawing sheet to the top. This layout reflects the hierarchical flow of data from field levels systems to superordinate IT systems, as well as the orientation of the system viewpoint.

Table 11: Notation elements for the data flow viewpoint of the graphical modeling notation.

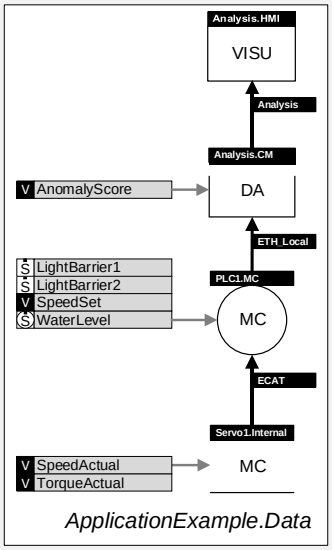
Symbol(s)	Description
	Component serves as the *Source* of a new *DataFlow*. A *Source* can receive no data, data is only flowing out. The element refers to a concrete *SoftwareFunctionality* from the system viewpoint hosted on an *IPhysicalConfigurationContainer*. The naming scheme for UID is *IPhysicalConfigurationContainer*.UID of *SoftwareFunctionality*.
	Component serves as the end (*Sink*) of a *DataFlow*. No data can be forwarded from a *Sink*. Data is only flowing in. The element refers to a concrete *SoftwareFunctionality* from the system viewpoint hosted on an *IPhysicalConfigurationContainer*. The naming scheme for UID is *IPhysicalConfigurationContainer*.UID of *SoftwareFunctionality*.
	Component serves as a transparent *DataTransducer*. All data flowing into the transducer is also available on the output side of the component but can be buffered by the software functionality. Transducer blocks may also calculate additional *DataElements*, which have to be specified individually. The element refers to a concrete *SoftwareFunctionality* from the system viewpoint hosted on an *IPhysicalConfigurationContainer*. The naming scheme for UID is *IPhysicalConfigurationContainer*.UID of *SoftwareFunctionality*.
	Component serves as a non-transparent data *SinkSource*. Data that flows into the component is not automatically available on the output side of the component. The *DataElements* that should be available on the output side need to be specified explicitly. *SinkSources* may alter or buffer data. *SinkSource* blocks may also calculate additional *DataElements* which have to be specified individually. The element refers to a concrete *SoftwareFunctionality* from the system viewpoint hosted on an *IPhysicalConfigurationContainer*. The naming scheme for UID is *IPhysicalConfigurationContainer*.UID of *SoftwareFunctionality*.
	Specification of a *DataFlow* from one component to another (*IService*) in the form of a continuous stream of data. Continuous streams are characterized by a cyclic exchange of often small data packages. UID refers to a *NetworkRelation* if data flows over a network, or to an *IPhysicalConfigurationContainer* if data flows between two *SoftwareFunctionalities* on the same hardware system (inter-process communication, IPC).
	Specification of a *DataFlow* from one component to another (*IService*) in the form of discrete batches of data. Batches of data are often generated by buffering a continuous stream of data in a database or buffer. Batched data often flows only sporadically and in large packages. UID refers to a *NetworkRelation* if data flows over a network, or to an *IPhysicalConfiguration-Container* if data flows between two *SoftwareFunctionalities* on the same hardware system.
	Indicates that the *DataFlow* is distributed over multiple drawing sheets. A *DataFlow* from a *SoftwareFunctionality* ends at this symbol and references to another sheet (SheetNumber). On the other sheet, the *DataFlow* starts again at the top of the symbol and ends at a *Software-Functionality* to form a *SoftwareFunctionality*-to-*SoftwareFunctionality* connection. Every sheet break needs a unique name (label UniqueName) for identification.

The application of the data flow viewpoint to the application example is given in AE.Part 8.

AE.Part 8: Graphical model in data flow viewpoint.

As mentioned in AE.Part 2, the servo drive is a *Source* of data from a data analysis point of view. Therefore, at the bottom of the graphical model (see AE.Figure XIV), the *MC* functionality of the servo drive (unique name *Servo1.Internal*) is depicted. The two software variables *SpeedActual* and *TorqueActual* are assigned to the *Source*. From here, data flows continuously over the *ECAT* EtherCAT bus to the *MC* functionality of *PLC1*. This component acts as a *DataTransducer*, transparently forwarding the ingoing data and adding more variables (the light barriers, the set speed, and the *WaterLevel)*. All data is then sent over *ETH_Local* to the analysis computer and its data analysis (*Analysis.CM*), where the continuous flow is ending (*SinkSource*), and a new variable (*AnomalyScore*) is calculated. Finally, this information is internally sent to the dashboard (*Analysis.HMI*) and displayed, where the overall flow of data ends.

The graphical model is completed by a drawing frame and a unique label (*ApplicationExample.Data*). Please also refer back to AE.Parts 2, 3, and 4 for the corresponding instances of the metamodel, as well as AE.Part 7 for the corresponding graphical model as part of the system viewpoint.



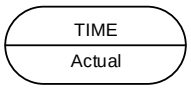
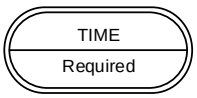
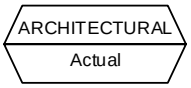
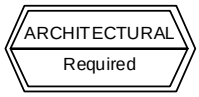
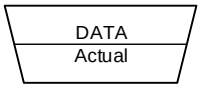
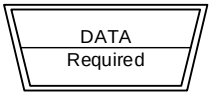
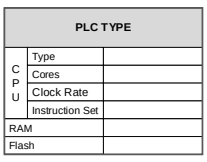
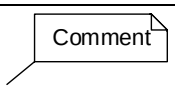
AE.Figure XIV: Example of the application example in the data flow viewpoint.

Annotations

Annotations allow the user to specify additional information to characterize the system. This information may be, for instance, a requirement that has to be fulfilled for the system to function correctly, e.g., a maximum acceptable latency of data transmission. The annotation elements for adding properties, requirements, and additional information to the graphical models are summarized in Table 12. Properties are depicted as single-bordered and requirements as double-bordered shapes. The notation differentiates between three types of properties/requirements based on the

shape of the annotations: time-related information, for instance, communication latency or sample rates, architectural information that defines types of data storage or scalability of components, and data flow-related information on protocols, semantics, or encryption. Different shapes can differentiate these. The graphical differentiation between requirements and properties, as well as the idea of categorization, follows the original approach [Vog+11]. A list of properties and requirements and their categorization was given in Table 7 in Subsection 5.2.1.

Table 12: Annotation elements for both viewpoints of the graphical modeling notation.

Symbol(s)	Description
	A time-related property (I) of a system or data flow. A property from Table 7 replaces TIME placeholder. Actual gives the actual value of the property.
	A time-related requirement (I) of a system or data flow. A requirement from Table 7 replaces TIME placeholder. Required gives the specified value of the requirement.
	An architecture-related property of a system or data flow. A property from Table 7 replaces ARCHITECTURAL placeholder. Actual gives the actual value of the property.
	An architecture-related requirement of a system or data flow. A requirement from Table 7 replaces ARCHITECTURAL placeholder. Required gives the specified value of the requirement.
	A data-related property of a system or data flow. A property from Table 7 replaces DATA placeholder. Actual gives the actual value of the property.
	A data-related requirement of a system or data flow. A requirement from Table 7 replaces DATA placeholder. Required gives the specified value of the requirement.
	Annotation line (I) for a system. It connects the annotation element with the related software functionality, network, or signal in the system viewpoint.
	Annotation line (I) for a data flow. It connects the annotation element with the related software functionality, network, or signal in the data flow viewpoint.
	Annotation line (I) for a property or requirement with a reference. It connects the referenced element to the property or requirement. The annotation element already needs to be connected to another element of the diagram with one of the annotation lines for systems or data flows. For instance, Latency requirements always are associated with an *IConsumer* and refer to an *IProducer* to reflect latency in communication between the two elements.
	Specification of the type of a *PLC* or *Computer* (PLCTYPE) and its important characteristics, including the characteristics of the central processing unit (CPU) (type, number of cores, clock rate, and supported instruction set), as well as available Random-Access-Memory (RAM) and flash memory. If characteristics are unknown or not specified, only PLCTYPE element without further information may be used. Double outer line indicates a requirement for a specific platform. Adapted from [Has+13].
	Non-formal comments to add information to a diagram.

The application of annotations for amending the graphical models with properties, requirements, and additional information is given in AE.Part 9.

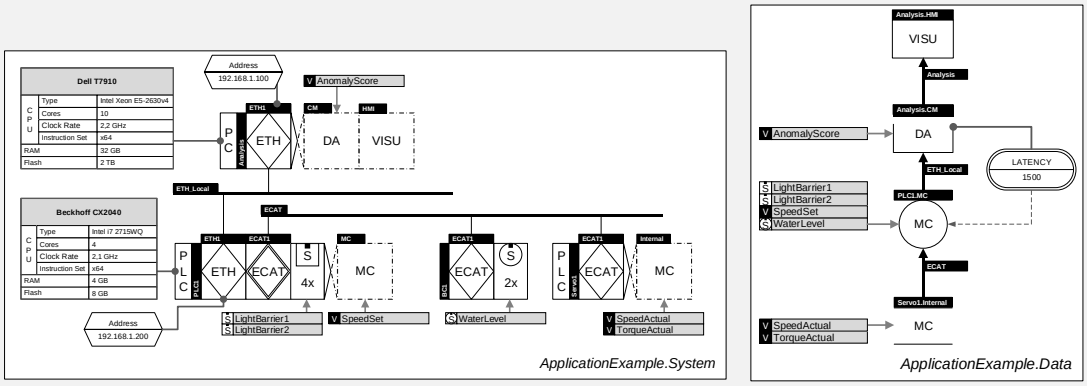
AE.Part 9: Annotated graphical models.

This part of the application example graphically amends the models (AE.Parts 7 and 8) with the information from AE.Part 5 (see AE.Figure XV below for the annotated graphical models, left system viewpoint, right data flow viewpoint).

The addresses of *Analysis.ETH1* and *PLC1.ETH1* can be modeled in the system viewpoint as architecture-related properties. Additionally, the hardware details of the CX2040 PLC, as well as the analysis computer (DELL T7910), are specified as supplementary information.

The data flow viewpoint already includes the information that *DataFlow F2* (between *PLC1.MC* and *Analysis.CM*) over *ETH_Local* is a continuous data stream (solid arrow, not dotted). Finally, the maximum allowed latency between *PLC1* and *Analysis* computer can be added as a time-related requirement (double-edged).

The Figures below reflect annotated graphical models and, therefore, still have the same labels.



AE.Figure XV: Annotated graphical models of the application example in both viewpoints.

Data Mapping Table

DataElements often have system-specific variable names. These change over their way through the system as part of a *DataFlow*. Therefore, the concept of data dictionaries [DeM79; HP88] is adapted for this approach as so-called data mapping tables. While the content of the data dictionary is altered in comparison to DeMarco's concept, its function remains: ensuring traceability of data throughout the system. The data mapping table correlates the system-specific UIDs of *DataElements* to generic UIDs that uniquely identify the element.

In contrast to the graphical elements presented previously, the data mapping table includes no graphical representation but serves as a dictionary to collect and structure additional information on *DataElements*. The columns, and therefore the contained information is summarized in Table 13. Additional columns may be added for specific use-cases if appropriate. [TWV20]

AE.Part 10 depicts the application of the mapping table to the application example.

Table 13: Columns of the mapping table and description of their meaning. Adapted from Trunzer et al. [TWV20].

Column Name	Description
VariableUID	Unique identifier of a *DataElement* across all systems. It corresponds to the *Name*-attribute of the SA/RT [HP88].
SystemUID	Unique identifier of the system the SystemSpecificVariableUID is valid for. Adapted *Member of*-attribute of the SA/RT [HP88].
SystemSpecificVariableUID	Unique identifier of a *DataElement* used in a specific system.
DerivedFromVariableUID	If data is based on other data (calculated, derived, composite, or used in the model), the original unique identifier of these *DataElements* (VariableUIDs) can be given here. Otherwise empty. It can be multiple separated by commas for composite *DataElements*.
Description	Optional description of a variable. It corresponds to the *Comments*-attribute of the SA/RT [HP88].
Address	Optional address inside the specific system, for instance, register numbers or addresses of associated bus couplers and terminal channels.
Type	Type of the variable, signal, or model, for instance, float, integer, boolean, or model.
Resolution	Measurement resolution, if available. Otherwise empty. It corresponds to the *Resolution*-attribute of the SA/RT [HP88].
Timeseries	Simplification for stating that a *DataElement* is always a tuple of actual time (timestamp) and value. No separate declaration of the *CompositeDataElement* using the *DerivedFromVariableUID* column is needed.

AE.Part 10: Data mapping table.

AE.Figure XVI reflects the usage of the data mapping table for the case of the application example. All variables are associated with a unique *VariableUID*. Variables can be referenced in multiple systems (e.g., *SpeedActual* in *Servo1.Internal* and *PLC1.MC*) and have individual names inside the systems (*SystemSpecificVariableUID*). Additionally, the mapping tables provide the possibility to amend data types (*FLOAT* for *SpeedActual*) and the measurement resolution (12 bit in this case). The last column states if the variable is always transmitted with an associated timestamp (if it is a time series) or if it resembles a value without time information. The column *DerivedFromVariableUID* refers to other variables that are used for calculating the regarded variable. In this example, all variables from the field level are used for the calculation of the *AnomalyScore* inside *Analysis.CM*.

VariableUID	SystemUID	SystemSpecificVariableUID	DerivedFromVariableUID	Type	Resolution	Timeseries
SpeedActual	Servo1.Internal			FLOAT	12 bit	yes
TorqueActual	Servo1.Internal			FLOAT	12 bit	yes
LightBarrier1	PLC1.MC	LB1		BOOL		yes
LightBarrier2	PLC1.MC	LB2		BOOL		yes
WaterLevel	PLC1.MC	Level		UINT32	8 bit	yes
SpeedSet	PLC1.MC	V_Set		FLOAT		yes
SpeedActual	PLC1.MC	V_Act		FLOAT		yes
TorqueActual	PLC1.MC	M_Act		FLOAT		yes
AnomalyScore	Analysis.CM	AnomalyScore	SpeedActual, TorqueActual, LightBarrier1, LightBarrier2, WaterLevel, SpeedSet	DOUBLE		yes

AE.Figure XVI: Excerpt of the data mapping table for the application example.

Mapping of the Viewpoints and the Mapping Table

With the help of the unique labeling system, the information from the different viewpoints and the mapping table can be related to each other. The principle is illustrated in Figure 30 for a basic scenario of the measurement of one variable (*PressureActual*, measured in *Machine1*), the derivation of an alarm message if the pressure is too high (*PresureExceeded*), and the transport of this information to an analyzer on another system (*Analyzer1*).

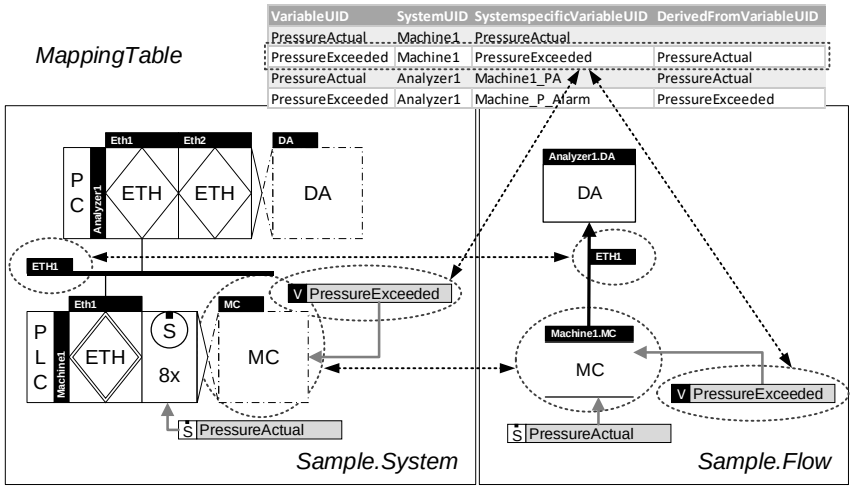


Figure 30: Basic example of the graphical notation illustrating the mapping between the different viewpoints (system viewpoint (left) and data flow viewpoint (right)) and the mapping table (top).

5.3. Architecture Software Framework

Communication architectures for data collection in industrial automation are commonly realized in an ad-hoc fashion to allow access to data quickly. This, however, may lead to a suboptimal selection of technologies. Also, high implementation efforts lead to vendor lock-in and prevent migration of deployed communication architectures to more suitable technologies.

A software framework with a unified programming interface (*Req-SF$_{API}$*) could potentially decrease implementation efforts through reusability and simplify the migration from one technology to another. As the available technologies and their programming interfaces are very heterogeneous, the technology-specific aspects of the distinct technologies must be abstracted to provide a common platform for the implementation of data collection architectures (*Req-SF$_{ACP}$*). Such abstraction allows application programmers to develop their software and rely on the functionality of the developed API without special considerations of the actual communication technology. If a change of communication technology becomes necessary at some point in time, only minimal changes to the code are necessary, which has the potential to simplify migrations in the future considerably. Furthermore, the definition of interfaces facilitates a modular software design and simplifies future extensions of the software framework's functionalities.

The software framework serves as a basis for practical realizations of data collection architectures in industrial automation. It can be used independently of the other described concepts but, at the same time, serves as a basis for the model-driven generation of the communication architecture, which is explained later. The presented software framework is a rewritten version for increased reusability and modularity based on previously published work ([Tru+19b]).

The definition of the standard interfaces and the core of the software framework are depicted in Figure 31. The software framework differentiates communication services as *IReceiveServices* to receive data and *ITransmitServies* to send data. Both interfaces inherit from the base interface *IMessagingService*, which contains generic definitions that every communication service must implement. *IReceiveService* serves as the superclass for the derived interfaces *IRequestService* (Receive-Response messaging pattern) and *ISubscribeService* (Publish-Subscribe messaging pattern). On the transmitting side, *IPublishService* inherits from *ITransmitService*. Figure 31 contains the two example technologies *TechAService* and *TechBService*, as placeholders for concrete realizations. Inheritance from *ISubscribeService*, *IReceiveService*, and *IRequestService* reflects the functionality that is implemented using specific communication technology. The services, therefore, implement the corresponding method signatures and map the generic functionalities to the

technology-specific functionality. For instance, the service for technology A (*TechAService*) implements Publish-Subscribe as well as Request-Response functionality. On the other hand, the service for technology B only provides Publish-Subscribe functionality.

The framework is designed to be applicable to a wide range of use-cases. Therefore, accepted software design patterns are employed to increase the reusability of code and to abstract implementation details. For instance, every communication service is created by an associated service factory that implements a standard interface (*IServiceFactory*). This so-called abstract factory design pattern [Gam11] reduces the dependency of application code on the concrete technologies and implementations. Clients depend on the functionality defined in the standard interfaces *IReceiveService*, *ITransmitService*, and *IServiceFactory* without consideration of the concrete implementations of technologies. This decoupling allows a simple exchange of communication technologies with minimal adjustments to the code by requesting the creation of a different communication service from the service factory. Consequently, the application-specific logic of clients is separated from the internals of communication and can remain almost unchanged.

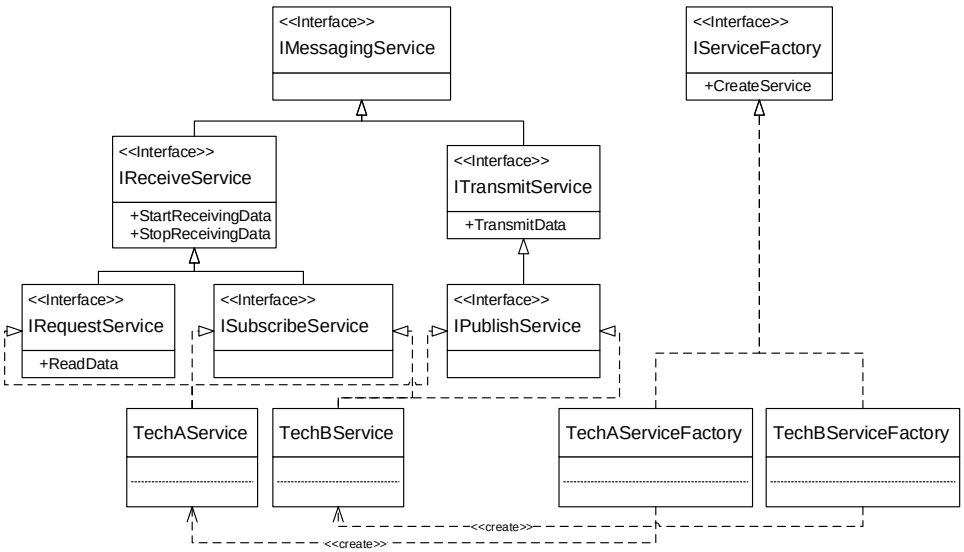


Figure 31: UML class diagram of the interface definitions for the core software framework. The left part reflects the definition of communication services, right part the service factory definition for dynamic creation of services. Usage of the interfaces shown for two example communication technologies TechA and TechB.

5.4. Automatic Generation of the Communication Architecture

The last building block of the concept is the automatic generation of the communication architecture, which is depicted in Figure 32. The starting point is a model of the data collection architecture on the MOF M1 layer. This model is an instance of the metamodel presented in Subsection 5.2.1 (MOF M2 layer). A code generation engine queries the modeled elements and executes model to

text transformation (M2T). These transformations rely on code templates from the software framework (Section 5.3). The code templates are combined in the transformation step to construct

- the communication parts of each modeled participant, including the receiving and publishing of data and
- configuration files for middlewares.

Therefore, the code templates contain placeholders that are filled by the code generation engine with the related information from the model, for instance, the concrete technology for communication or IP addresses. The code generation engine has a minimum set of rules to check the consistency of the modeled information that guarantees a deployable data collection architecture. Incomplete models, e.g., lacking a description of addresses or communication protocols or with no network connection between sender and receiver, lead to an error. These need to be resolved by the experts before repeating the step of code generation.

The result of the code generation step is preconfigured code for a communication architecture that reflects the modeled flow of information. It handles the sending and receiving of data over the configured data flows (OSI layer 7). Still, application-specific code that glues together the data flows inside the SoftwareFunctionalities is not automatically generated (also OSI layer 7). This includes, for instance, the translation from one information model to another, or the calculation of derived variables. Therefore, the automatic generation is no complete generation of the data collection architecture, but a partial generation for the communication parts of the overall architecture on OSI layer 7 [BCW17]. Experts insert the application-specific code into specially marked placeholders inside the generated code fragments. The application-specific code is embedded into so-called protected areas that are preserved when regenerating the architecture. After the addition of the application-specific logic, a ready-to-deploy data collection architecture prototype is the result. This prototype can now be compiled and deployed to the individual systems by the experts. The modeled information serves as a specification. In the end, the deployed data collection architecture is an instance of the architecture model on the MOF M0 layer.

AE.Part 11 demonstrates the code generation for the application example. Please note that all code is expressed as pseudo-code and greatly simplified to remain technology-neutral and to give an impression of the concept, not its real implementation using specific programming languages or communication protocols. Furthermore, it is assumed that the code generation can be used for all systems irrespective of the underlying platform and supported programming languages.

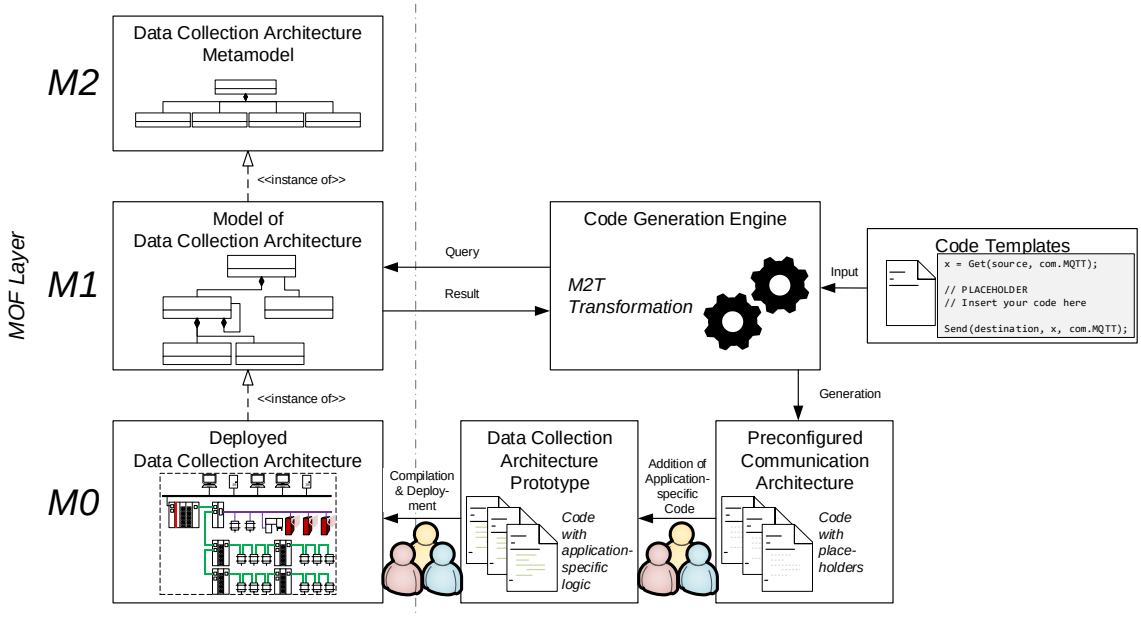
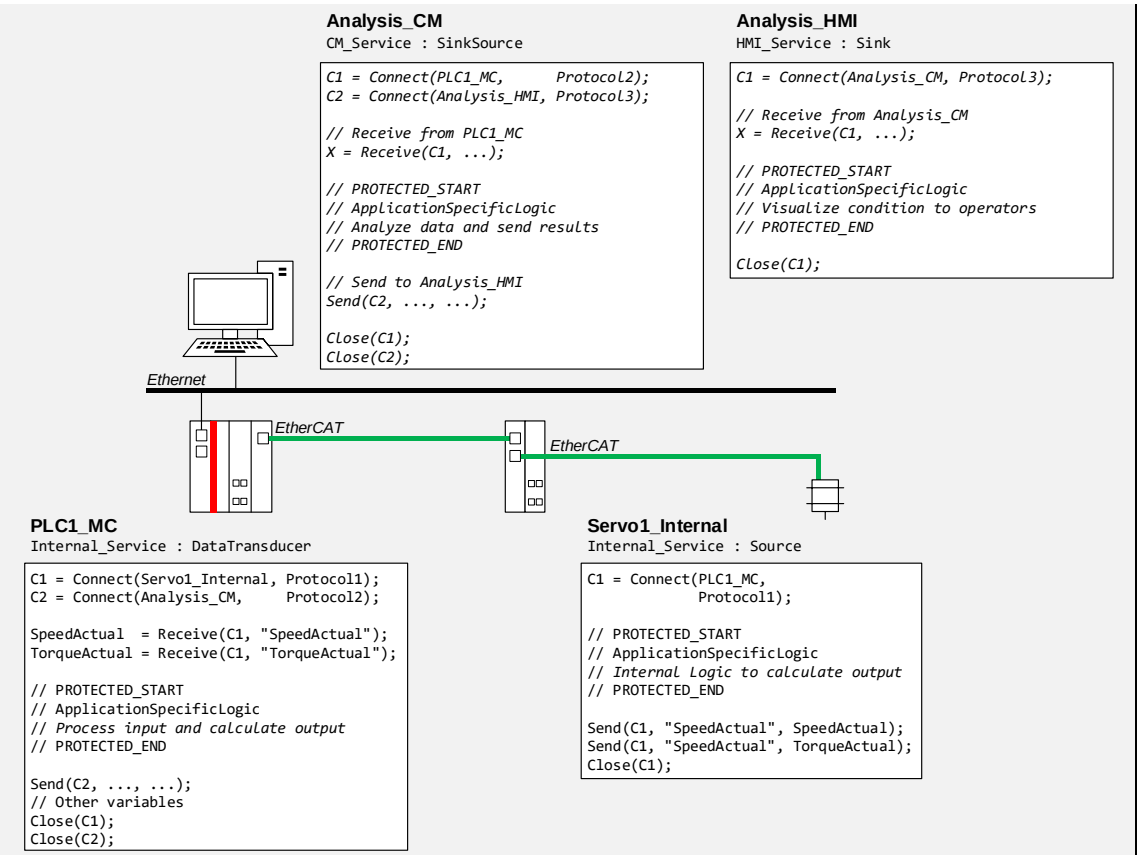


Figure 32: Overview over the process of transforming the model of the data collection architecture to a deployed instance via code generation and addition of application-specific code (adapted from Brambilla et al. [BCW17]). The left side reflects the different models and the related MOF layers. The right side illustrates the process of automatic generation of the communication architecture on OSI layer 7, as well as the addition of application-specific code and the deployment by experts.

AE.Part 11: Example of code generation for the application example.

The aim of the code generation is to generate the communication code for OSI layer 7 based on the modeled information. Therefore, the modeled data flow is taken as a basis to set up the direction of data transfer. Furthermore, the additional information in the form of IP addresses or hostnames is used to address specific systems. The code generation is focused on the communication part of the data collection architecture while generating protected sections where programmers can insert their application-specific logic that uses or modifies the data. AE.Figure XVII reflects the code generation with simplified pseudo-code for each system.

The generated code encompasses the functionality to automatically set up a connection between two related systems in a data flow and to handle this connection. In the application example, only direct communication without a distinct broker is modeled. Additionally, the code portions for receiving and sending data are generated. Experts can then insert their application-specific code into the protected code section (mimicked by `// PROTECTED_START/END`) here.



AE.Figure XVII: Simplified sample of generated pseudo-code for the application example.

6. Implementation

In this Chapter, a brief overview of the concept implementation is given. It includes the DSL, the software framework, and the model-driven generation of the communication architecture.

6.1. Domain-specific Language

The DSL consists, following the definition of a modeling language [Rod15], of a metamodel and a graphical notation. The data collection architecture metamodel is implemented within the Eclipse Modeling Framework (EMF), as the defacto standard framework for model-driven development [BCW17], in version 2.18 [Ecl19b]. As an editor for the metamodel, the Eclipse IDE with installed Eclipse Modeling Tools in version 4.13 / release 2019-09 is used [Ecl19c]. A graphical representation of the *PhysicalContainer* in the Eclipse Modeling Tools is given in Figure 33.

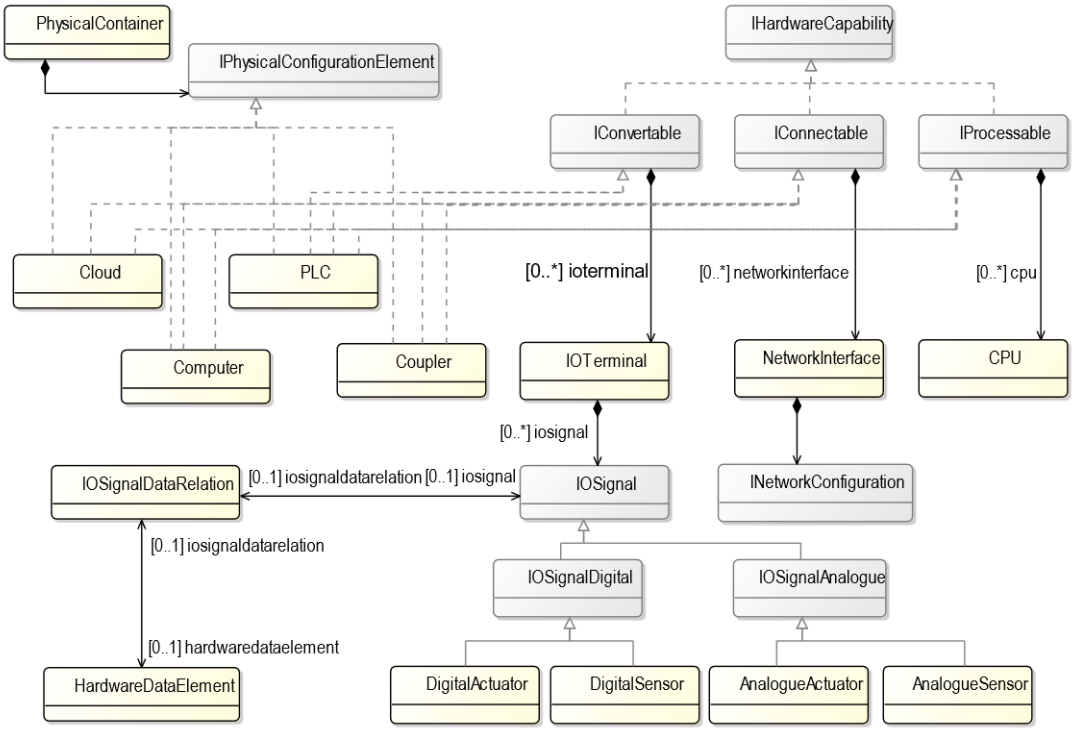


Figure 33: Excerpt of the metamodel modeled with the Eclipse Modeling Tools showing the PhysicalContainer (cf. Figure 26).

The graphical notation is provided as stencils for Microsoft Visio (see Figure 34) [Mic19a]. End users can graphically edit the model representation with the provided stencils. An automatic link between graphical representation and model instance of the metamodel is currently not part of the implementation. Existing tooling, such as Graphiti [Ecl19f] or Sirius [Ecl19a], could be used in the future to provide an integrated graphical editor in Eclipse.

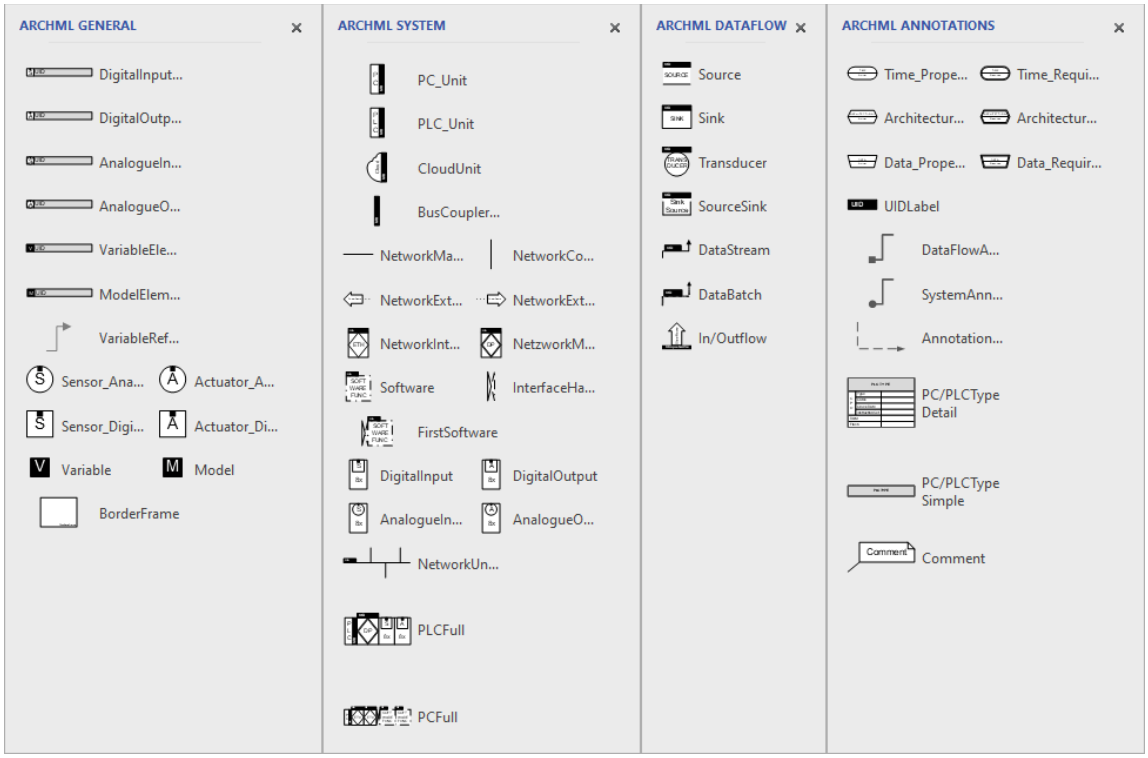


Figure 34: Screenshot of the Microsoft Visio stencils provided for the graphical modeling of data collection architectures.

6.2. Architecture Software Framework

The software framework is implemented with C# 8.0 [.NE19] and the .NET Core 3.1 framework [.NE20] in Visual Studio 2019 [Mic19b]. .NET Core is characterized by active development, an open-source MIT license, and a large ecosystem with broad availability of third-party libraries. C# is a state-of-the-art programming language for object-oriented programming and widely accepted for the realization of industry-scale software projects. The following communication technologies are implemented natively as implementations of *IReceiveService* and *ITransmitService*. They represent typical protocols for industrial communication (see Section 2.3.2).

- Apache Kafka [Apa19], with the library Confluent.Kafka [Con19] for .NET Core support;
- MQTT [ISO20922], tested for the Eclipse Mosquitto broker [Ecl19e], using the Open-NETCF MQTT library for .NET Core support [Tac19];
- OPC UA [IEC62541], using the OPC foundation's reference stack [OPC19]; and
- AMQP [ISO19464], tested for the RabbitMQ broker [Piv19a] and using the RabbitMQ .Net Client library [Piv19b].

The .NET Core framework offers cross-platform support, including Windows, macOS, as well as Linux on x86/x64 and ARM platforms. This cross-platform support provides great flexibility in heterogeneous environments, as can be found in industrial automation. In essence, the same code can be executed on servers that are operated under Linux, retrofitted gateways on cheap ARM platforms, and client computers for data analysis on Windows. Therefore, multiple protocols can be supported on various platforms using the lightweight and accepted .NET Core platform.

As not all communication technologies are supported by libraries for .NET Core, a flexible extension mechanism is foreseen. Therefore, Google gRPC [Goo19a] is implemented as an additional communication service. gRPC is based on Google Protocol Buffers [Goo19b] and is an open-source, high-performance remote procedure call (RPC) framework with cross-language support. The Protocol Buffer framework defines an interface description language (IDL) for the definition of data types and functionalities. These are platform and language independent. Via integrated code generators, language-specific code reflecting the definitions stated with the IDL can be automatically generated as part of gRPC. The architecture software framework provides a language-independent definition of a communication service and offers a gRPC endpoint that can be used by other applications. This gRPC endpoint allows the implementation of communication services in other programming languages and with libraries incompatible to .NET Core 3.1. Furthermore, the actual provider of the functionality (gRPC client in a different language) is decoupled from the gRPC endpoint of the framework. Both services can run on different machines and communicate over networks, allowing a decoupled microservice architecture.

For instance, the support for Beckhoff ADS [Bec19c] is implemented using the gRPC endpoint of the software framework (see Figure 35). Beckhoff to date only provides ADS client libraries for .NET framework 4.6, as well as other programming languages, which are all incompatible with the .NET Core 3.1 framework. Therefore, a decoupled microservice wrapping the functionality of ADS communication is part of the software framework and communicates with the core of the framework over gRPC for interoperability.

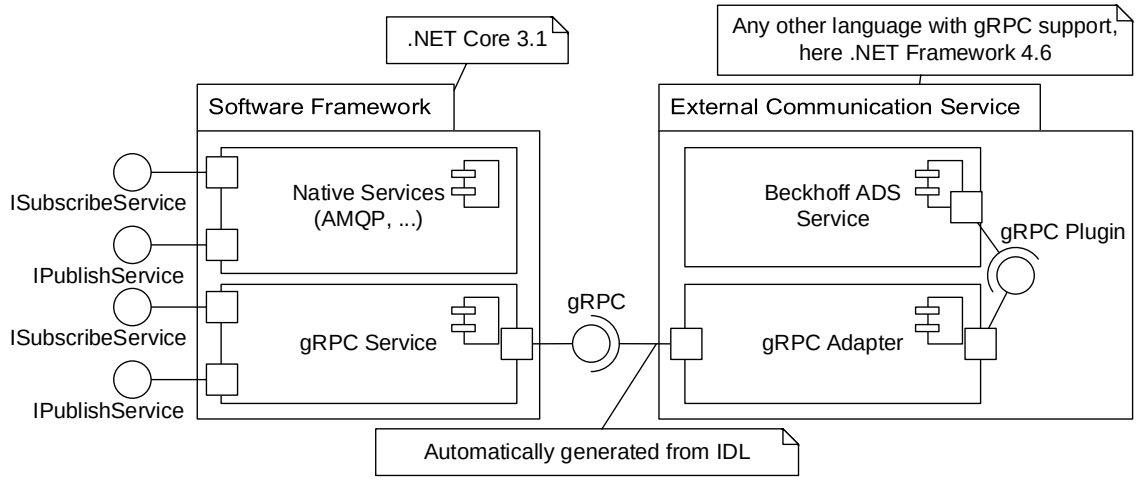


Figure 35: Working principle of the flexible extension mechanism via gRPC. Example of Beckhoff ADS support as an external communication service.

6.3. Automatic Code and Configuration Generation

Acceleo is an implementation of the OMG MOF Model to Text Transformation language specification [OMG08] by the Eclipse Foundation. The model transformation for the automatic generation of the communication architecture and configuration of brokers is implemented with the Acceleo transformation language in version 3.7.8 [Ecl19g]. Modular code templates distribute the transformation logic into smaller files and simplify maintenance of the transformation.

The templates are based on the C# implementation of the software framework and rely on the .NET Core 3.1 framework and the libraries described in Section 6.2. Middleware configurations are text files and individually created during the model transformation, depending on the configuration characteristics of each middleware. The code generation includes templates for the brokers mentioned in Section 6.2. All model transformations are initially set up for the creation of the model-driven approach and can be reused for all subsequent applications.

Figure 36 shows an example of an M2T transformation in the Acceleo transformation language for the instantiation of communication services. The first template (`serviceInstantiations`, lines 1 to 8) generates the code for the instantiation of communication functionalities. It, therefore, iterates through every applicable model instance (line 2), uses the provided code template to generate the corresponding code, and replaces the blanks (gray background) with the information from the model. In the last line of the first template (line 6), the instantiation of the communication service takes place. The second template (`simpleServiceConfig`, lines 10 to 18) is called from the first template and is used for extracting and generating the service configuration, including IP addresses, ports, and credentials from the model. Figure 37 depicts a possible output from the shown transformation snippet for an MQTT service (here called MosquittoService).

```
[template public serviceInstantiations(swPackage : SoftwarePackage)]
  [for(service : IConnectionService | swPackage.iconnectionservice)]
    var [getServiceId(service)/]Factory =
      new [factoryFor(service)/]([simpleServiceConfig(service)/]);
    [getServiceId(service)/] = ([serviceInterfaceFor(service)/])
      [getServiceId(service)/]Factory.CreateService();
  [/for]
[/template]

[template private simpleServiceConfig(service : IConnectionService)]
new ServiceConfig
{
  Server =  "[getTargetIp(service)/]",
  Port =     [getTargetPort(service)/],
  UserName = [usernameFromService(service)/],
  Password = [passwordFromService(service)/]
}
[/template]
```

Figure 36: Example of Acceleo M2T transformations for instantiation of communication services. Blanks with gray background.

```
var SomePublisherFactory = new MosquittoServiceFactory(new ServiceConfig
{
  Server =  "127.0.0.1",
  Port =     1884,
  UserName = "foo",
  Password = "bar"
});
SomePublisher = (ITransmitService)SomePublisherFactory.CreateService();
```

Figure 37: Example for generated C# code from the M2T transformation in Figure 36. Filled blanks with gray background.

The templates contain protected sections to ensure that user-added code (application-specific implementation) is not overwritten when the model transformation process is executed repeatedly to update the generated software code. Furthermore, the model to text transformation generates log files that can be used to trace the transformation process and verify its correctness. Additionally, Visual Studio 2019 project files for .NET Core 3.1 are set up, which allow a comfortable building of the respective projects. These project files include the necessary references to the underlying communication libraries, as well as the compiler configuration, and the respective shared libraries as DLLs (dynamic-link libraries). Furthermore, to allow the creation of portable and lightweight containerized applications, the projects include descriptions to create Docker containers (so-called dockerfiles) automatically. The created containers include the compiled executables as well as the required communication libraries and all additional dependencies (e.g., the .NET Core 3.1 runtime itself).

The model transformation and the deployment of the compiled docker containers are automated with a build pipeline. Therefore, after the model transformation step, users can create or update their application-specific implementation manually. Afterward, the code is pushed to a Git [Git20] repository used for version management, as well as continuous integration (CI) and deployment

[Fow15; FS17]. The build pipeline for CI is configured to generate the executables of the respective projects on every update in the repository and subsequent cross-compilation of the corresponding Docker containers for multiple platforms, including Linux x86, Linux x64, as well as ARMv7, via the buildx system [Doc20c]. After a successful compilation, the Docker images are published to a local Docker repository [Doc20a] that manages all Docker images.

All non-legacy systems of the use-cases execute their own Docker runtime and are connected to a central, so-called node manager. This manager orchestrates all connected clients using the Docker swarm mode [Doc20b]. If the corresponding image that is associated with a client is updated on the registry server, the local copy of the image can be automatically replaced by the newer version. Furthermore, the node manager allows monitoring of all connected clients, as well as enhanced configurations for fail-over operation and distribution of images across multiple clients for scalability. This CI-pipeline simplifies the deployment of updated configurations into operations and allows a flexible and agile software development.

7. Evaluation

The developed concepts for model-driven data collection architectures will be evaluated using the requirements formulated in Chapter 3. For this purpose, various evaluation scenarios and methods are employed, each addressing distinct aspects of the requirements. Table 14 summarizes the requirements and maps them to the evaluation scenarios and the corresponding Sections.

The evaluation is split into six major parts; these are:

1. interviews with industry experts and mapping to other state-of-the-art architectures to assess the technology-neutral architecture concept (Section 7.1);

2. expert evaluation of the graphical modeling notation with semi-structured interviews in four industrial case-studies conducted with industrial experts from the domain (Section 7.2);

3. a lab-scale feasibility study including the graphical modeling of the overall systems and a subsequent automatic code generation with deployment to the lab environment (Section 7.3);

4. a code generation for one of the industrial case-studies modeled in Section 7.2 to evaluate the scalability of the approach (Section 7.4);

5. an estimation of the implementation effort using the developed approach in comparison to classical, non-model-driven programming using minimal clients and extrapolation of the corresponding efforts (Section 7.5); and

6. a questionnaire with industrial experts concerning the overall approach in comparison to the current industrial practice (Section 7.6).

The results of this chapter are used to assess the fulfillment of requirements in the subsequent Chapter 8.

Table 14: Evaluation scenarios per requirement and reference to the relevant Sections.

	Requirement		Expert interviews on architecture concept	Mapping to other proposed architectures	Industrial case-studies with expert interviews	Lab-scale case-study	Industrial case-study	Effort extrapolation case-study	Expert questionnaire
	Section		7.1.1	7.1.2	7.2	7.3	7.4	7.5	7.6
Data Collection System Architectures (*Req-A*)	*Req-* A_{ATP}	Data collection from different levels of the automation pyramid	●			●			●
	Req- A_{TAC}	Technology-agnostic concept	●	●					
	Req- A_{POP}	Parallel operation to pyramid architecture	●			●			●
	Req- A_{Dep}	Simplified implementation and configuration				●		●	●
	Req- A_{ReDep}	Simplified migration between technologies				●		●	●
Software Framework (*Req-SF*)	*Req-* SF_{API}	Standardized interfaces to minimize effort				●			●
	Req- SF_{ACP}	Abstraction of technology-specific properties of communication				●		●	●
	Req- SF_{Leg}	Support for legacy systems	●			●			
Architecture Modeling Language (*Req-M*)	*Req-* M_{Sys}	System viewpoint			●				
	Req- M_{DF}	Data flow viewpoint			●				
	Req- $M_{PropReq}$	Annotations for properties and requirements			●				
	Req- M_{Graph}	Graphical modeling notation			●				
Model-driven Generation (*Req-G*)	*Req-* G_{Com}	Model-driven generation of communication interfaces				●	●		

A mapping of the evaluation scenarios to the building blocks of the concept is illustrated in Figure 38. While the expert interviews and the mapping to other state-of-the-art architectures evaluate the generic, technology-neutral architecture concept, the industrial case-studies are used for the assessment of the graphical modeling notation as part of the DSL. The following case-studies (lab-scale, industrial, and effort extrapolation) are dedicated to the interplay of the DSL, the software framework in the form of code templates, and the model-driven generation of the data collection architecture. The last Section, the expert questionnaire, covers aspects that characterize the concept as a whole.

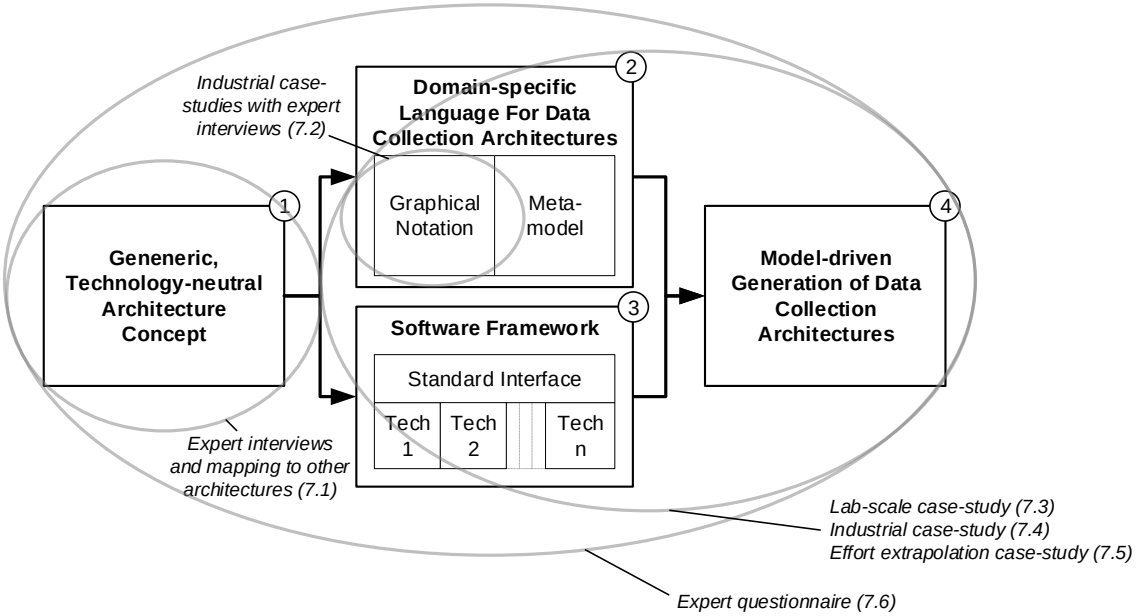


Figure 38: Graphical mapping of case-studies to the parts of the concept's building blocks.

7.1. Evaluation of Architecture Concept

This Section captures the evaluation of the developed architecture concept. The first part describes the results of semi-structured interviews with industrial experts. The second part presents a mapping of the concept to other system architectures proposed in state-of-the-art projects to show the technology-neutrality of the concept and its generalizability.

7.1.1. Interviews with Industry Experts

The developed architecture concept was evaluated via the conceptual application of the architecture to two distinct scenarios and subsequent, semi-structured interviews with a total of five industry experts. All involved experts have profound knowledge of the required data sources, the integration of the relevant data, and the existing system architectures currently in operation. The results of these interviews were initially published in [Tru+17].

The first use-case is related to live monitoring and predictive maintenance of valves in the chemical process industry via data analysis and stems from the SIDAP project [SID19]. SIDAP involves the data exchange across the life cycle of valves to increase the value of data analysis as data is dispersed into different data silos. For instance, while plant operators have data about the operation of a valve, the original equipment manufacturer has extended knowledge of the physics and specification of valves. Hence, the analysis requires data from several existing distributed systems. These include the measurements from the valves themselves (e.g., valve stroke), historical measurements from a superordinate historian (in this case, an OSIsoft PI system [OSI19]), as well as valve specifications from the engineering and maintenance documentation from an ERP. All mentioned systems are existing legacy systems and have their specific interfaces and protocols for communication. Moreover, live monitoring would require the implementation of at least one data analysis component that collects and analyses the data streams, as well as a visualization dashboard for the operators. Additional existing legacy applications must be considered, as well. In collaboration with the industrial experts and within an offline data analysis based on historical data, the relevant data sources were identified. Afterward, the architecture was conceptually applied for the use-case of a valve monitoring and predictive maintenance platform across multiple involved partners. A representation of the developed architecture can be seen in Figure 39.

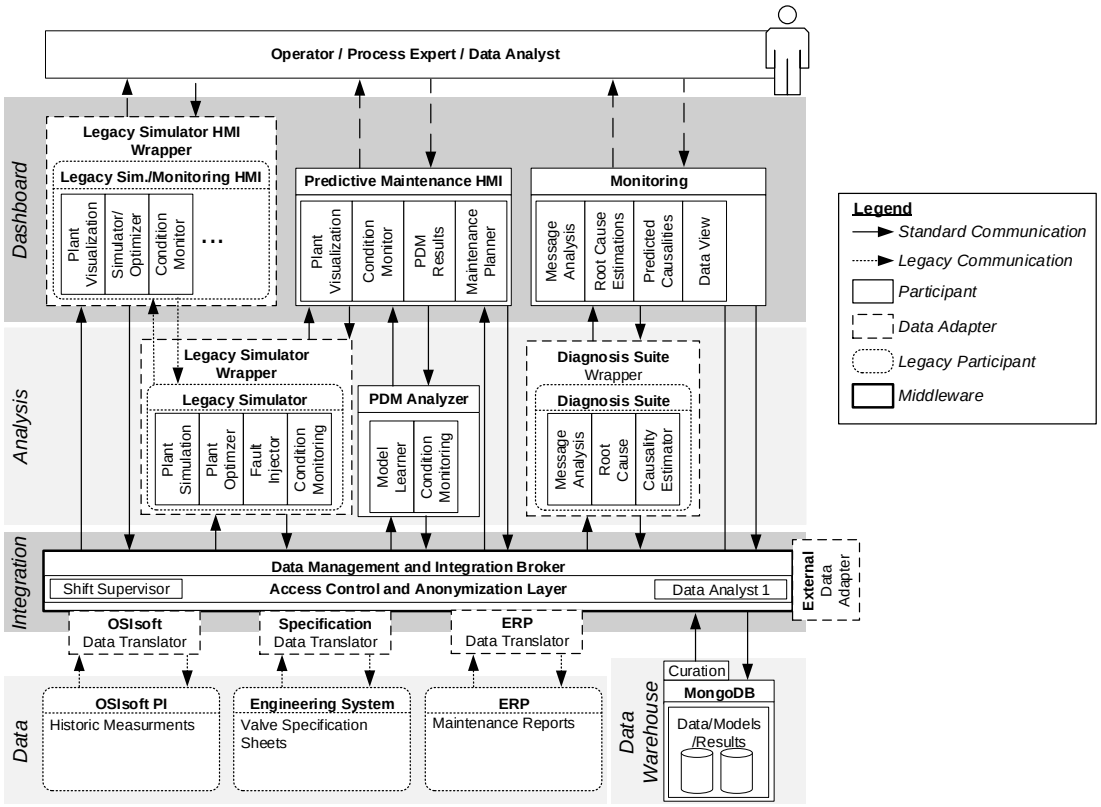


Figure 39: Representation of the conceptualized architecture for the SIDAP use-case (graphically adapted from Trunzer et al. [Tru+17]).

The second use-case stems from the project IMPROVE [IMP19]. The aim of IMPROVE is the creation of a virtual factory as a virtual representation of a real production facility that can, for instance, serve as a basis for off-line optimization of production parameters. This use-case requires transparent and fast access to data from various sources from within and outside of the automation pyramid, with numerous legacy systems. The systems include data from the PLCs of the production plants, superordinate information from SCADA and MES systems, as well as off-line quality measurements from a lab database. Furthermore, the concept of a virtual factory requires the implementation of various analyzers to monitor plant operation, a simulation, and an optimization engine that allows off-line optimization of the production processes, as well as dashboards that visualize the data and information for human operators. A visual representation of the conceptually-applied architecture is given in Figure 40.

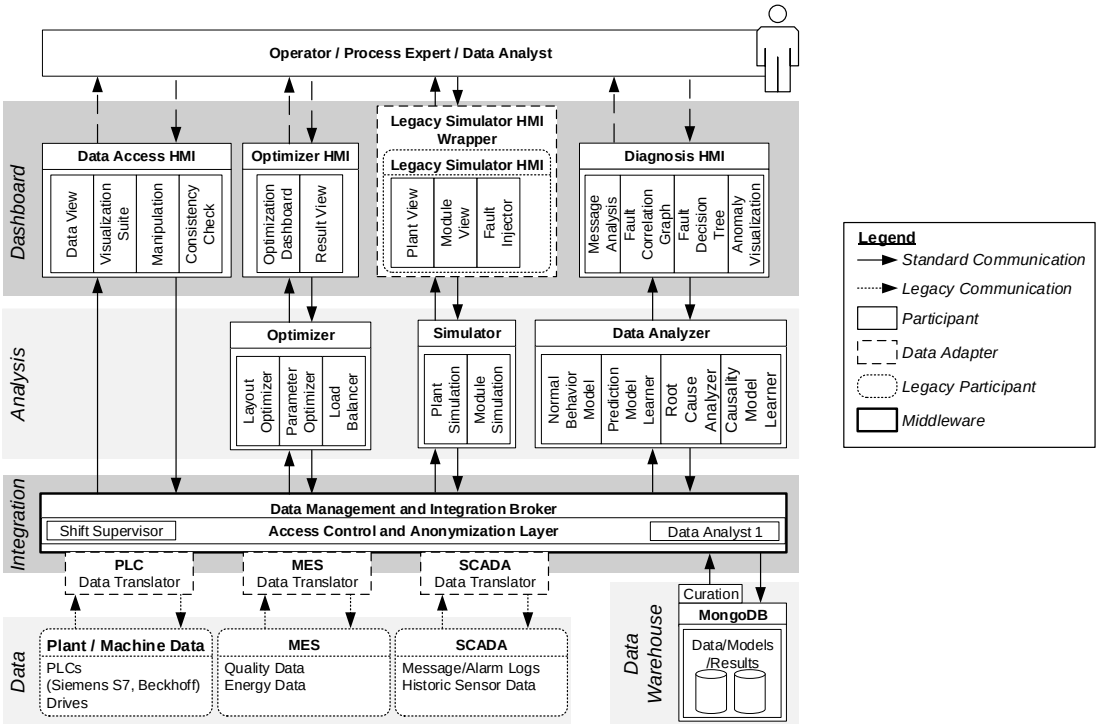


Figure 40: Representation of the conceptualized architecture for the IMPROVE use-case (graphically adapted from Trunzer et al. [Tru+17]).

A large number of legacy systems characterize both use-cases as part of an existing IT/OT landscape. According to the experts, the introduction of the architecture would replace and automate data collection and integration tasks that are currently carried out by hand. Furthermore, the conceptualized architecture for the IMPROVE use-case shows parallels to an ongoing internal effort in one of the involved companies. The two architectures feature a central data warehouse component that can be used to store data, models, and results. In both cases, it is conceptualized as a MongoDB [Mon19] database accessible by all systems connected to the Data Management and Integration Broker. This data warehouse is optional and allows persistent storage of data across a

system landscape, reliefs the individual systems from storing separate copies of relevant data, and can be used as a building block for the so-called Lambda architecture proposed by Marz and Warren [MW15]. The Lambda architecture is a paradigm for data analysis architectures in a big data environment that must handle large batches of historical data as well as streamed real-time data during the analysis. By a separate analysis of the two data types in different layers and subsequent combination of the results, a Lambda architecture can provide accurate results with low latency.

The expert interviews aimed to evaluate the feasibility of the conceptual architectures and their suitability for the use-cases compared to the existing infrastructure. Therefore, two semi-structured interviews were conducted, one with the experts from the SIDAP project, and another with the IMPROVE experts. In the interviews, the experts were sure that the installation and operation of the conceptualized architecture in parallel to the existing control and automation infrastructures is viable (*Req-*A_{POP}). In their eyes, such a decoupled design separates the control and operations domain from the data collection and analysis process. This decoupling is especially beneficial for mission-critical production systems. In the use-cases, data from different levels of the automation pyramid (PLCs, SCADA, MES, and ERP systems) is collected and forwarded to the respective client systems (*Req-*A_{ATP}).

Another aspect is the integration of legacy systems over data adapters in heterogeneous environments. Existing interfaces and connections between legacy systems stay untouched and functional (cf. the connection between the legacy simulator and its HMI in Figure 39). The efficient integration of existing legacy systems and newly developed systems is perceived very positively by the experts. The flexibility of integrating legacy systems with different data adapter principles, ranging from translators embedded into the Data Management and Integration Bus to Data Wrappers (cf. Section 5.1), was highlighted by the experts (*Req-*SF_{Leg}). Nevertheless, the experts considered the implementation effort for initial deployment as relatively high and complex. Besides, the implementation effort to program all necessary data adapters is a significant obstacle. Therefore, a step-wise introduction and migration are proposed. This migration scenario minimizes the initial effort for deployment and makes more and more data transparently available over time.

Both conceptualized architectures are not bound to a specific implementation technology and can be implemented using various available technologies. Depending on the specific requirements of a realization, a suitable technology can be selected. Even combinations of technologies are possible. For instance, the system could be implemented using a commercially-supported ESB, such as IBM Integration Bus, an open-source alternative (e.g., RabbitMQ), or a high-throughput system such as Apache Kafka. Therefore, the experts stated that the architecture concept and its application to the use-cases are technology-agnostic (*Req-*A_{TAC}).

7.1.2. Mapping to State-of-the-Art Architectures

In the second part, a mapping of the developed system architecture to the system architectures of the PERFoRM and BaSys 4.0 projects (see Figure 41) is presented. Both projects developed state-of-the-art system architectures for Industrie 4.0 applications and address various aspects of system integration in industrial automation. The results were initially published with coauthors from the two other projects in [Tru+19c].

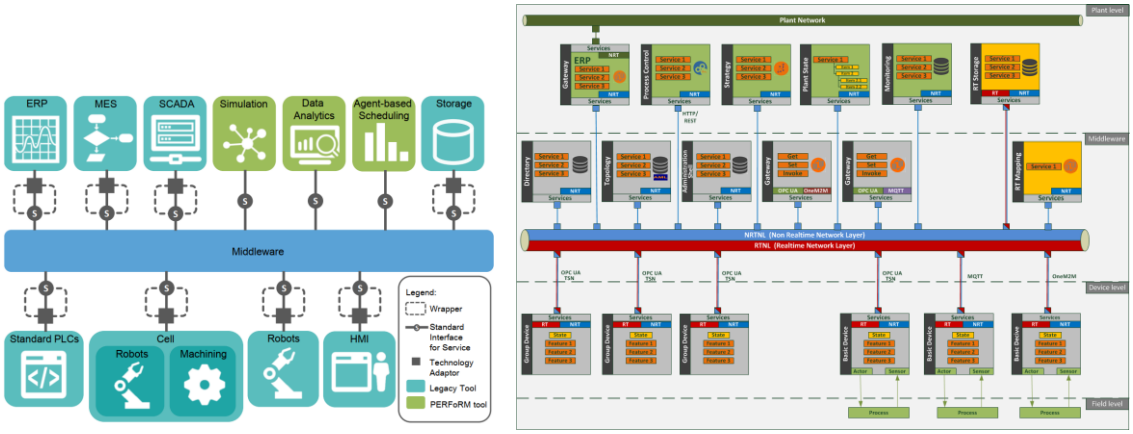


Figure 41: PERFoRM (left) [Lei+16] and BaSys 4.0 (right) [Tru+19c] architecture concepts.

The three system architecture concepts (BaSys 4.0, PERFoRM, and this work) originate from different use-cases, are subject to distinct boundary conditions, and are tailored for their specific field of application. While the architecture conceptualized in this work aims to simplify data collection and analysis, the PERFoRM architecture concept aims to provide the possibility of a reconfigurable production, and BaSys 4.0 a real-time communication between systems. The PERFoRM architecture concept was demonstrated to be implementable using various technologies [Cha+17; Gos+18; PER16b]. In contrast to this work and PERFoRM, BaSys 4.0 relies on a replacement of the existing automation architecture and is bound to an implementation framework that includes a so-called Virtual Automation Bus (VAB) as the middleware component for real-time communication [Kuh+18]. Still, all three approaches share a substantial number of similarities. In essence, all three foresee a common communication bus, usage of a single protocol to interface systems, the integration of legacy systems via data adapters (or administration shells in BaSys 4.0), and consider a layered architecture.

A generic architecture applicable to the respective application fields and their use-cases, which was derived in [Tru+19c], is very similar to the one conceptualized in this work. The main differences between the derived architecture and the concept of this thesis are the introduction of a real-time communication channel and the added support for service detection and orchestration. However, on the one hand, as this approach is not aiming at replacing the existing control structure, no

real-time communication is needed inside the architecture. If no real-time communication is required, the BaSys 4.0 architecture concept would not be bound to the VAB and hence implementable with a wide variety of available technologies. On the other hand, service detection and orchestration are additional functionalities that the connected systems must support, not the system architecture itself. As the core of the three system architecture concepts is remarkably similar, and their realization is not bound to a specific technology, the presented concept for a data collection architecture can be seen as technology-agnostic (*Req-*A_{TAC}). Therefore, the abstract, technology-neutral architecture concept of this thesis can directly be applied to all considered use-cases and implemented using various available technologies.

7.2. Expert Evaluation of Graphical Modeling Notation

The graphical modeling notation of the developed DSL was evaluated by applying the notation to four industrial use-cases (subsequently called Case-Studies A to D) and successive, semi-structured interviews with industrial experts. The evaluation procedure is shown in Figure 42 and will be explained throughout this Section.

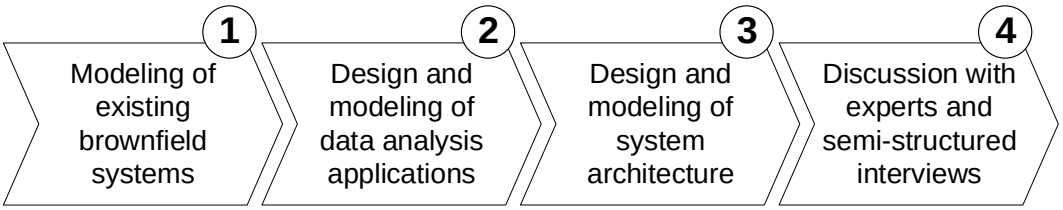


Figure 42: Procedure for the expert evaluation of the graphical modeling notation.

The four use-cases reflect typical and representative applications of data collection architectures that interact with CPSoS and bridge IT and operational technology. They include data collection from a multitude of heterogeneous systems, including legacy systems, and the involvement of experts from various domains. All four are modeled with the help of experts from different domains. In total, eight experts (2 per case-study), which have profound and long-term expertise in their respective fields of application, were questioned. The selected experts are all employed by the respective OEMs. They are particularly qualified to evaluate the notations as they have great industrial experience in the realization of data collection and analysis projects. Additionally, all have an interdisciplinary background from at least two domains relevant to the use-cases (technical experts, data analyst, IT architect, control engineer). The experts are, for instance, heads of information technology or senior engineers for digitization in their respective companies. Due to confidentiality, the boundary conditions and the conceptualized architectures are modified slightly for this thesis (for instance, different protocols, single systems connected to other networks, or abstraction of company-specific information related to security configurations). Evaluation results were partly published in [TWV20], but cover only the three case-studies B to D.

Based on the documentation provided as well as input from technical experts and IT architects, the brownfield production systems, without any additional data analysis, were modeled using the graphical modeling notation (step 1 in Figure 42). These diagrams were then adapted and extended with the help of data analysts. They stated which data is needed for the analysis and where the analysis models should be deployed (edge, cloud). Furthermore, they expressed additional requirements, e.g., allowed latency and sample rates (step 2 in Figure 42). [TWV20]

In the next step (step 3 in Figure 42), IT architects drafted the adapted system architecture with additional data analysis components. Supplemental requirements, such as data security (encryption, authentication), communication (protocols, semantics), and system sizing (scalability, the capacity of storage), were specified and added to the models. [TWV20]

The extended models were then discussed with the experts in joint sessions (step 4 in Figure 42). This first part of the qualitative evaluation was to verify the correctness of the models. Afterward, a structured questionnaire with a total of 20 qualitative questions about the clarity of the graphical notation, its syntactic constructs, and its completeness was conducted in the joint session. The questionnaire was divided into four parts: syntax and completeness of the system viewpoint, syntax and completeness of the data flow viewpoint, mapping between the two views and annotation elements, and clarity of the graphical notation. [TWV20]

Table 15 summarizes the main characteristics of the use-cases. In the following Section, the detailed model of the data collection architecture for Use-Case A is discussed. Afterward, an overview of Use-Cases B to C is given (the corresponding models can be found in 14). Subsequently, the results of the expert interviews are presented.

Table 15: Summary of use-cases for expert evaluation of graphical modeling notation.

Use-Case	Analysis application	Type of architecture	№ experts	№ employees / company size	Section with graphical models
A	Condition monitoring	Private / public cloud	2	~ 1,500	7.2.1
B	Anomaly detection	Cloud / edge architecture	2	~ 400	Appendix A.1
C	Alarm management system	Public cloud	2	~ 7,000	Appendix A.2
D	Alarm management system	Hybrid cloud	2	~ 500	Appendix A.3

7.2.1. Use-Case A: Retrofitting and Condition Monitoring

Use-Case A captures a CNC machine retrofitted with additional sensors and control hardware for condition monitoring. The full system diagram is shown in Figure 43 and is explained and derived in the following. The CNC machine (*Machine11*) is provided by an OEM and includes a closed legacy system for control. A Siemens Sinumerik 840D control unit is installed inside the machine, but no modifications to the original control code are possible. The control unit provides various continuous variables (*S7_Var1* to *S7_Var13*), as well as additional, event-based data points (*S7_Event1* to *S7_Event37*). Still, the machine condition is not fully characterized by these available data points. Therefore, an additional Beckhoff PLC (*GW11*) is installed and connected to various additional bus couplers (*ST1* to *ST10*) with sensors over EtherCAT (*ECAT1*). The additional sensors capture further data (*Var1* to *Var33*) relevant to characterize the machine's condition.

Additionally, the Beckhoff PLC is also used as a gateway between the CNC machine and superordinate systems, as it acts as a central data collector. To further decouple the machine's main PLC from the Beckhoff PLC, an additional gateway (*PI11*) is installed as a Raspberry Pi-based KUNBUS RevPi device. This gateway collects the data from the machine PLC over Profinet, translates the format, and forwards it to the Beckhoff-based data collector on *GW11* via Profibus. *GW11* collects all data (machine PLC and additional data) and forwards it to the distributed control system (*DCS*).

Additionally, two cloud environments are part of the use-case: a local, on-premise cloud for internal analysis (*IBMPMQ*) and a public cloud (*AZURE*) that facilitates the monitoring of multiple machines across production sites. As the AZURE cloud environment is hosted on the internet, the connection between factory network (*ETH1*) and *AZURE* is a bottleneck for data transmissions. Besides, a production site can contain several machines that are all subject to this limited connectivity. All local systems connected to *ETH1* are configured to be part of the VLAN with ID 3.

At the current stage, the aim of the architecture is the collection of data for the generation of a historic database reflecting the operating conditions of the plant. This database can be used to train data analysis models. In the future, the data from the machine should be leveraged as a continuous data stream to monitor the condition of all connected machines using a fleet management approach and trained analysis models.

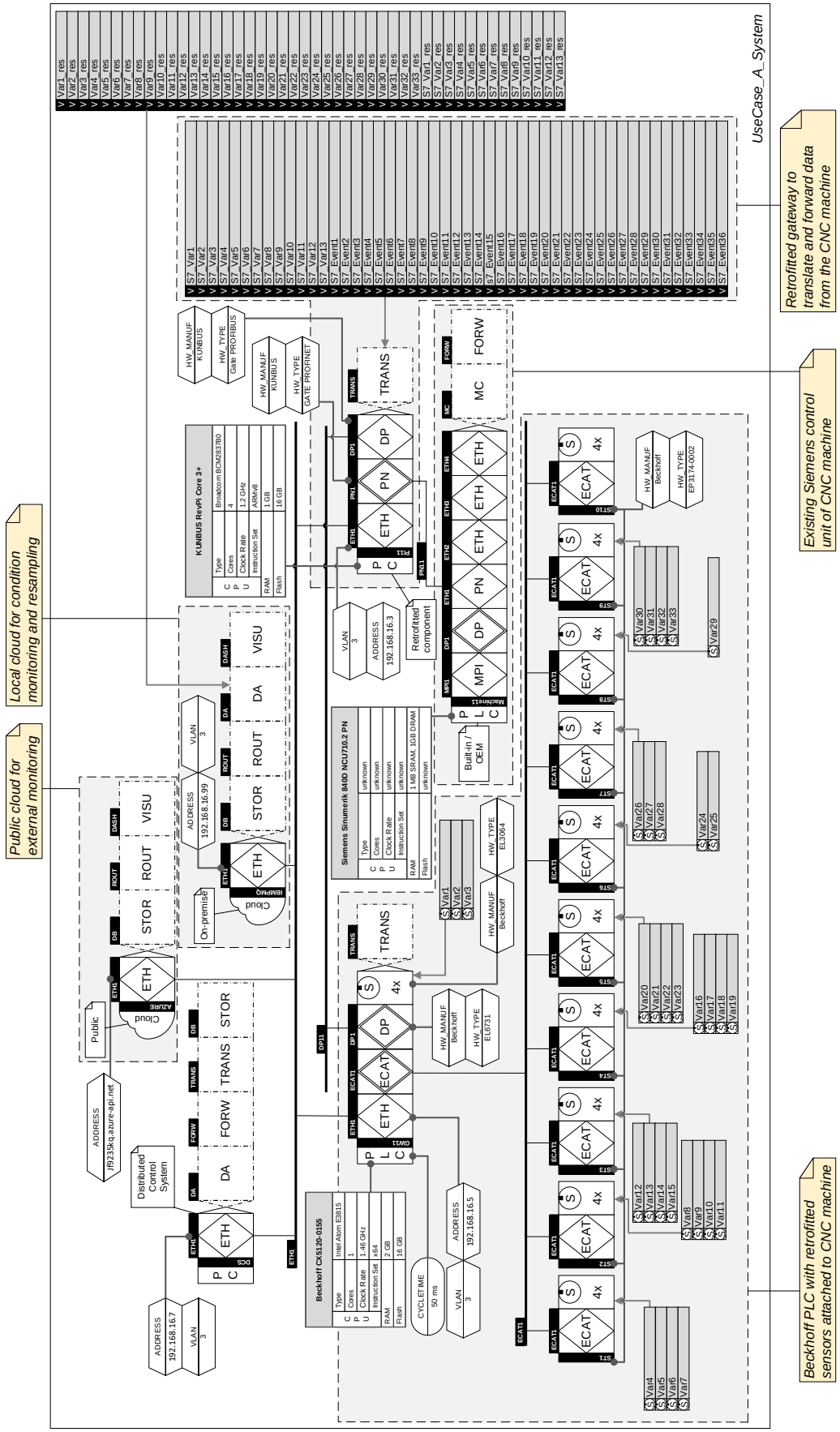


Figure 43: Retrofitted condition monitoring system of Use-Case A modeled in the system viewpoint.

Two distinct data flows can be distinguished in the use-case: the flow of event (alarm) data (see Figure 44) and the flow of continuously measured time series (see Figure 45). Both will be explained in more detail in the following.

The event data from the machine's PLC (*S7_Event1* to *S7_Event36*) is forwarded to the *PI11*, its data format modified, and then sent to *GW11*. Event data is not generated continuously, hence its frequency is comparatively low. Therefore, it can be directly forwarded also to the cloud environment. The collection of data from multiple machines in a single production facility should not overload the connection between the factory and the public cloud (*AZURE*). Hence, data is forwarded from *GW11* directly to *AZURE*, the local cloud (*IBMPMQ*), the storage system of the *DCS* (*DCS.DB*), and the data analyzer embedded into the DCS (*DCS.DA*) for the calculation of production KPIs. Both cloud environments consist of a routing component, storage, and a dashboard for visualizing the raw data. The resulting data flow is modeled in Figure 44. All flows are marked as batch data as they do not transmit data continuously but on a sporadic, event-triggered basis.

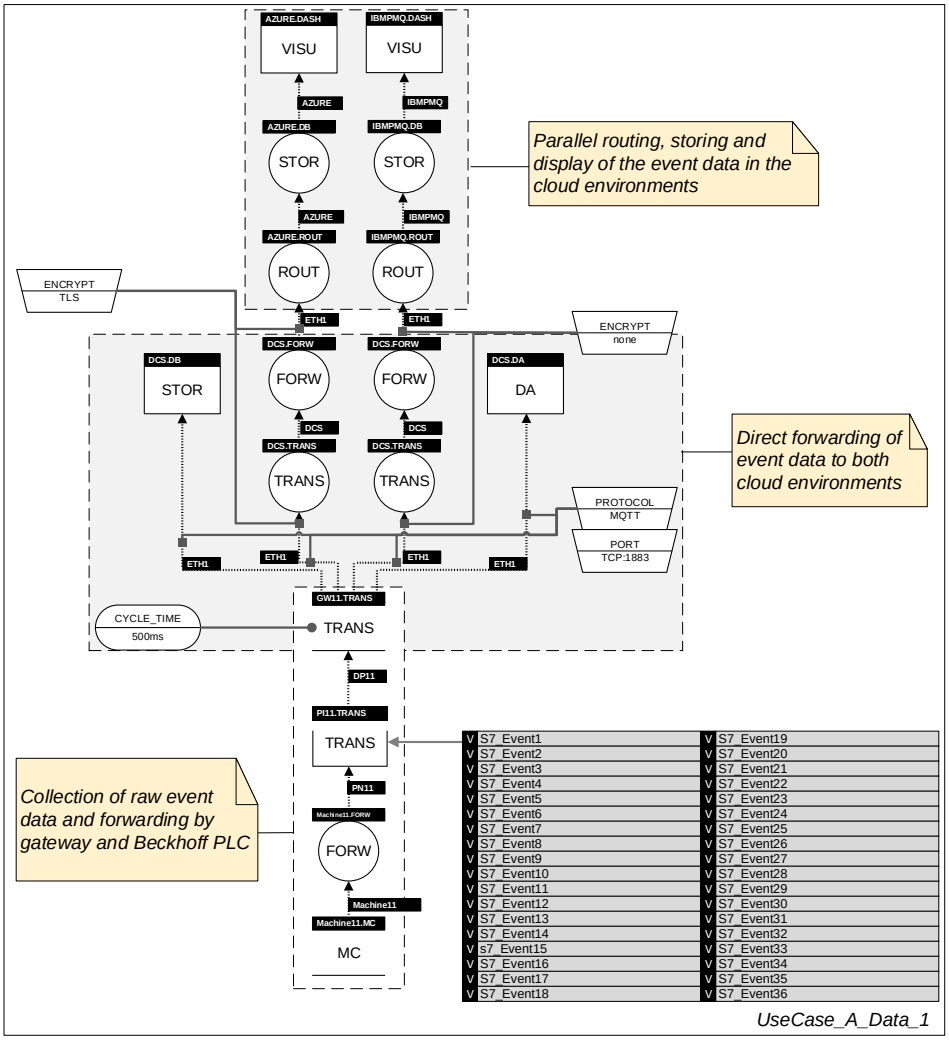


Figure 44: Data flow diagram of Use-Case A with the description of event-based data.

In contrast to the flow of event data, the other variables (*S7_Var1* to *S7_Var13* and *Var1* to *Var33*) are continuously measured time series. Interfacing several machines at the same time can overload the internet connection to the public cloud (*AZURE*). Therefore, the variables are downsampled in the private cloud to a lower sample frequency before forwarding them to the public cloud. The respective data flow can be seen in Figure 45. While the Beckhoff PLC works with a cycle time of 500 ms, the sample time of the directly connected system (*DCS.DB*, *DCS.DA*, and *IBMPMQ.DA*) is 1 s. The analysis functionality of *IBMPMQ.DA* resamples the incoming data to the sample time of 10 s and, therefore, further reduces the amount of data by a factor of ten. The resampled data is then forwarded to the public cloud.

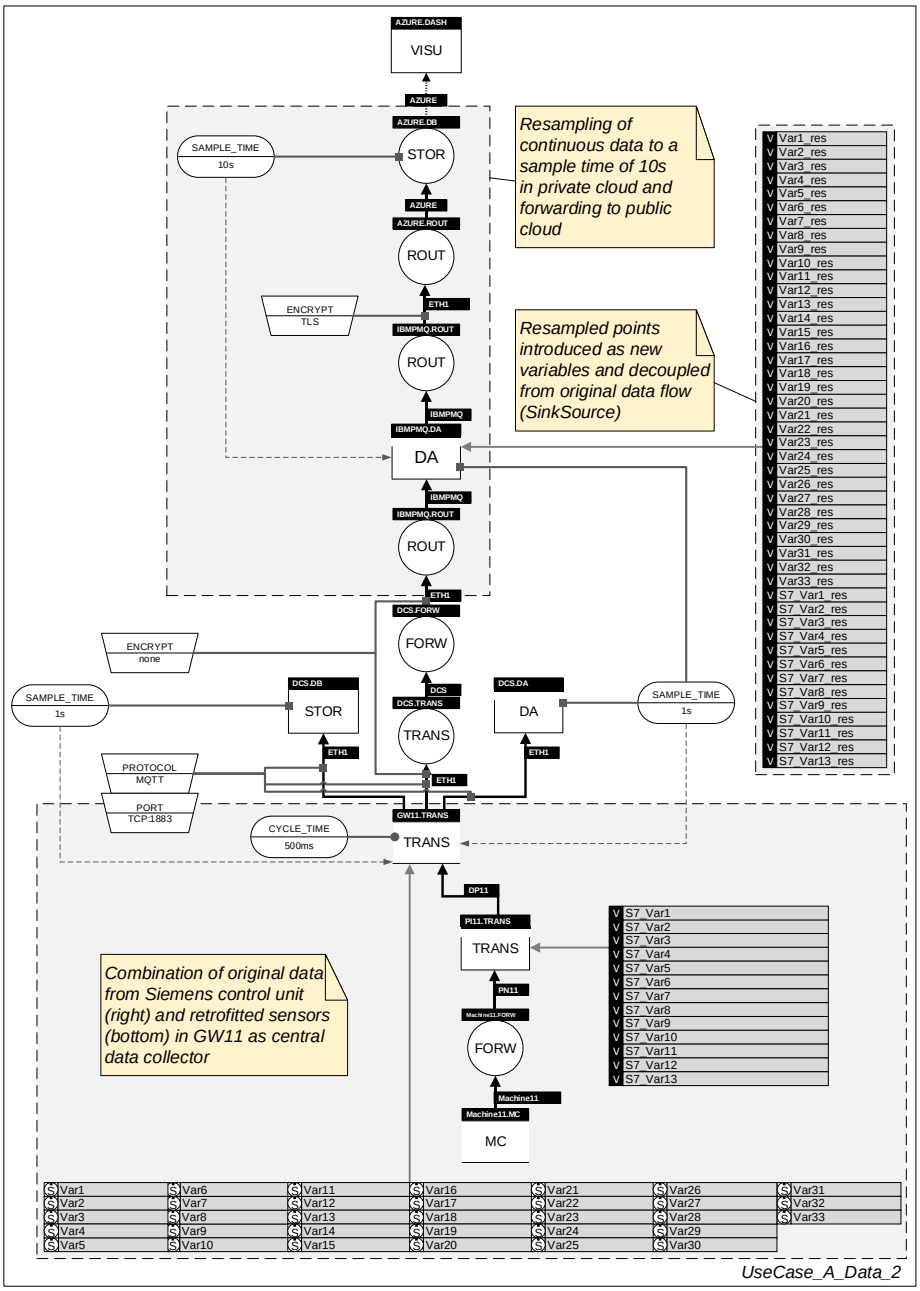


Figure 45: Data flow diagram of Use-Case A with the description of continuous data.

Table 16 gives an excerpt of the full data mapping table for Use-Case A. All variables are marked as time series as they contain the primary data element and an associated timestamp. Nevertheless, only the non-event data is flowing continuously with the cyclic update frequencies shown in the data flows.

Table 16: Excerpt of the data mapping table for Use-Case A.

VariableUID	SystemUID	SystemSpecific VariableUID	Derivedfrom VariableUID	Description	Address	Type	Resolution	Timeseries
Var1	GW11	Var1		Variable from gateway	GW11.1.1	FLOAT	16 bit	yes
S7_Event1	PI11	S7_Event1		Event message from Sinumerik		BOOL		yes
Var1_res	IBMPMQ	Var1_res	Var1	Variable from gateway, resampled		FLOAT	16 bit	yes
Var2_res	IBMPMQ	Var2_res	Var2	Variable from gateway, resampled		FLOAT	16 bit	yes
S7_Var1_res	IBMPMQ	S7_Var1_res	S7_Var1	Variable from Sinumerik, resampled		INT32		yes
S7_Var2_res	IBMPMQ	S7_Var2_res	S7_Var2	Variable from Sinumerik, resampled		STRING		yes

7.2.2. Use-Cases B to D: Anomaly Detection and Alarm Analysis

The detailed models (system and data flow viewpoints) of Use-Cases B to D can be found in 14. In the following, only a brief overview of the use-cases is given.

The second use-case (B, see Figures 55 and 56 in Appendix A.1) foresees a combined cloud and edge architecture for anomaly detection. Four to five production plants with one PLC each and several hundred in- and outputs per PLC are connected to a shared cloud environment. Therefore, a Siemens IPC communicates with various bus couplers on the field level and executes the machine control program. This program calculates additional variables in the control logic. Additionally, a local computer is connected to the IPC over the field bus and hosts a human-machine interface to visualize process values and to interact with the production process. Based on the data that is available on the IPC, an anomaly detection system is implemented. Therefore, the raw data is forwarded to a public cloud hosted by the OEM of the production machine. As the amount of data to be transmitted to the public cloud environment is subject to internet connectivity, data compression is executed. Hence, the Siemens IPC compresses the data (*DA_CP*) and forwards it to the cloud environment, where it is decompressed by a second system (*DA_RC*) and forwarded to the database. In the cloud environment, data from multiple machines is processed to train an anomaly detection model. The trained models are then sent back to the field level and executed on the edge (*DA_RA*) to detect anomalies during the production process.

The two other use-cases (C, see Figures 57 and 58 in Appendix A.2; and D, see Figures 59 and 60 in Appendix A.3) describe alarm management systems for two kinds of production machines, which support operators by preventing alarm floods and finding their root-causes. In use-case C, approximately 500,000 alarms are generated per year of operation with 200 distinct types of alarms for each production machine. The alarm management system of use-case C is hosted in a public cloud by the OEM of the machine, which offers additional diagnostic services. Therefore, the alarm messages of several hundreds of these machines, scattered over multiple customers and production sites, have to be transferred to the public cloud. In use-case D, the hosting of the alarm management systems follows a hybrid approach with both private and public clouds. Customers can analyze data in their private cloud to ensure privacy. As an additional service, the OEM offers to combine data with datasets from similar machines to improve the quality of the analysis. One machine generates between 3,600,000 and 6,000,000 alarm messages per year, with there being approximately 40 machines per customer and production site. A total of 500 distinct alarms exist. Several customers connect their own private clouds with the public cloud offered by the OEM. [TWV20]

7.2.3. Results of the Expert Evaluation

First, the completeness of the graphical notation and its elements was evaluated for both viewpoints. The experts pointed out that all relevant information could be captured and structured using the notation. Both the system architecture, with its hardware and software elements (*Req-*M_{Sys}), as well as the data flow through the system (*Req-*M_{DF}), could be expressed and structured. The differentiation between hardware devices and software functionality that is executed on this hardware in the system viewpoint was considered as extremely helpful to structure the system. The same is valid for the data flow view, where the distinction between the types of data handling (*Source*, *Sink*, *Transducer*, and *SinkSource*) is useful to follow the flow of data. The data can easily be traced through the associated hardware and software systems via the combination of the two views. This separation of concerns greatly reduces the complexity when designing and sizing data collection architectures for all involved parties. Furthermore, the number of different constructs and symbols is relatively low and makes the notation manageable for different expert groups (*Req-*M_{Graph}). [TWV20]

Concerning the annotations as an essential part of the graphical notation during the specification and design of system architectures, the expert opinion was positive as well (*Req-*$M_{PropReq}$). Especially for complex connected production cells and robots, latency requirements or protocol constraints can easily be structured and exchanged. The experts considered the categorizing of annotations (time, architecture, and data) as helpful to separate concerns. Minor concerns were related

to the absolute number of symbols, especially in large data flow views. Here, the number of annotations can be huge in a confined space. Grouping of annotations and references to multiple data streams or nodes in the data flow could be considered for future versions of the notation. Furthermore, an interactive graphical editor may overcome this limitation as visibility of elements could be adjusted on-the-fly to provide experts only with the needed information. This would, for instance, include separate modeling views with only relevant model elements selectable and autocompleting of unique names, properties, and requirements. Furthermore, the connectable model elements could be highlighted if a network or an annotation is selected inside the editor. The editor, therefore, would support experts with an improved and simplified workflow. [TWV20]

Especially the distribution of information across separate viewpoints and multiple sheets was highly appreciated. The information is efficiently distributed and grouped to manage the density and amount of information per sheet. Capturing all flows of data in large systems proved to be challenging in one diagram. Grouping of these flows in sub-views on separate sheets limited the overall complexity. The means provided for integrating the two viewpoints and different sheets were considered sufficient and intuitive by the experts. Still, an integrated editor for the DSL, as well as an automatic synchronization between the model instance and its graphical representation, is currently lacking (*Req-*M_{Graph}). [TWV20]

Experts had no problem differentiating between the distinct types of elements and annotations. Additionally, utilization of the same family of shapes for the specification of properties and requirements was pointed out as helpful without compromising the perceptual discriminability (*Req-*M_{Graph}). [TWV20]

In summary, the graphical notation is a powerful approach to structure information during the engineering and operational phases of CPSoS. In contrast to existing approaches, the notation can capture information from the operational technology as well as the IT domains. It contains constructs for combined hardware and software architecture as well as the stream of data through all connected systems on different levels of the system hierarchy. [TWV20]

7.3. Lab-scale Feasibility Study

Based on the architecture concept and the developed DSL, a model-driven generation of the communication architecture of data collection architectures can be carried out. In the following Section, the results of a reasonably complex, lab-scale feasibility study will be presented and discussed. The case-study will be implemented twice: once using a classical, manual programming approach and once with the model-driven approach developed in this work. After an introduction to the experimental setup, the graphical models of the case-study will be presented. Afterward, the

model-driven generation step and the deployment of the components is discussed. The Section closes with a comparison of the implementation efforts for initial deployment and redeployment between the model-driven and the classical approaches.

7.3.1. Experimental Setup

The experimental system consists of several lab-scale production plants and connected systems for data analysis and visualization. The UML deployment diagram in Figure 46 gives an overview of all relevant systems. It should be noted that the diagram includes all systems that must be interfaced, but not the architecture's technical realization, including infrastructure components and data adapters needed for the implementation. The experimental setup aims to describe a sufficiently complex and representative scenario for the evaluation of the developed, model-driven approach. Therefore, it includes automated production systems, further legacy systems, as well as newly implemented greenfield systems (*Req-A$_{POP}$*). Furthermore, the systems are part of different networks, all connected to each other. The systems will be introduced in the following.

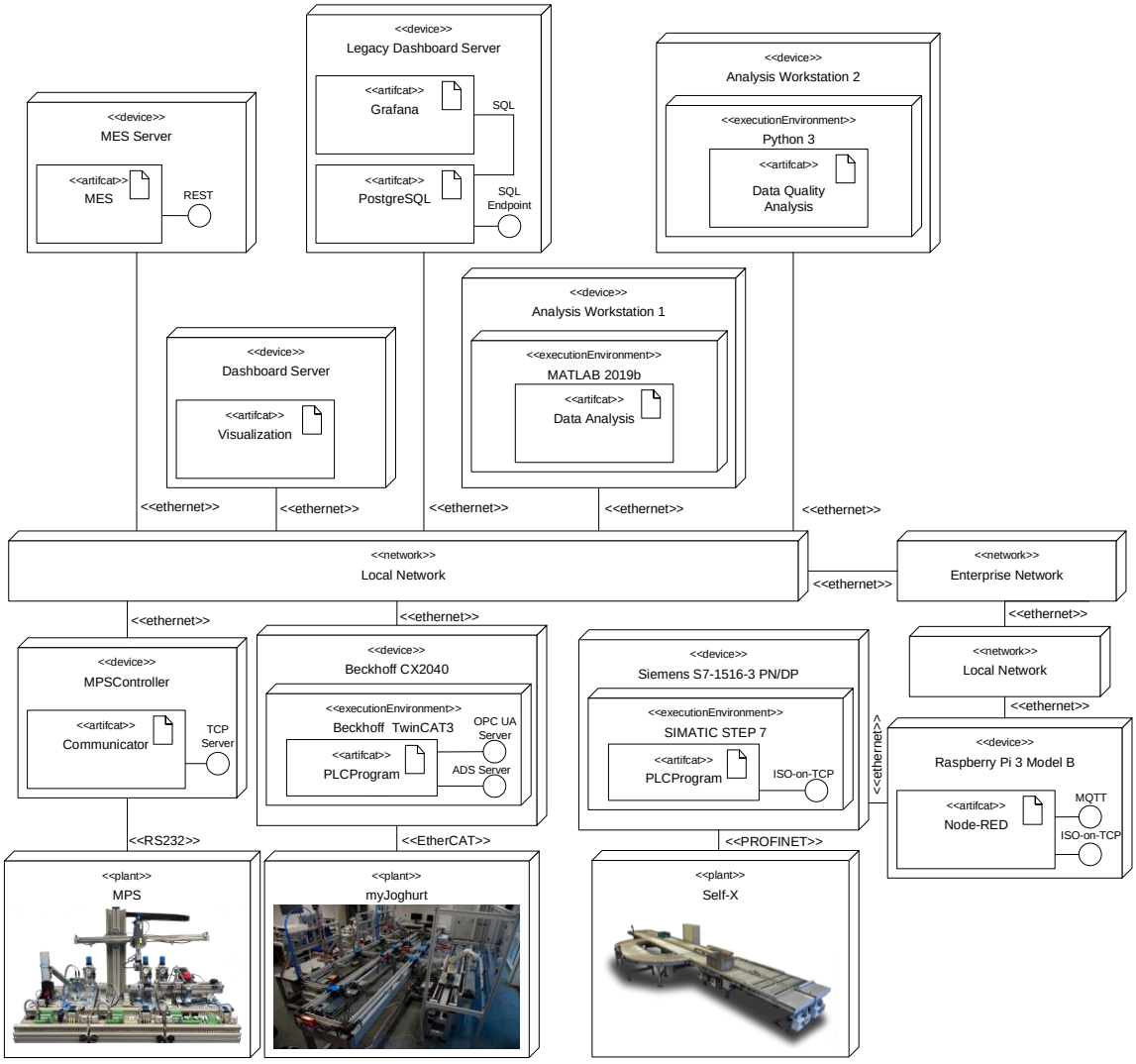


Figure 46: Systems in the lab-scale feasibility study without gateways and infrastructure components.

Modular Production System (legacy system)

Modular Production Systems (*MPS*) [Fes20] are a series of production modules for research and teaching manufactured by the German company Festo. The systems represent typical applications of automation components. In this work, a distribution station with a stack magazine and a pick and place unit is interfaced. The station features a total of 32 binary I/Os all connected to a so-called EasyPort [Fes08]. The EasyPort allows access to these I/Os via a serial RS232 connection with a proprietary protocol. An additional computer (*MPSController*) hosts an application that communicates with the EasyPort using this proprietary protocol and executes the corresponding logic to control the MPS plant. The application was developed for lecture purposes and not to allow access to the plant, yet it allows to read/write all I/Os over a direct TCP connection. The protocol of the TCP connection is another proprietary protocol tailor-made for the specific application. Therefore, the system represents a legacy system, with closed interfaces that can only be interfaced via a retrofitted, external data adapter.

myJoghurt (evolving, retrofitted plant)

The myJoghurt production plant [Ins13; Ins20; May+13; Vog+14d] is one of the key research demonstrators of the Institute of Automation and Information System at the Technical University of Munich. The plant serves as an Industrie 4.0 demonstration platform and simulates the manufacturing of individualized yogurt with a lot size one. It consists of three plant sections: a storage system with a 5-axis Mitsubishi handling robot mounted on an additional, linear axis; a process technology part for batch manufacturing and the two associated filling stations, each with one tank for liquid and two silos for solid material; and a material handling system with multiple switches and conveyors that transports products between storage system and filling stations. Through the combination of discrete and batch production processes, the plant resembles an overall hybrid production process with an interface between batch and discrete processes at the filling stations. The plant is controlled by a central Beckhoff CX2040 PLC running on TwinCAT3 [Bec19d]. All ten switches and 22 conveyor drives are directly connected to the PLC over an EtherCAT bus. Besides, the process technology part, several barcode scanners mounted next to the conveyors, and the robot are interfaced over Profibus. The Profibus master terminal is not mounted directly on the PLC but on the first bus coupler inside the EtherCAT bus. In total, around 250 I/Os, as well as 250 internal variables, are part of the control system and can be accessed using ADS or OPC UA. The plant represents a typical, evolving plant that is retrofitted and updated over time: new devices are introduced into the system, and control hardware is replaced, while existing devices remain part of the plant.

Self-X Material Flow Demonstrator

The Self-X material flow demonstrator [Aic18] is a research testbed related to modular software development and self-x functionalities in the domain of intralogistics. It features two roller-driven tracks as well as a bend and T-junctions to enable material flow. A Siemens S7-1516-3 PN/DP PLC is the central control unit of the plant and connected to the distributed I/Os over Profinet (167 in- and outputs in total). For data access, the system is retrofitted with an external gateway. A Raspberry Pi 3 Model B hosts Node-RED [JS 19], a visual, browser-based programming environment for IoT applications. Node-RED includes basic functionalities to get, process, and forward data, but also provides a flexible extension mechanism for loading additional functionalities from user packages. One of these functionalities is the support for the S7 ISO-on-TCP protocol [RFC1006]. Data is transferred from the PLC to the gateway, processed, and then forwarded via MQTT. The Self-X plant is not connected to a larger network, but instead directly attached to the gateway over Ethernet. Therefore, the gateway decouples the plant from the superordinate Ethernet network, to which the gateway is connected.

Other Systems

Furthermore, the case-study includes several additional systems that are introduced briefly in the following:

- a sample MES functionality is part of the system. It provides recipes, order data, and progress reports associated with the manufacturing process in the myJoghurt plant over a REST interface;
- a greenfield dashboard that is developed in the course of the architecture implementation. It should be used to monitor the operation of the myJoghurt plant;
- an additional legacy dashboard based on Grafana [Gra20], a browser-based dashboard for visualization of data. Grafana is an open-source, widely adopted, easy-to-use graphical dashboard used in various industrial applications. The Grafana instance is configured to use a PostgreSQL database [Pos19] hosted on the same machine for data storage. Data can be visualized in the dashboard by storing it in the database;
- a data analysis workstation based on MATLAB 2019b, as MATLAB is an accepted application in the engineering domain [Mat19]; and
- an additional data analysis workstation based on Python 3 representing a typical environment for industrial data analysis [Pyt19].

These additional systems, as well as the system that will be implemented as infrastructure or data adapter systems, are hosted on Raspberry Pis with Debian Buster [Deb19] or computers / virtual machines with Windows 10 / Ubuntu Linux 18.04.3 LTS [Can18].

7.3.2. Graphical Model of the Lab-scale Architecture

Based on the experimental setup discussed above, a suitable data collection is conceptualized and modeled using the developed DSL. The AMQP protocol was chosen as a suitable protocol for the initial realization of the data collection architecture. With an installed RabbitMQ broker, it provides good scalability as well as support for advanced QoS features that may be necessary for the future. The respective graphical models can be found in Appendix B. Here, Figures 61 to 63 contain the system diagrams, while Figures 64 to 67 display the corresponding data flow diagrams.

All infrastructure systems can be found in the system diagram in Figure 63. One of the systems (*rabbitmq*) hosts an instance of the mentioned RabbitMQ broker as a central communication backbone of the data collection architecture. Furthermore, an additional Mosquitto broker is part of the system (*mosquitto*) that can accept and forward the data from the gateway of the Self-X plant over MQTT. The systems *Worker0* to *Worker3* provide data translation functionalities used to translate the data formats between the connected heterogeneous systems. All four systems are based on a cheap Raspberry Pi, Debian Linux, and Docker. Due to the missing support of the Beckhoff ADS library for the .NET Core 3.1 framework, the data adapter for the myJoghurt plant is hosted on a separate Windows machine (*myJoghurtAdap*). Here, the data translator uses the gRPC interface to connect the ADS functionality to the architecture.

The data from the myJoghurt production plant, as well as the order data from the MES, is sent to the Python-based *DA2* analyzer. Here, the first analysis functionality (*MESDA*) calculates KPIs based on the production and MES data. While the greenfield dashboard (*Viewer.HMI*) receives all data, including the KPIs, only the KPIs are forwarded to the Grafana dashboard and the MES database. Therefore, the data flow is split at *DA2.MESDA*, which acts as a *Transducer* for the data flow with destination greenfield dashboard, and as a *SinkSource* for the other flow. In parallel, the second analyzer functionality (*DA2.DriveCM*) calculates the probability of an anomaly for the two monitored servo drives of the myJoghurt plant. The results are only sent to the data translator related to the greenfield dashboard (*Worker3.FDA*) and are not available to any other system.

A second anomaly monitoring functionality is available in *DA1.DA* as a MATLAB program. The analyzer monitors the condition of the MPS plant. It calculates timings of typical actions, counts them, and provides an additional anomaly score. All data, including this, is collected by the data collection architecture and sent to the Grafana dashboard for visualization and the MES database

for long-term storage. The only exceptions are the results of the drive condition monitoring mentioned above that are only available in the greenfield dashboard.

The modeling of all involved systems and data flows was carried out by three persons. The time efforts for the modeling are summarized in Table 17 with a description of the profiles of the participants. Therefore, as a baseline, the modeling effort is assumed as the maximum measured effort of 4 h 40 min (4.66 h) in the following. This value includes time for adjusting the layout of the diagrams as well as checking the consistency of the models.

Table 17: Modeling efforts for modeling the lab-scale case-study of three persons and their experience with the notation and background in industrial automation.

Person	Experience level	Total effort for modeling of the lab-scale setup
1	Well-experienced user, strong industrial automation background, applied the graphical notation several times.	2 h 30 min
2	Semi-experienced user, medium industrial automation background, applied the notation occasionally.	4 h 20 min
3	Inexperienced user, strong industrial automation background, recently introduced to the notation.	**4 h 40 min**

7.3.3. Model-driven Generation of the Communication Architecture

After the creation of the model instance, the model-driven generation of the communication architecture using AMQP as the standard communication protocol is executed. The model-to-text transformation step included the generation of the pre-configured communication architecture as C# code (*Req-*G_{Com}), the setup of Visual Studio 2019 project files, related Docker configuration files for the generation of individual containers per software functionality, and the configuration for the RabbitMQ broker. In total, 4284 lines of C# code were generated, with an additional 616 lines of configuration and project files.

Subsequently, the application-specific logic was manually implemented to yield the prototype of the data collection architecture (cf. Figure 32). This code included the internal analyzer and dashboard functionalities which process the communicated data. Furthermore, credentials were updated by hand, as they should not be part of the models due to security reasons. In the next step, the individual software functionalities were compiled and deployed. While all worker functionalities (*Worker0* to *Worker3,* cp. Section 7.3.2 and Figure 63) were deployed as Docker containers to the respective Raspberry Pis using the CI pipeline (cf. Section 6.3), the other functionalities were manually copied to the respective systems and started.

The data flows in the deployed data collection architecture were examined for all modeled systems. As expected, the modeled data flows were correctly set up and working. With the dashboard, it was possible to monitor the operation of the connected plants and the flow of data. This includes, besides the newly implemented systems, data from the existing legacy systems. These systems were successfully interfaced and integrated into the data collection architecture (*Req-SF_{Leg}*). In addition, the existing control hierarchy was unaffected as the data collection architecture was implemented in parallel to the existing systems. All plants were correctly working, and the control interaction was not conducted over the architecture (*Req-A_{POP}*).

In the next step, the created models were modified. Instead of AMQP, the usage of the MQTT protocol was specified as MQTT is a lightweight protocol especially suitable for IoT applications. The update of the respective annotations took a total of 20 minutes. The model transformation step for the creation of the preconfigured data collection architecture was executed again for the modified models (*Req-G_{Com}*). This resulted in a total of 4284 newly generated lines of C# code, with an additional 288 lines of configuration and project files. As the code sections with the manually programmed application-specific code are marked as protected, they were not overwritten in the code generation step. Furthermore, due to the defined programming interfaces and the abstraction of the protocol-specific aspects in the underlying software framework, no additional modifications besides the updating of the credentials were necessary. Therefore, this application-specific code can remain unchanged while still being functional (*Req-SF_{API}*, *Req-SF_{ACP}*).

After a new compilation step and a redeployment, the data collection architecture was again correctly running and working. This time based on the MQTT protocol instead of the AMQP.

7.3.4. Effort Metrics for Deployment and Redeployment

Based on the results of the model-driven generation of the data collection architecture in the previous Section, a comparison of implementation efforts between model-driven and classical, manual programming approach is performed. This comparison should answer the question if a model-driven generation decreases the implementation efforts for initial deployment (*Req-A_{Dep}*) and migration of such architectures (*Req-A_{ReDep}*).

Therefore, the data collection architecture was additionally implemented manually. The implementation considered all systems and data flows as previously modeled. The aim was to replicate the model-driven architecture using the AMQP protocol for communication as far as possible. In total, 989 lines of C# code were manually implemented (LoC_{Total}). This excludes the implementation effort for the application-specific logic as these are also not part of the model-driven gener-

ation. All code was programmed following the same programming style and in avoidance of writing unnecessary code (e.g., for future extensions, increased modularity, or better exception handling).

For a comparison of the programming effort, measured in lines of code (LoC), and the effort using the model-driven approach, a common effort metric has to be found. The first possibility is a direct comparison of the lines of code between the output of the model-driven and the manually implemented approaches. Yet, this is not a suitable comparison, as the timely efforts per line of code differ greatly. Furthermore, the automatically generated code does not follow the same coding conventions as the manually implemented code and includes additional code portions targeted to increase the modularization of the code.

Another possibility would be a conversion factor for every hour of modeling to lines of code. However, as this is highly dependent on the DSL, the underlying use-case, and the model-transformation step, no representative figures can be found in the literature. As an alternative, figures related to the productivity of an experienced programmer can be used. Several studies can be found in the literature that investigate different aspects of programming productivity during the last decades. While many studies focus on outdated programming languages, some recent publications can be found. For instance, Cusumano et al. [Cus+03] give an average productivity of 436 LoC per month and programmer in Europe. They investigated a total of 104 large-scale software projects worldwide from leading software companies written with different programming languages. This productivity corresponds to approximately 2.5 LoC/h (4.33 weeks per month, five working days per week, and eight hours work per day), but includes related tasks, such as code testing and code reviews. Alternative figures can be found in a study from Prechelt [Pre00]. The study compares the implementation of a small-scale program by individual programmers in different programming languages, including Java, which is quite similar to C#. Prechelt gives a median productivity of 22 LoC/h for Java, and a productivity of 36 LoC/h for the upper quartile (75^{th} percentile, including 75% of the observations). These figures are significantly higher than the productivities measured by Cusumano et al. [Cus+03]. Possible reasons are the smaller scale of the project and the implementation by just single programmers, which limits the productivity loss caused by increased documentation and communication efforts. They represent very high productivities which are not commonly found in industry, but on the other hand, the absolute maximum productivity of a programmer. Therefore, productivity measured at the upper quartile $p_{LoC} = 36\,\mathrm{LoC/h}$ will serve as a basis for a conservative comparison of the implementation efforts between a manual implementation and the model-driven approach. With p_{LoC}, the implementation effort $E_{Classical}$ for a manual programming can be calculated to

$$E_{Classical} = \frac{LoC_{Total}}{p_{LoC}} = \frac{989\ \text{LoC}}{36\ \frac{\text{LoC}}{\text{h}}} = 27.47\ \text{h}. \quad (1)$$

In comparison, the implementation effort for the model-driven approach $E_{Model-driven}$ corresponds to the time for modeling of the system, therefore

$$E_{Model-driven} = 4.66\ \text{h}. \quad (2)$$

The relative effort between both implementations without the initial creation of the model-driven toolchain, can, consequently, be accounted to

$$\frac{E_{Model-driven}}{E_{Classical}} = 16.96\%. \quad (3)$$

The effort is accordingly significantly reduced by the model-driven approach, which simplifies the implementation of data collection architectures (*Req-A*$_{Dep}$). The effort for the initial creation of the toolchain will be considered in one of the extrapolation case-studies in Section 7.5.

In case of a migration from one communication protocol to another (in this example from AMQP to MQTT), the manual implementation must undergo a partial reimplementation. This includes an adaption of all communication functionalities and also possibly modifications to the application-specific logic due to interface incompatibilities. The respective number of modified lines of codes will be in the magnitude of the AMQP-based implementation.

On the other hand, using the model-driven approach, the models have to be updated. Furthermore, some single lines of the newly generated code that contain the credentials of the respective systems have to be modified after the execution of the code generation step (approximately 15 in total for the use-case). This results in a total effort of approximately 45 minutes. The application-specific code can remain unchanged as the same interfaces are supported as before the migration.

Therefore, a model-driven data collection architecture can also reduce the implementation effort for migration in comparison to manual migration (*Req-A*$_{ReDep}$). Under the assumption that a partial reimplementation of the architecture for MQTT accounts for 60% of the initial effort (593 LoC in total), the relative effort is significantly reduced to 4.55%.

7.4. Industrial Case-Study

The model-driven generation of the communication architecture is applied to an industrial case-study to evaluate its scalability. Therefore, Case-Study A from Section 7.2.1 is used as a basis for the model-driven approach.

Based on the models, a total of 2906 lines of C# code were generated. Additionally, the model transformation process produced another 110 lines of configuration, project, and docker files. Manual checking of the code verified the correct generation of the preconfigured data collection architecture (*Req-*G_{Com}). No suitable hardware, as defined in the models, was available for the scalability tests. Therefore, simple direct forwarding functionalities substituted the missing application-specific implementation of the internal logic of all systems. The data collection architecture was then deployed and executed in docker containers to verify its proper functioning. All modeled data flows were correctly working as specified, and data was flowing through the system.

It can be concluded that the model-driven approach for the development of data collection architectures is also applicable to the industrial use-case. Moreover, the industrial case-study manifested the representativeness of the lab-scale feasibility study as the number of considered systems, as well as the generated lines of code, are significantly greater for this scenario.

7.5. Effort Extrapolation Case-Study

The case-studies in Sections 7.3 (lab-scale feasibility study) and 7.4 (industrial case-study) provided insights into the model-driven generation of the communication architecture. They included an analysis of the feasibility and scalability of the approach, as well as a basic implementation effort comparison for a specific use-case. Nevertheless, an answer to the question of whether the model-driven approach can decrease implementation efforts for a broader range of use-cases could not be given. Therefore, an extrapolation case-study will be presented in the following.

The case-study is based on minimal publisher/subscriber pairs with an adjustable number of transported variables between both systems. Figure 47 gives the models of the smallest possible system with one pair and one communicated variable. The publisher is a PC with a legacy software functionality that generates one variable named *TestByte*. This data is routed over a middleware functionality on an additional computer and forwarded to the subscriber. The protocol specification in Figure 47 is a placeholder as various protocols will be considered.

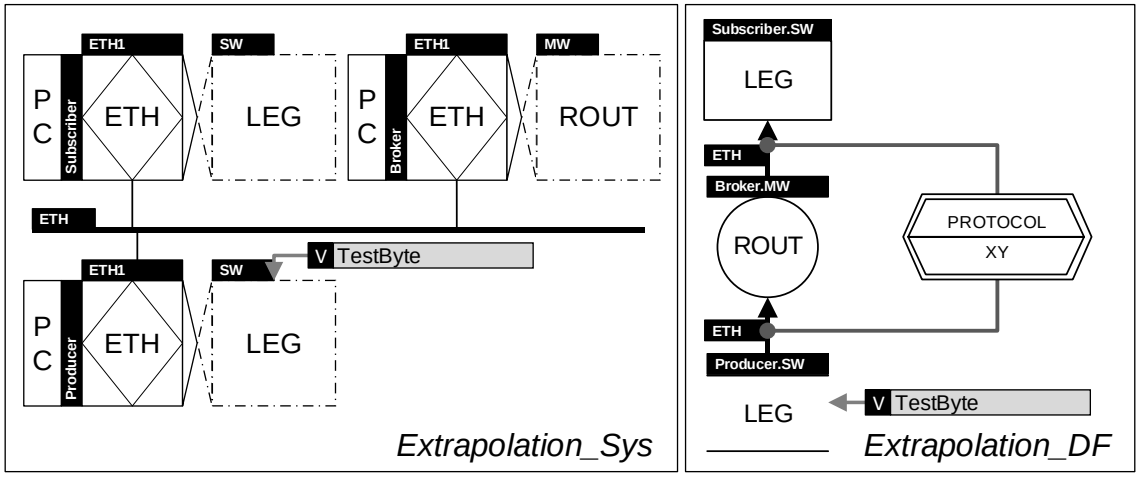


Figure 47: System and data flow of the minimal extrapolation use-case modeled with the graphical modeling notation. The protocol specification (requirement) as a placeholder labeled with XY is to be replaced by the specific protocol.

As a baseline, the described minimal system was implemented manually in C# using the technology-specific programming libraries for AMQP, Apache Kafka, Beckhoff ADS, MQTT, and OPC UA. The code was developed with a focus on decoupling the communication functionalities from the main functionality of the legacy program (sending/receiving of data). This decoupling was done to improve reusability in case of migrations between communication technologies. All code samples for the minimal producers and subscribers can be found in Listings 1 to 10 in Appendix C. Lines of code metrics for all clients were collected to calculate the average effort for implementing a minimal producer and subscriber pair. For the case-study, it was assumed that the line of code, where the communication functionality is instantiated and configured with address, port, and credentials (`var client = …`), must be changed by hand in both manual coding and model-driven generation. Hence, this line of code is not accounted for, and all raw lines of code results are decremented by one. Afterward, all Listings were analyzed concerning the migration between communication technologies. Due to the decoupling mentioned above, some parts of the programs can remain unchanged during migration. Therefore, only modified lines were counted. Table 18 summarizes the programmed, corrected lines of code per protocol for initial deployment, and migration between communication technologies. Furthermore, it includes the respective mean lines of code across all considered protocols.

Table 18: Manually programmed lines of code (LoC) for minimal producer and subscriber functionalities. The corresponding Listings can be found in Appendix C.

Protocol	Total lines of Code (LoC) without user name and password for an initial deployment			Total lines of Code (LoC) without user name and password for a migration		
	Producer	Subscriber	Producer/ subscriber pair	Producer	Subscriber	Producer/ subscriber pair
AMQP	27	42	69	19	29	48
Beckhoff ADS	29	54	83	21	41	62
Apache Kafka	26	50	76	18	37	55
MQTT	20	33	53	12	20	32
OPC UA	52	64	116	44	51	95
MEAN	30.8	48.6	79.4	22.8	35.6	58.4

The total, average lines of code can be divided into lines of code for the programming of a representative producer/consumer pair i (LoC_{Pair_i}) and the additional lines of code per communicated variable j between the two systems ($LoC_{Var_{i,j}}$, 2 for an initial deployment, 0 for a migration). Therefore, the total lines of code of a project LoC_{Total} can be calculated with the number of pairs N_{Pair} and the number of variables N_{Var_i} per pair i to

$$LoC_{Total} = \sum_{i=1}^{N_{Pair}} \left(LoC_{Pair_i} + \sum_{j=1}^{N_{Var_i}} \left(LoC_{Var_{i,j}} \right) \right). \quad (4)$$

Both, N_{Pair} and N_{Var_i}, must be integers larger than 0. With the programming productivity of $p_{LoC} = 36$ LoC/h [Pre00] from Section 7.3.4, the implementation effort $E_{Classical}$ for a manual programming can be calculated as

$$E_{Classical} = \frac{LoC_{Total}}{p_{LoC}}. \quad (5)$$

In Table 19, the average efforts in lines of code per producer/subscriber pair based on the results across all considered protocols from Table 18, as well as the programming efforts based on the productivity, are summarized. These will be used in the extrapolation case-study as representative implementation efforts for manual programming

Table 19: Effort in lines of codes and programming time for manual implementation of minimal producer/subscriber pairs.

Symbol	Description	Initial deployment	Migration
LoC_{Pair_i}	Average lines of code per producer/subscriber pair i	77.4	58.4
$LoC_{Var_{i,j}}$	Average lines of code per variable j in producer/subscriber pair i	2	0
$E_{P,Pair_i}$	Average programming effort per producer/subscriber pair i	2 h 8 min 20 s	1 h 37 min 20 s
$E_{P,Var_{i,j}}$	Average programming effort per variable j in producer/subscriber pair i	3 min 15 s	0 s

On the other hand, the effort $E_{Model-driven}$ using the model-driven approach is influenced by two factors:

- first, an initial effort $E_{Toolchain}$ for the creation of the DSL, the model-driven generation of the communication architecture, and the underlying software framework. This initial effort includes the implementation of all protocols from Table 18; and
- second, a variable modeling effort $E_{Modeling}$ summing up the modeling efforts for every element to be modeled. For the effort extrapolation case-study, $E_{Modeling}$ can be expressed as

$$E_{Modeling} = \sum_{i=1}^{N_{Pair}} \left(E_{M,Pair_i} + \sum_{j=1}^{N_{Var_i}} \left(E_{M,Var_{i,j}} \right) \right), \quad (6)$$

with $E_{M,Pair_i}$ the modeling effort per producer/subscriber pair i and $E_{M,Var_{i,j}}$ the modeling effort for each variable j per pair i. The figures listed in Table 20 were measured for an experienced engineer assuming an automatic synchronization of the graphical model and the metamodel instance. All measurements were rounded up to full minutes. Here, the efforts are independent of the underlying communication protocol.

Table 20: Effort in time for modeling minimal producer/subscriber pairs.

Symbol	Description	Initial deployment	Migration
$E_{M,Pair_i}$	Modeling effort per producer/subscriber pair i	10 min	1 min
$E_{M,Var_{i,j}}$	Modeling effort per variable j in producer/subscriber pair i	1 min	0 min

Therefore, $E_{Model-driven}$ is expressed as

$$E_{Model-driven} = E_{Toolchain} + E_{Modeling}$$
$$= E_{Toolchain} + \sum_{i=1}^{N_{Pair}} \left(E_{M,Pair_i} + \sum_{j=1}^{N_{Var_i}} \left(E_{M,Var_{i,j}} \right) \right). \quad (7)$$

Under the assumption that $E_{Toolchain}$ is only relevant for the first implementation of the approach, the total effort of the model-driven approach $E_{Model-driven}$ for all subsequent realizations equals the modeling effort $E_{Modeling}$.

In the following, three scenarios are discussed:

- a comparison of the implementation efforts for a classical, manual programming and the model-driven approach for an initial deployment of a data collection architecture under the assumption that the toolchain already exists;
- afterward, a migration of an existing architecture realization from one communication technology to another using both approaches under the assumption that the toolchain already exists; and
- an estimation of the necessary number of producer/subscriber pairs and variables for a realization of the architecture taking the effort for the toolchain creation into account.

7.5.1. Initial Deployment

The implementation efforts for an initial implementation of an average data collection architecture based on a classical, manual programming approach and the model-driven approach are given in Figure 48. For this and all following figures of this section, the number of variables N_{Var_i} per publisher/subscriber pair i is expressed as the average number of variables $\overline{N_{Var_i}}$ per pair i. As can be seen from the figure, the implementation effort for the classical approach is significantly higher than for the model-driven approach. Both surfaces show the influence of an increasing number of pairs as well as variables per pair, with a higher sensitivity towards the number of pairs.

The relative effort between model-driven and classical approach can be expressed as

$$\frac{E_{Model-driven}}{E_{Classical}} = \frac{0.1667 + 0.0167 \cdot \overline{N_{Var_i}}}{2.1500 + 0.0556 \cdot \overline{N_{Var_i}}}. \quad (8)$$

Therefore, the relative effort is independent of the number of pairs N_{Pair}. The relative effort as a function of the average number of variables per pair is plotted in Figure 49. For large systems, the relative effort tends to converge to a value of about 30%.

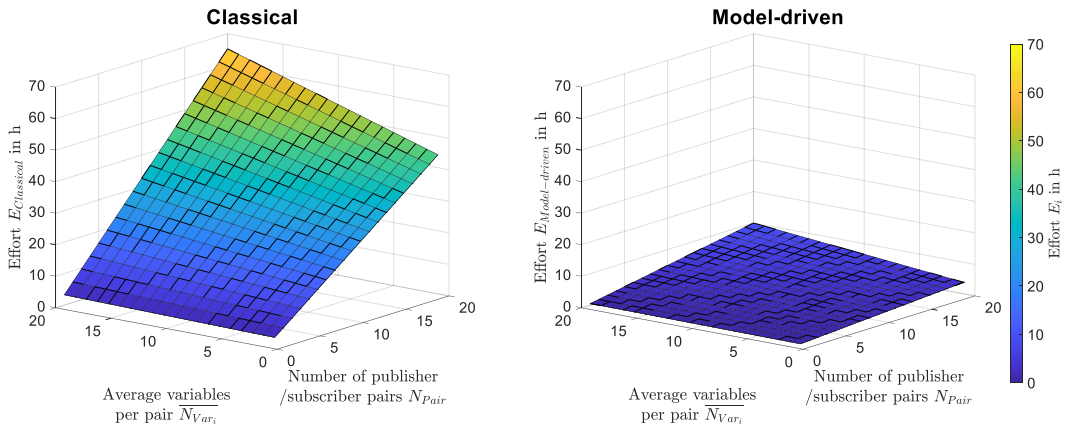


Figure 48: Comparison of implementation efforts for initial deployment as a function of the number of publisher/subscriber pairs and the average number of variables per pair. Classical, manual programming (left), and model-driven approach (right).

On the other hand, the smaller the number of variables per system is, the more advantageous it is to use the proposed approach. This observation can be explained by the significant overhead of on average 77.4 lines of code for the creation and instantiation of the relevant communication libraries in manual programming. Adding additional variables to an already instantiated communication channel between publisher and subscriber adds only two additional lines to this existing codebase. In comparison, the modeling of small systems tends to be significantly faster than programming them. Furthermore, also the modeling of additional variables causes less effort compared to manual programming, but higher relative effort compared to the instantiation of the communication.

Therefore, under the assumption that the toolchain exists and can be used out of the box, the model-driven approach for the initial generation of the communication architecture significantly outperforms the classical, manual programming approach in terms of implementation effort (*Req-A_{Dep}*). Relative implementation efforts are in the range of 8% to 30% depending on the size of the system.

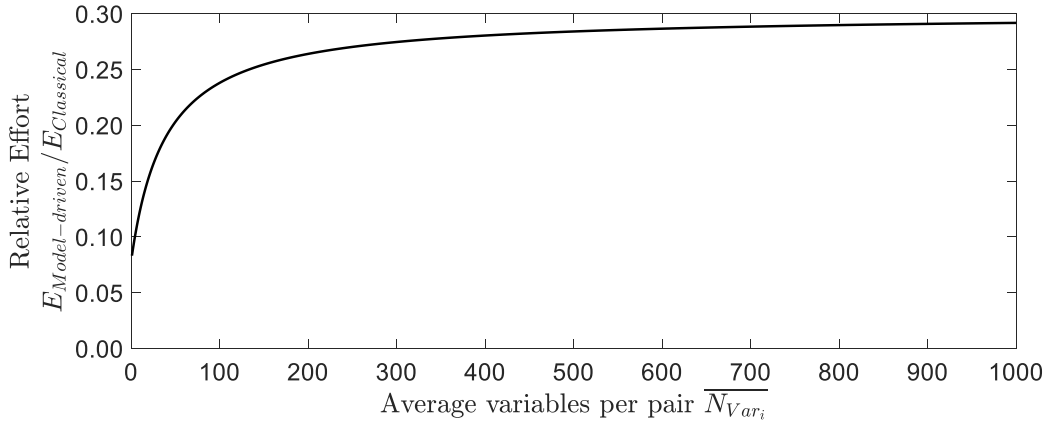


Figure 49: Relative effort between model-driven approach and classical, manual programming for initial deployment as a function of the average number of variables per pair.

7.5.2. Migration

In the case of migration between two communication protocols (redeployment), only several lines of code have to be modified. Due to the modular structure of the code templates, it is not necessary to change any variable-related code. On the other hand, the change of the communication protocol requires only the modification of a single annotation label in the DSL per pair of producer/subscriber (*Req-SF*$_{ACP}$). Additionally, no further actions are needed for the variables. Therefore, only the number of pairs to be migrated are relevant for the implementation efforts in case of a migration. Figure 50 gives the implementation efforts for both cases. In comparison to the initial deployment, the effort ratio between model-driven and classical approach is further decreased to

$$\frac{E_{Model-driven}}{E_{Classical}} = 0.01, \tag{9}$$

independent from the number of pairs N_{Pair} and the average number of variables per pair $\overline{N_{Var_i}}$. Therefore, also implementation efforts for redeployment are significantly decreased through the model-driven approach (*Req-A*$_{ReDep}$).

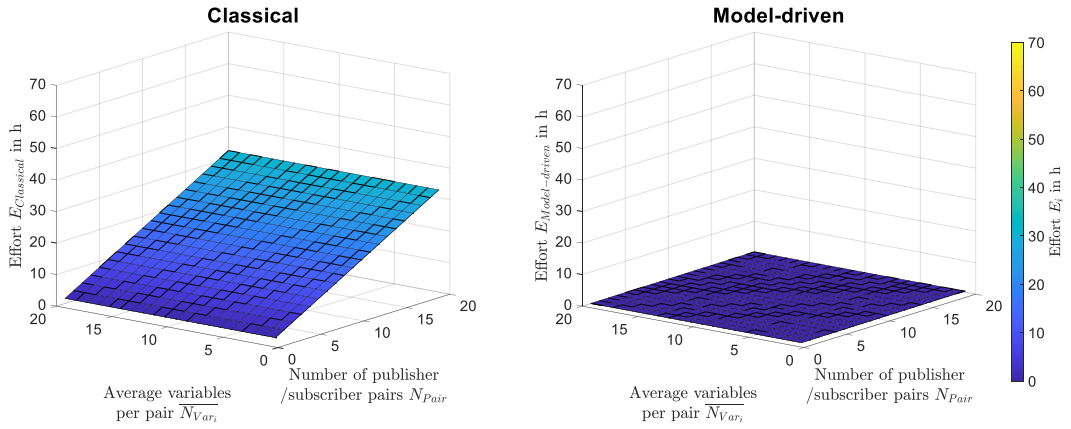


Figure 50: Comparison of implementation efforts for a migration scenario as a function of the number of publisher/subscriber pairs and the average number of variables per pair. Classical, manual programming (left), and model-driven approach (right).

7.5.3. Estimation of Necessary System Sizes for Break-even

The last part of this section is dedicated to the estimation of the minimal system sizes that make the development of the toolchain for a model-driven generation of data collection architectures feasible. Therefore, Table 21 lists the relevant code sizes and efforts that influence the effort for the first creation of the toolchain $E_{Toolchain}$. The effort for the development of the DSL, including the metamodel and the graphical notation, was estimated to a full person-year (twelve months per year, 4.33 weeks per month, with five working days per weeks, and eight hours working time per day). The total lines of code in the software framework sum up to 4000 lines. The transformation logic contains a total of 1350 lines of code, mainly written in the Acceleo transformation language,

but also containing small snippets of Java and C# code. Taking into account the programming productivity P_{LoC}, the total effort for the creation of the toolchain was estimated to 2227 h.

The discussed figures do not include the continuous maintenance of the developed code basis, nor a sophisticated test-driven development. Both efforts would also have to be taken for classical, manual implementations of the communication architecture. Therefore, in the following, it is assumed that they do not influence the estimation of the break-even.

Table 21: Efforts and lines of code for the creation of the toolchain for model-driven generation of communication architectures. Effort for the development of the DSL was estimated. Programming efforts for the software framework and the transformation logic based on the productivity p_{LoC} [Pre00] and the assumption that p_{LoC} is also valid for Acceleo code.

Symbol	Description	Lines of code / Effort
E_{DSL}	The effort for the development of the graphical notation and the underlying metamodel for the DSL	2078 h (one person-year)
LoC_{SF}	Lines of code in the software framework (C#)	4000
E_{SF}	The effort for the development of the software framework based on p_{LoC}	111 h
LoC_{M}	Lines of code in the transformation logic (Acceleo/Java/C#)	1350
E_{M}	The effort for the development of the code generation logic based on p_{LoC}	38 h
$E_{Toolchain}$	Total effort for the creation of the toolchain for model-driven generation of communication architectures	2227 h

Taking the initial effort for the creation of the model-driven approach into account, the efforts for the model-driven implementation of data collection architectures are severely impacted (see Figure 51). Especially for small systems, the classical approach is superior and should be preferred. Nevertheless, gradients in both dimensions are significantly smaller for the model-driven approach than the classical. Based on these observations, the questions where the break-even between the efforts can be found should be answered in the following. In other words, what minimal system size (as a function of producer/subscriber pairs N_{Pair} and the average number of variables per pair $\overline{N_{Var_i}}$) is needed until the model-driven approach can outperform classical software development.

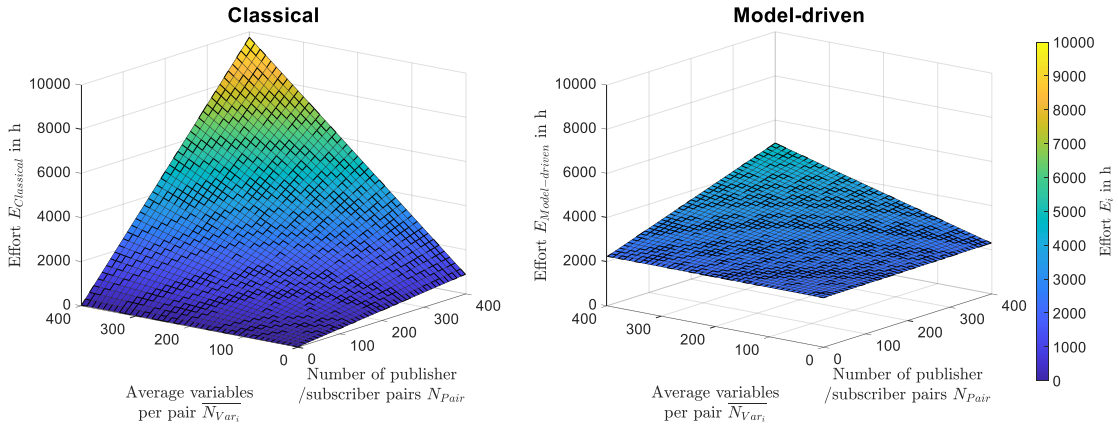


Figure 51: Comparison of implementation efforts for an initial deployment, including the effort for the creation of the model-driven toolchain as a function of the number of publisher/subscriber pairs and the average number of variables per pair. Classical, manual programming (left), and model-driven approach (right).

Therefore, the relative efforts over a wide range of pairs and the average number of variables are plotted in Figure 52. The Figure includes two scenarios: the first captures only the initial deployment, while the second includes one protocol migration for all systems. At first, the results for an initial deployment without any migrations are discussed.

For small systems in the size of the scenarios considered earlier in this Section, the model-driven approach requires about a hundred, up to a thousand times higher efforts compared to the classical implementation. Nevertheless, the more systems are modeled, and the more variables they communicate, the better the effort ratio gets. It must be noted here that these systems can also be part of multiple, independent projects for which the toolchain is applied.

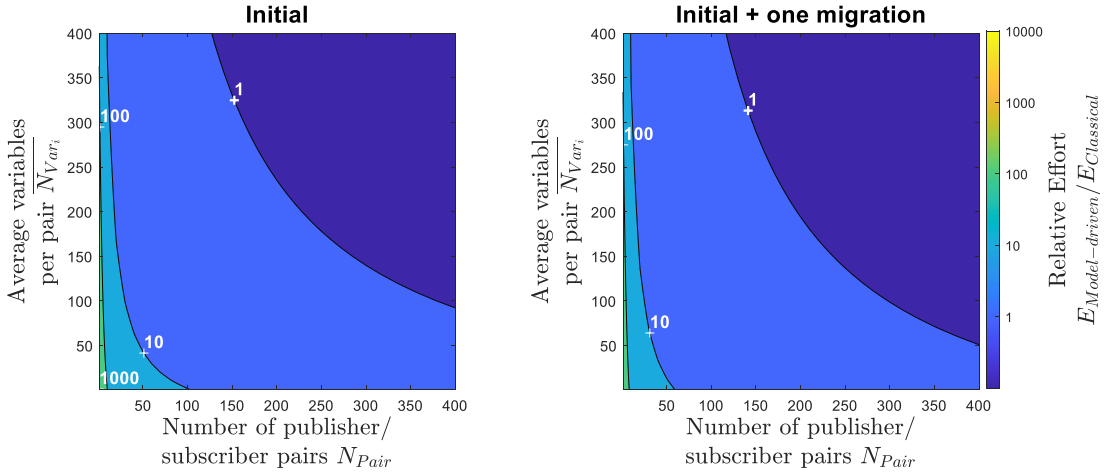


Figure 52: Relative effort between model-driven approach and classical, manual programming, including the effort for the creation of the toolchain as a function of the average number of variables per pair. Only initial deployment (left), including one migration (right). Logarithmic scale of the colormap.

At an average number of variables per publisher/subscriber pair of

$$\overline{N_{Var_i}} \geq \text{ceil}\left(\frac{57266 - 51 \cdot N_{Pair}}{N_{Pair}}\right), \tag{10}$$

the model-driven approach can outperform classical software development for the case of only initial deployment. For instance, this includes systems with 300 pairs and 140 variables each or systems with 150 pairs and 330 variables each. These systems are in the range of sophisticated data collection architectures where data from a multitude of systems must be gathered. On the other hand, these systems do not have to be part of a single project but can be part of several independent projects.

When one protocol migration for all systems is additionally considered, the relative effort is reduced to

$$\overline{N_{Var_i}} \geq \text{ceil}\left(\frac{57266 - \frac{646}{7} \cdot N_{Pair}}{N_{Pair}}\right). \tag{11}$$

Therefore, the break-even is reached for smaller systems. This includes, for instance systems with 300 pairs and 99 variables per pair or systems with 150 pairs and 290 variables per pair. As indicated due to the results of the migration scenario, the break-even is earlier reached for systems containing a large number of pairs. The reason is that an effort reduction during migration is only related to the number of pairs in the system.

Therefore, given the typical sizes of data collection projects in industrial automation, and the reusability of the DSL and the toolchain, the model-driven generation of the communication part of data collection architectures can decrease implementation efforts (*Req-A*$_{Dep}$). Furthermore, the approach can scale up for industrial applications.

7.6. Expert Workshop and Questionnaire

A workshop with industry experts was conducted to evaluate the approach and to support the findings related to the individual case-studies presented earlier. The expert group consisted of $n = 14$ industrial experts from the field of industrial automation ranging from OEMs to production plant manufacturers and operators of large chemical plants. The positions of the respective experts range from project engineers tasked with digitization projects, data analysts in the field of predictive maintenance and control, to head of their respective departments, for instance, research and development.

At the beginning of the workshop, an introduction of 20 minutes was given to the experts. The introduction included a wrap-up of industrial problems and challenges related to data collection and integration. Moreover, industrial protocols and state-of-the-art approaches to overcome these problems were presented. Subsequently, the developed approach was introduced in more detail using the application example from Section 5.2 (see Chapter 13 for a list of all occurrences of the application example). The presentation closed with a comparison of the implementation efforts between classical programming of a P2P architecture and the model-driven approach using a middleware based on a preliminary version of the effort extrapolation case-study (cf. Section 7.5).

Afterward, the contents of the presentation and the preliminary results were discussed for about 20 minutes with all experts. During the discussion, the experts pointed out the benefits of the approach, but also raised concerns. Especially the limitation of the code generation to C# code was criticized. This limitation restricts the applicability of the approach for greenfield PLCs where code manipulations are possible. Here, the generation of IEC 61131-3 [IEC61131] compliant code would be beneficial to include the communication functionality into the PLCs directly. One possibility can be the generation of code in the PLCopen XML exchange format [IEC61131] for direct import into the respective programming environments.

Nevertheless, it must be considered that despite the IEC 61131-3 being a standard, some PLC manufacturers rely on modified versions of the programming languages defined in IEC 61131-3 or only support programming in C. Furthermore, the generation of IEC 61131-3 compliant code would require specific support libraries on the PLCs. For instance, while Beckhoff supports ADS [Bec19c], OPC UA [Bec19a], and MQTT [Bec19b], there is currently no support for Apache Kafka nor AMQP. Therefore, concerning the PLC-level, the code generation can only be used for external gateways or alternatively an execution in the non-real-time part of soft-PLCs that support the .NET Core framework.

At last, a questionnaire with two pages and 16 questions was filled out individually by the experts answering questions and giving estimations related to the comparison of the classical, manual programming, and the developed model-driven approach. The original German version of the questionnaire can be found in Appendix D.

One aspect of the questionnaire was the assessment of the approaches related to the dimensions feasibility of a realization, total effort, and expected benefit. The averaged results of this assessment are plotted in Figure 53 as a spider diagram, with values ranging from 1 (very low) to 10 (very high). The detailed results can be found in Table 23 in Appendix D.

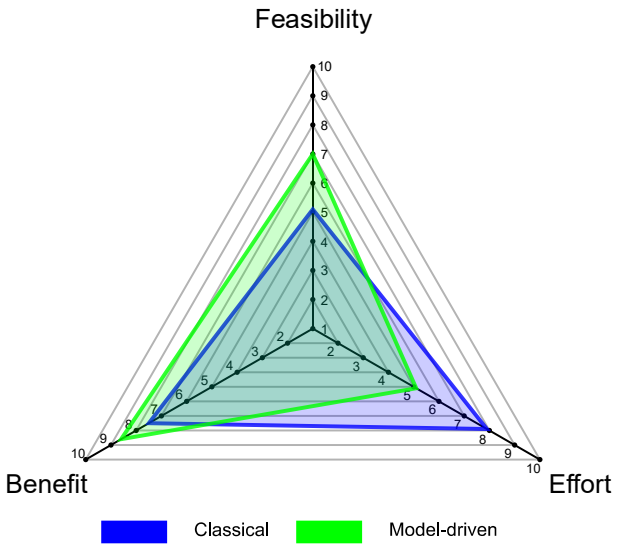


Figure 53: Comparison of the expert assessment of the dimensions feasibility, total effort, and benefit for classical, manually implemented P2P network and model-driven, middleware-based approach (n =14). Scale from 1 (very low) to 10 (very high).

While the expected benefits of both approaches only deviate to a small extent, the differences in the two other dimensions are more significant. Experts assessed the feasibility of the classical approach at an average value of around 5 and the necessary total effort at a value of 8. Therefore, it seems that the classical realization of data collection architectures can be feasible, yet, not without obstacles. The result could indicate an awareness of the importance of the topic and the possibility of a P2P implementation if the specific use-case justifies the significant implementation efforts. Different assessments were given for the developed model-driven approach: While the experts rated the feasibility of a model-driven implementation at a value of 7, the total effort is estimated to an average value of around 5. These results indicate that compared to the classical approach, the experts evaluated the feasibility of a model-driven data collection architecture substantially higher at decreased implementation efforts. It must be noted here, that the effort includes the effort for the modeling of the system, the subsequent model-driven generation of the system architecture, and the manual completion of the generated code basis with the user-specific code. Based on the assessment, it can be concluded that the industry experts expect significantly decreased efforts for the realization of a data collection architecture (*Req-*A_{Dep}). The different underlying concepts may explain the difference in the benefit assessment: while the classical implementation is based on direct connections between the systems, the model-based relies on a common *Data Management and Integration Broker*. This central broker makes data not only available between directly connected systems, but to all systems of the architecture if required. Furthermore, the addition of further participants is greatly simplified as only a single connection to the broker needs to be programmed.

The second aspect of the questionnaire was a detailed assessment of multiple statements related to both approaches. Therefore, the experts rated their subjective approval of each statement for both approaches. The answer scale included the possible answer options “disagree”, “rather disagree”, “partly/partly”, “rather agree”, and “agree”. During the analysis of the questionnaire, the expert answers were normalized to a scale ranging from -1 (disagree) to 1 (agree). The translated statements and results are summarized in Figure 54, while Table 24 in Appendix D contains the exact mean values and standard deviations per answer.

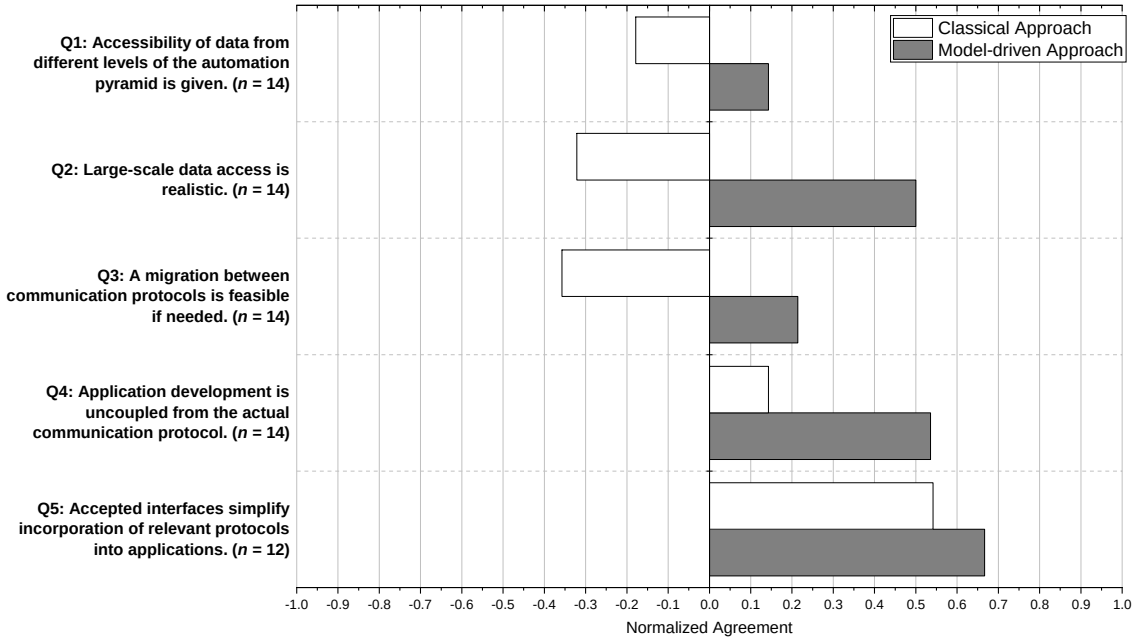


Figure 54: Normalized results of the expert evaluation per question (-1 Disagreeing, 1 Agreeing). Detailed results in Table 24.

Question Q1 is dedicated to the accessibility of data from different levels of the automation pyramid. The experts stated that better accessibility of the data for the model-driven approach (-0.18 versus 0.14 normalized agreement). However, the standard deviations of both mean answers are relatively large. This result indicates that individual agreements are not as ambiguous and deviate significantly. Therefore, the answers are in the range of the measurement uncertainty, but with a trend towards improved accessibility using the model-driven approach (*Req-*A_{ATP}).

The second question (Q2) is centered around the feasibility of large-scale data access. Here, significant differences in the expert agreement can be observed. The normalized agreement concerning this question is significantly higher for the model-driven approach (0.50) than for the classical approach (-0.32). Therefore, the experts see a practical implementation of an industrial data collection architecture based on the model-driven approach considerably more realistic. This large-scale data access includes access to data from various levels of the automation pyramid (*Req-*A_{ATP}).

Moreover, the proposed parallel operation to the existing control infrastructure is seen as feasible (*Req-*A_{POP}).

The feasibility of a migration scenario is the subject of Q3. The experts stated that a migration of the communication protocol is not very feasible when using the classical approach (-0.36 normalized agreement). This result can be explained by the significant portions of code that have to be rewritten as the extrapolation case-study in Section 7.5 showed. In contrast, a migration scenario was seen more positively with the model-driven approach (0.21 normalized agreement, *Req-*A_{ReDep}). Nevertheless, the experts were not entirely convinced of the feasibility. One possible explanation could be the missing support for IEC 61131-3 code in the model-driven approach. This lack of code generation makes manual changes to the PLC code necessary in case of a migration. Additionally, experts may fear the transition phase when migrating an existing system architecture to another protocol while in operation. Further investigations related to these aspects are needed in the future to identify these concerns accurately.

Question Q4 captures the expert opinion on the possibility of a decoupled development of the applications from the underlying communication protocol for data collection. Here, the experts tend to prefer the model-driven approach with its standardized interface (*Req-*SF_{API}) and an abstraction of the specifics of the protocols (*Req-*SF_{ACP}) over the classical approach (0.54 normalized agreement compared to 0.14). Using the developed software framework, the developed software can be efficiently decoupled from the communication technology. However, concerns could be raised around the high abstraction level of the developed programming interfaces. A possible solution to overcome this would be a multi-layered software framework with specific interfaces for complete abstraction of protocol-specific properties and a semi-abstracting layer that allows access to the specific features of the protocols, e.g., special QoS features.

Interestingly, the standard deviation for the classical approach is relatively high (0.52). This value may indicate different programming practices inside the respective companies for which the experts work. While some companies develop their software without a particular focus on reusability, others may define a standardized interface to decouple the distinct parts of the developed applications.

The same explanation could apply to the results of Q5: while the importance of standardized interfaces was highlighted for both approaches (0.54 for classical versus 0.67 for the model-driven approach), the standard deviation of the classical approach is around 0.54 (0.24 for the model-driven approach). This result once again means that experts tend to agree that the standardized

interface of the model-driven approach simplifies the support of multiple communication protocols (*Req-SF$_{API}$*). On the other hand, the expert opinion is not so uniform for the classical approach, indicating different software development practices inside the respective companies. If the software is developed with a strong focus on reusability and with defined interfaces, the support of additional protocols is relatively simple. In contrast, if the developed software is of a more monolithic structure, support for various protocols is more costly and difficult.

Therefore, according to the experts, a model-driven and middleware-based approach for the implementation of data collection architectures has the potential to mitigate the existing industrial challenges. Expert feedback was positive but also indicated that a code generation of IEC 61131-3-compliant code for PLCs should be focused in the future.

8. Assessment of the Fulfillment of the Requirements

The previous Chapter presented and discussed the findings of the evaluation case-studies and the fulfillment of the stated requirements (see Chapter 3). These are summarized with a detailed assessment in Table 22 with a reference back to the respective Sections in Chapter 7. The majority of requirements were evaluated positively in separate case-studies. Experimental results and expert assessments proved the suitability of the currently prototypical approach for model-driven data collection architectures.

However, significant concerns arise around the current lack of an integrated modeling platform that synchronizes model instance and visual representation of the DSL, as well as around the missing support for the generation of IEC 61131-3-compliant code for PLCs. Furthermore, the high level of abstraction in the programming interface can be problematic if specific characteristics of a communication protocol are of major importance for the realization of a specific use-case.

Table 22: Summary of the fulfillment of requirements and reference to the relevant Section in the evaluation Chapter (+ fulfilled, ○ partly fulfilled, - not fulfilled).

	Requirement		Rating	Details and reference to evaluation Section
Data Collection System Architectures (*Req-A*)	*Req-*A_{ATP}	Data collection from different levels of the automation pyramid	+	Experts verified the applicability of the concept for data collection from different levels (7.1.1). The lab-scale feasibility study demonstrated data collection from different levels (7.3). The expert questionnaire approved the feasibility of large-scale data access and suitability of the approach (7.6).
	Req-A_{TAC}	Technology-agnostic concept	+	Experts verified the technology-agnosticism of the concept (7.1.1). Mapping to other architectures demonstrated the applicability of the concept using different technologies and use-cases (7.1.2).
	Req-A_{POP}	Parallel operation to pyramid architecture	+	Experts verified that a second data channel for parallel operation to the pyramid architecture is included (7.1.1). The lab-scale feasibility study demonstrated the parallel operation (7.3). The expert questionnaire approved the possibility of parallel operation and data access over the second data channel (7.6).
	Req-A_{Dep}	Simplified implementation and configuration	○	The lab-scale feasibility study attested reduced implementation and migration efforts and a simplified implementation (7.3). The effort extrapolation case-study generalized the results and proved simplified implementation and migration between protocols. Yet, the initial effort for the creation of the model-driven toolchain is a major one-time effort (7.5). The experts assessed significantly simplified implementation and migration when using the model-driven approach but criticized the missing support for IEC 61131-3-compliant code generation (7.6).
	Req-A_{ReDep}	Simplified migration between technologies	○	

	Requirement		Rating	Details and reference to evaluation Section
Software Framework (*Req-SF*)	*Req-* SF_{API}	Standardized interfaces to minimize effort	○	The lab-scale feasibility study demonstrated the abstraction of technology-specific aspects of communication protocols and the standardized interfaces that prevented additional modifications to existing application-specific code in case of migrations (7.3). The extrapolation case-study confirms and intensifies these findings (7.5). The expert questionnaire affirmed the benefits of abstraction and the introduction of standardized interfaces for software development. The high level of abstraction was criticized for not giving access to enhanced protocol functionalities (7.6).
	Req- SF_{ACP}	Abstraction of technology-specific properties of communication	+	
	Req- SF_{Leg}	Support for legacy systems	+	Experts verified the feasibility of the architecture concept for the integration of legacy systems (7.1.1). The lab-scale feasibility study demonstrated the interfacing and integration of legacy systems into the data collection architecture (7.3).
Architecture Modeling Language (*Req-M*)	*Req-* M_{Sys}	System viewpoint	+	The industrial case-study with expert interviews evaluated the aspects of the modeling language positively (7.2). The system and data flow viewpoints were able to represent all relevant aspects. Annotation for properties and requirements allows the formalization and consideration of additional information. The graphical modeling notation was perceived positively by the experts, but an integrated editor for the DSL, including an automatic synchronization with the model instance, is currently missing.
	Req- M_{DF}	Data flow viewpoint	+	
	Req- $M_{PropReq}$	Annotations for properties and requirements	+	
	Req- M_{Graph}	Graphical modeling notation	○	
Model-driven Generation (*Req-G*)	*Req-* G_{Com}	Model-driven generation of communication interfaces	+	The communication architecture was automatically generated and included all communication interfaces for non-legacy systems in the lab-scale feasibility case-study (7.3) and the industrial case-study (7.4).

9. Summary and Outlook

Data analytics and big data principles are one of the central aspects of the I 4.0 concept. Through digitization and better connectivity, an ever-increasing amount of data from CPSoS and related systems is available for analysis. However, the distributed data has to be collected and integrated first before it can be analyzed. System architectures for data collection can automate and operationalize this task. Yet, the significant implementation efforts to realize such architectures induced by a large number of heterogeneous legacy systems prevalent in industrial automation impedes industrial uptake of I 4.0 concepts and prevents leveraging of data. Several researchers identified the concept of model-driven development as a possible solution to overcome these challenges [WMW18].

Nevertheless, no model-driven data collection architecture with support for multiple protocols and automatic generation of the communication architecture exists in the literature. Furthermore, DSLs with a visual notation and a formal description of CPSoS and associated data flows in the domain of industrial automation are a research gap.

Therefore, a model-driven approach for the realization of data collection architectures was developed in this thesis. It is based on a technology-neutral architecture concept that describes the elements and principles of data collection architectures. A DSL with visual notation was introduced that serves as a universal language during the interdisciplinary design of data collection architectures. A supporting metamodel structures the modeled information and makes it available for the model-driven generation of the data collection architecture. Here, M2T transformations are employed to generate the communication architecture based on predefined templates automatically. These templates stem from a developed software framework that supports an API for technology-abstracted communication based on multiple relevant IIoT protocols.

Distinct aspects of the approach were evaluated in multiple case-studies against requirements derived from industrial practice and the state-of-the-art. Expert interviews confirmed the suitability of the architecture concept for interfacing of existing legacy systems and parallel operation to the automation pyramid. The technology-neutral concept serves as a basis for practical realizations and guides the development process.

Furthermore, the expert evaluation of the DSL proved that relevant features of the systems, as well as the flow of data between them, could be successfully modeled and understood by experts from different disciplines. Additionally, the possibility to annotate the models with properties and requirements of the systems was evaluated positively.

The model-driven generation of data collection architectures was evaluated in three distinct case-studies. A lab-scale feasibility study was used to compare implementation efforts of manual programming versus the model-driven approach for a sufficiently complex use-case. The results showed significantly reduced implementation efforts for the model-driven generation of the data-collection architecture, even under the very conservative figures used for the comparison. An additional industrial case-study was used to verify the scalability of the model-driven generation for industrial-scale applications. The last case-study, an extrapolation study, was used to generalize the previous findings and to estimate scalability and implementation effort reduction of the approach sophisticatedly.

The evaluation proved the fulfillment of most requirements. Nonetheless, several weaknesses of the approach were uncovered. These include the missing synchronization between the graphical model and the instance of the metamodel, as well as the lack of code generation for PLCs due to the restriction on C#. Additionally, the high level of abstraction in the software framework was identified as problematic. Nevertheless, the hypothesis **(H1)** to **(H3)** can be seen as confirmed:

(H1)	A technology-neutral concept for a data collection architecture can bridge operational technology (OT) and information technology (IT) and allow data collection from production systems.
(H2)	A special domain-specific language with a graphical notation for data collection architectures supports the understanding and structuring of information during the engineering phase of these architectures by multi-disciplinary teams composed of engineers, IT architects, programmers, process experts, and data analysis.
(H3)	A model-based approach for automatic generation of data collection architectures reduces the effort for implementation and migration of these architectures.

Therefore, the proposed approach is successfully addressing the research gap.

Further research is dedicated to tackling the weaknesses of the approach and to extend it for additional applications. As a first step, an integrated modeling environment with full synchronization between the graphical editor and the underlying model is necessary for industrial applications. The automatic synchronization integrates both views and would allow the approach to be practically applicable. Currently, due to the manual synchronization between the two views, inconsistencies might occur. A realization of the modeling environment based on Graphiti [Ecl19f] or Sirius [Ecl19a] could replace Visio and benefit from an active integration into Eclipse, where the metamodel is implemented with EMF.

Furthermore, the effort comparisons, especially the extrapolation case-study, could be extended and improved by utilizing a different, more sophisticated approach for the effort estimation of the manually implemented code. While the approach in this works assumes a linear model based on LoC, which is sufficient for the intended conservative comparison, a non-linear model such as COCOMO II [Boe+00; Boe+95] could increase the validity and insights of such case-studies. However, the utilization of the model comes with the complexity of determining the additional model parameters, e.g., the capability of personnel or the complexity of the software product, which may be challenging to define for new technology such as the model-driven development of data collection architectures.

An additional point is better integration into the engineering process. While currently, all information is modeled manually by experts, existing information could be reused. For instance, engineering tools for the field level, such as TwinCAT 3 [Bec19d] or TIA Portal [Sie19], contain detailed information about the bus configuration, all hardware signals, as well as software information. File-based exchange of information or direct access over interfaces between these tools and the modeling environment could significantly decrease modeling efforts, reduce redundancies, and increase consistency. Furthermore, feeding back information to these systems, e.g., parts of the communication architecture as IEC 61131-3-compliant code, would close the loop between the different environments and greatly simplify industrial applications.

Besides IEC 61131-3-compliant code, also support for additional programming languages and environments, such as C+ or Java, would be beneficial. Furthermore, direct support for languages commonly used for data analysis, such as Matlab, Python, or R could further bridge the gap between industrial automation and data analysis. Also, support for a greater variety of protocols, e.g., DDS or REST, would improve the applicability of the developed approach. Nevertheless, not only the number of supported protocols is relevant, but also the flexibility of the software framework. Therefore, a multi-layer software framework that provides not only highly-abstracted programming interfaces but also intermediate layers with enhanced support for QoS features at the cost of decreased reusability could be beneficial. This would allow programmers, on the one hand, to migrate between protocols with equivalent support of QoS features without additional modifications. On the other hand, if migration to a protocol with incompatible support for QoS features would be needed, the high-level communication code could still be reused, while only the QoS-specific parts would require reimplementation.

An extension of the DSL is an additional point for further research. Inside the author's group, several approaches based on the same basic graphical notation can be found to capture timing characteristics [Vog+11] or safety aspects [STV19; SVF17]. Therefore, an extension of the DSL

would allow a universal usage of the developed language and an integration of approaches. Furthermore, more sophisticated modeling of data analysis functions (cf. [Ard+18]) would increase the information content of the models and improve the understanding of interactions between data collection and analysis. Also, inclusion or adaption of modeling elements to capture the dynamics of systems, such as UML state or sequence diagrams [OMG17], would increase the modeling depth significantly. To manage the complexity of the integrated DSL, the introduction of additional modeling viewpoints and textual representations, for instance, to define mappings between communication channels of the broker or security aspects as demonstrated by [Ber+18], is possible.

Following the proposal of Vogel-Heuser et al. [VWT17], design space exploration could be conducted based on the modeled information in order to determine suitable deployment alternatives. For instance, as Vogel-Heuser et al. [Vog+20] elaborated for distributed control systems, the proper characterization of timing behavior is of major importance. Therefore, the integration of network and system simulations would allow an offline derivation of optimized design and deployment alternatives. Such an integrated development tool would support the engineering of data collection architectures also during the earlier stages of the systems engineering. A similar approach has already been published for DDS-based communication systems [TÇK18]. Based on the simulation of the systems and networks, such as presented by Jha et al. [Jha+20], multi-objective optimization [BTT98] could be used to distribute data collection and manipulation tasks inside a network automatically.

As the last point, the integration of DevOps and model@run.time principles for the model-driven development [BBF09; CW20; Wor+20] could increase information usage and minimize development times. For instance, monitoring of the runtime behavior of deployed architectures would provide insights and ensure proper operation. Based on the modeled information, the monitoring functionalities could be generated, configured, and deployed using the same model-driven toolchain. Monitoring of data flows and QoS fulfillment was identified as one of the major challenges for the integration of IIoT and data analytics by Ranjan et al. [Ran+18]. Both could be tackled based on the developed approach. Also, the consideration of the temporal factor in the models could allow tracing the evolution of the architecture and QoS fulfillment over time [Bil+18]. Also, the stronger coupling and integration of design-time models about CPS with runtime aspects, such as data analysis, as proposed by Wolny et al. [Wol+18; Wol+20], could enhance the information content of models. This would allow, for instance, the generation of application-specific logic and automatic reasoning of the actual physical meaning of transported data. Furthermore, with a full description of all involved processes during design time (cp. modeling of dynamics and dependencies between systems), specific parts of the data collection could be set up automatically without manual modeling.

10. Literature

[.NE19] .NET Foundation, 2019, „C#. Version 8.0“ [Online] Available: https://github.com/dotnet/csharplang/tree/master/spec, [Accessed: 05-04-20].

[.NE20] .NET Foundation, 2020, „.NET Core. Version 3.1“ [Online] Available: https://github.com/dotnet/core, [Accessed: 05-04-20].

[Aic18] Aicher, T., „Automatic Backwards Compatibility of Automated Material Flow Software,“ Dissertation, Technical University of Munich, Munich, Germany. Institute of Automation and Information Systems, 2018.

[AIM10] Atzori, L., Iera, A. and Morabito, G., „The Internet of Things. A Survey,“ In: *Computer Networks, vol.* 54, no. 15, pp. 2787–2805, 2010.

[Al-+15] Al-Fuqaha, A., Khreishah, A., Guizani, M., Rayes, A. and Mohammadi, M., „Toward Better Horizontal Integration among IoT Services,“ In: *IEEE Commun. Mag., vol.* 53, no. 9, pp. 72–79, 2015.

[Alv+18] Alvarez, M. L., Sarachaga, I., Burgos, A., Estevez, E. and Marcos, M., „A Methodological Approach to Model-Driven Design and Development of Automation Systems,“ In: *IEEE Trans. Automat. Sci. Eng., vol.* 15, no. 1, pp. 67–79, 2018.

[Apa19] Apache Software Foundation, 2019, „Apache Kafka. Version 2.3.1“ [Online] Available: https://kafka.apache.org/, [Accessed: 03-12-19].

[Apa20] Apache Software Foundation, 2020, „Apache PLC4X. Version 0.6.0“ [Online] Available: https://github.com/apache/plc4x, [Accessed: 16-04-20].

[Ara+15] Aravantinos, V., Voss, S., Teufl, S., Hölzl, F. and Schätz, B., „AutoFOCUS 3: Tooling Concepts for Seamless, Model-based Development of Embedded Systems,“ In: *ACES-MB&WUCOR@ MoDELS, vol.* 1508, pp. 19–26, 2015.

[Ard+18] Ardagna, C. A., Bellandi, V., Bezzi, M., Ceravolo, P., Damiani, E. and Hebert, C., „Model-based Big Data Analytics-as-a-Service. Take Big Data to the Next Level,“ In: *IEEE Trans. Serv. Comput.*, pp. 1, 2018.

[ATL05] ATLAS group, „ATL: Atlas Transformation Language. Specification of the ATL Virtual Machine,“ 2005.

[Aut14] AutomationML Consortium, „AutomationML Whitepaper. Communication,“ 2014.

[Ban+16] Bangemann, T., Riedl, M., Thron, M. and Diedrich, C., „Integration of Classical Components Into Industrial Cyber–Physical Systems,“ In: *Proc. IEEE, vol.* 104, no. 5, pp. 947–959, 2016.

[Bar+15] Barbosa, J., Leitão, P., Adam, E. and Trentesaux, D., „Dynamic Self-organization in Holonic Multi-agent Manufacturing Systems: The ADACOR evolution,“ In: *Comput. in Ind., vol.* 66, pp. 99–111, 2015.

[Bas+11] Bassi, L., Secchi, C., Bonfe, M. and Fantuzzi, C., „A SysML-Based Methodology for Manufacturing Machinery Modeling and Design," In: *IEEE/ASME Trans. Mechatron., vol.* 16, no. 6, pp. 1049–1062, 2011.

[Bau+05] Bauer, A., Broy, M., Romberg, J., Schätz, B., Braun, P., Freund, U., Mata, N., Sandner, R. and Ziegenbein, D., „AutoMoDe - Notations, Methods, and Tools for Model-Based Development of Automotive Software," In: *SAE Technical Paper Series:* SAE International400, Warrendale, PA, United States, 2005.

[BBF09] Blair, G., Bencomo, N. and France, R. B., „Models@ run.time," In: *Computer, vol.* 42, no. 10, pp. 22–27, 2009.

[BCW17] Brambilla, M.; Cabot, J.; Wimmer, M., „Model-driven Software Engineering in Practice." San Rafael, C..: Morgan & Claypool Publishers (Synthesis lectures on software engineering, 4), 2017.

[Bec19a] Beckhoff Automation GmbH & Co. KG, 2019, „TF6100. TC3 OPC UA" [Online] Available: https://www.beckhoff.de/TF6100/, [Accessed: 27-01-20].

[Bec19b] Beckhoff Automation GmbH & Co. KG, 2019, „TF6701. TC3 IoT Communication (MQTT). Version 3.1.4024.4" [Online] Available: https://www.beckhoff.de/TF6701/, [Accessed: 27-01-20].

[Bec19c] Beckhoff Automation GmbH & Co. KG, 2019, „TwinCAT ADS" [Online] Available: https://infosys.beckhoff.com/content/1033/tcadscommon/html/tcadscommon_intro.htm?id=898081192215463875, [Accessed: 23-08-19].

[Bec19d] Beckhoff Automation GmbH & Co. KG, 2019, „TwinCAT3. Version 3.1.4024.4" [Online] Available: https://www.beckhoff.de/twincat3/, [Accessed: 17-01-20].

[Ben+17] Benaben, F., Truptil, S., Mu, W., Pingaud, H., Touzi, J., Rajsiri, V. and Lorre, J.-P., „Model-driven Engineering of Mediation Information System for Enterprise Interoperability," In: *Int. J. Comput. Integ. M., vol.* 79, pp. 1–22, 2017.

[Ber+18] Berrouyne, I., Adda, M., Mottu, J.-M., Royer, J.-C. and Tisi, M., „Towards Model-Based Communication Control for the Internet of Things," In: Mazzara, M., Ober, I. and Salaün, G. (Eds.): *Software Technologies: Applications and Foundations, Bd.* 11176. Cham: Springer International Publishing (Lecture notes in computer science), pp. 644–655, 2018.

[Ber13] Berman, J. J., „Introduction," In: *Principles of Big Data:* Elsevier, pp. xix–xxvi, 2013.

[Ber14] Bernstein, D., „Containers and Cloud: From LXC to Docker to Kubernetes," In: *IEEE Cloud Comput., vol.* 1, no. 3, pp. 81–84, 2014.

[BFS13] Bonfè, M., Fantuzzi, C. and Secchi, C., „Design Patterns for Model-based Automation Software Design and Implementation," In: *Control Eng. Pract., vol.* 21, no. 11, pp. 1608–1619, 2013.

[Bi17] Bi, Z., „Embracing Internet of Things (IoT) and Big Data for Industrial Informatics,“ In: *Enterp. Inf. Sys., vol.* 11, no. 7, pp. 949–951, 2017.

[Bil+18] Bill, R., Mazak, A., Wimmer, M. and Vogel-Heuser, B., „On the Need for Temporal Model Repositories,“ In: Seidl, M. and Zschaler, S. (Eds.): *Software Technologies: Applications and Foundations, Bd.* 10748. Cham: Springer International Publishing (Lecture notes in computer science), pp. 136–145, 2018.

[Bir+10] Birkhofer, R.; Wollschläger, M.; Schrieber, R.; Winzenick, M.; Kalhoff, J.; Kleedörfer, C. et al., „Life-Cycle-Management für Produkte und Systeme der Automation: Ein Leitfaden des Arbeitskreises Systemaspekte im ZVEI Fachverband Automation.“: Zentralverb. Elektrotechnik- und Elektronikindustrie, Fachverb. Automation, 2010.

[Bis+99] Bisbal, J., Lawless, D., Wu, B. and Grimson, J., „Legacy Information Systems. Issues and Directions,“ In: *IEEE Softw., vol.* 16, no. 5, pp. 103–111, 1999.

[Boe+00] Boehm, B. William; Abts, C.; Brown, A. W.; Chulani, S.; Clark, B. K.; Horowitz, E. et al., „Software Cost Estimation with Cocomo II.“ Upper Saddle River, NJ: Prentice Hall, 2000.

[Boe+95] Boehm, B., Clark, B., Horowitz, E., Westland, C., Madachy, R. and Selby, R., „Cost models for Future Software Life Cycle Processes: COCOMO 2.0,“ In: *Ann Software Eng, vol.* 1, no. 1, pp. 57–94, 1995.

[Bou+19] Bouloukakis, G., Georgantas, N., Ntumba, P. and Issarny, V., „Automated Synthesis of Mediators for Middleware-layer Protocol Interoperability in the IoT,“ In: *Future Gener. Comp. Sy., vol.* 101, pp. 1271–1294, 2019.

[Bou17] Bouloukakis, G., „Enabling Emergent Mobile Systems in the IoT : from Middleware-layer Communication Interoperability to Associated QoS Analysis,“ Doctoral Thesis, Université Pierre et Marie Curie - Paris VI. École Doctorale Informatique, Télécommunications et Électronique, 2017.

[Boy+18] Boyes, H., Hallaq, B., Cunningham, J. and Watson, T., „The Industrial Internet of Things (IIoT): An analysis framework,“ In: *Comput. Ind., vol.* 101, pp. 1–12, 2018.

[Bro+08] Broy, M., Fox, J., Hölzl, F., Koss, D., Kuhrmann, M., Meisinger, M., Penzenstadler, B., Rittmann, S., Schätz, B., Spichkova, M. and Wild, D., „Service-Oriented Modeling of CoCoME with Focus and AutoFocus,“ In: Rausch, A., Reussner, R., Mirandola, R. and Plášil, F. (Eds.): *The Common Component Modeling Example, Bd.* 5153. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture notes in computer science), pp. 177–206, 2008.

[Bro+93] Broy, M., Dedrichs, F., Dendorfer, C., Fuchs, M., Gritzner, T. F. and Weber, R., „The Design of Distributed Systems – An Introduction to FOCUS,“ Technical University of Munich, 1993.

[BS15] Breivold, H. P. and Sandstrom, K., „Internet of Things for Industrial Automation—Challenges and Technical Solutions," In: *2015 IEEE International Conference on Data Science and Data Intensive Systems:* IEEE, pp. 532–539, 2015.

[BTT98] Blickle, T., Teich, J. and Thiele, L., „System-Level Synthesis Using Evolutionary Algorithms," In: *Des. Autom. for Embed. Syst., vol.* 3, no. 1, pp. 23–58, 1998.

[BXW14] Bi, Z., Xu, L. D. and Wang, C., „Internet of Things for Enterprise Systems of Modern Manufacturing," In: *IEEE Trans. Ind. Inf., vol.* 10, no. 2, pp. 1537–1546, 2014.

[Cai+19] Caigny, J. de, Tauchnitz, T., Becker, R., Diedrich, C., Schröder, T., Großmann, D., Banerjee, S., Graube, M. and Urbas, L., „NOA – Von Demonstratoren zu Pilotanwendungen," In: *atp, vol.* 61, no. 1–2, pp. 44–55, 2019.

[Cai18] Caigny, J. de, 2018, „Namur Open Architecture. Ready for Products," NAMUR-Hauptsitzung (Organizer: Interessengemeinschaft Automatisierungstechnik der Prozessindustrie e.V (NAMUR) Bad Neuenahr, 11/8/2018.

[Cal+17] Cala, A., Lüder, A., Cachada, A., Pires, F., Barbosa, J., Leitão, P. and Gepp, M., „Migration from Traditional Towards Cyber-physical Production Systems," In: *2017 IEEE 15th International Conference on Industrial Informatics (INDIN):* IEEE, pp. 1147–1152, 2017.

[Can18] Canonical Ltd., 2018, „Ubuntu. Version 18.04.3 LTS" [Online] Available: http://releases.ubuntu.com/18.04/, [Accessed: 17-01-20].

[Car17] Carlsson, O., „Engineering of IoT Automation Systems," Doctoral Thesis, Luleå University of Technology, Luleå. EISLAB, 2017.

[CFV20] Cha, S., Fischer, J. and Vogel-Heuser, B., „Analysis Of Metamodels For Model-Based Production Automation System Engineering," In: *IET Collaborative Intelligent Manufacturing*, pp. 13, 2020.

[Cha+17] Chakravorti, N., Dimanidou, E., Angione, G., Wermann, J. and Gosewehr, F., „Validation of PERFoRM Reference Architecture Demonstrating an Automatic Robot Reconfiguration Application," In: *2017 IEEE 15th International Conference on Industrial Informatics (INDIN):* IEEE, pp. 1167–1172, 2017.

[Cha04] Chappell, D. A., „Enterprise Service Bus." Sebastopol: O'Reilly Media (Theory in practice), 2004.

[Che+18] Cheng, B., Zhang, J., Hancke, G. P., Karnouskos, S. and Colombo, A. W., „Industrial Cyberphysical Systems. Realizing Cloud-Based Big Data Infrastructures," In: *IEEE Ind. Electron. M., vol.* 12, no. 1, pp. 25–35, 2018.

[Cia+17] Ciavotta, M., Alge, M., Menato, S., Rovere, D. and Pedrazzoli, P., „A Microservice-based Middleware for the Digital Factory," In: *Procedia Manuf., vol.* 11, pp. 931–938, 2017.

[Clo20] Cloud Native Computing Foundation, 2020, „Kubernetes. Version v1.18.0“ [Online] Available: https://github.com/kubernetes/kubernetes, [Accessed: 06-04-20].

[COC18a] COCOP Project, „Deliverable 3.7 - Software Architecture Description for the Runtime System (Update),“ 2018.

[COC18b] COCOP Project, „Deliverable 3.5 - Interface and Protocol Definitions,“ 2018.

[Con19] Confluent Inc., 2019, „Confluent.Kafka. Version 1.0.0-RC2“ [Online] Available: https://github.com/confluentinc/confluent-kafka-dotnet, [Accessed: 23-08-19].

[CPC17] Cimini, C., Pinto, R. and Cavalieri, S., „The Business Transformation Towards Smart Manufacturing: A Literature Overview About Reference Models and Research Agenda,“ In: *20th World Congress of the International Federation of Automatic Control*, pp. 15517–15522, 2017.

[Cus+03] Cusumano, M., MacCormack, A., Kemerer, C. F. and Crandall, B., „Software Development Worldwide: The State of the Practice,“ In: *IEEE Softw., vol.* 20, no. 6, pp. 28–34, 2003.

[CW20] Combemale, B. and Wimmer, M., „Towards a Model-Based DevOps for Cyber-Physical Systems,“ In: Bruel, J.-M., Mazzara, M. and Meyer, B. (Eds.): *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment, Bd.* 12055. Cham: Springer International Publishing (Lecture notes in computer science), pp. 84–94, 2020.

[Deb19] Debian Project, 2019, „Debian. Version 10“ [Online] Available: https://www.debian.org/releases/stable/index.en.html, [Accessed: 17-01-20].

[DED17] Derhamy, H., Eliasson, J. and Delsing, J., „IoT Interoperability - On-demand and Low Latency Transparent Multi-protocol Translator,“ In: *IEEE Internet Things J.*, pp. 1754–1763, 2017.

[Del+11] Delsing, J., Eliasson, J., Kyusakov, R., Colombo, A. W., Jammes, F., Nessaether, J., Karnouskos, S. and Diedrich, C., „A Migration Approach Towards a SOA-based Next Generation Process Control and Monitoring,“ In: *IECON 2011 - 37th Annual Conference of the IEEE Industrial Electronics Society:* IEEE, pp. 4472–4477, 2011.

[Del+17a] Delsing, J., Varga, P., Ferreira, L., Albano, M., Perreira, P. P., Eliasson, J., Carlsson, O. and Derhamy, H., „The Arrowhead Framework Architecture,“ In: Delsing, J. (Ed.): *IoT automation.* Arrowhead Framework. Boca Raton, FL: CRC Press, Taylor & Francis Group, pp. 43–88, 2017.

[Del+17b] Delsing, J., Eliasson, J., Albano, M., Varga, P., Ferreira, L., Derhamy, H., Hegedűs, C., Perreira, P. P. and Carlsson, O., „Arrowhead Framework Core Systems and Services,“ In: Delsing. J. (Ed.): *IoT automation.* Arrowhead Framework. Boca Raton, FL: CRC Press, Taylor & Francis Group, pp. 89–138, 2017.

[DeM79] DeMarco, T., „Structured Analysis and System Specification.“ Englewood Cliffs, NJ: Yourdon Press (Yourdon computing series), 1979.

[Der+15] Derhamy, H., Eliasson, J., Delsing, J. and Priller, P., „A Survey of Commercial Frameworks for the Internet of Things,“ In: *2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA):* IEEE, pp. 1–8, 2015.

[DIN91345] DIN SPEC 91345, 2016, „Reference Architecture Model Industrie 4.0 (RAMI4.0).“

[DLH13] Dehof, M., Lüder, A. and Heinze, M., „An approach for modelling communication networks in industrial control systems,“ In: *IECON 2013 – 39th Annual Conference of the IEEE Industrial Electronics Society:* IEEE, pp. 7702–7707, 2013.

[Doc20a] Docker Inc., „docker docs,“ [Online] Available: https://docs.docker.com/registry/, [Accessed: 14-01-20], 2020.

[Doc20b] Docker Inc., „docker docs,“ [Online] Available: https://docs.docker.com/engine/swarm/, [Accessed: 14-01-20], 2020.

[Doc20c] Docker Inc., „docker docs,“ [Online] Available: https://docs.docker.com/buildx/working-with-buildx/, [Accessed: 14-01-20], 2020.

[Doc20d] Docker Inc., „Enterprise Container Platform,“ [Online] Available: https://www.docker.com/, [Accessed: 23-12-19], 2020.

[Dot+18] Dotoli, M., Fay, A., Miśkowicz, M. and Seatzu, C., „An Overview of Current Technologies and Emerging Trends in Factory Automation,“ In: *Int. J. Prod.*, pp. 1–21, 2018.

[Dra+08] Drath, R., Luder, A., Peschke, J. and Hundt, L., „AutomationML – the Glue for Seamless Automation Engineering,“ In: *2008 IEEE International Conference on Emerging Technologies and Factory Automation:* IEEE, pp. 616–623, 2008 - 2008.

[DvT01] Dashofy, E. M., van der Hoek, A. and Taylor, R. N., „A Highly-extensible, XML-based Architecture Description Language,“ In: *Proceedings Working IEEE/IFIP Conference on Software Architecture:* IEEE Comput. Soc, pp. 103–112, 2001.

[DWD14] Dorn, C., Waibel, P. and Dustdar, S., „Architecture-Centric Design of Complex Message-Based Service Systems,“ In: Franch, X., Ghose, A. K., Lewis, G. A. and Bhiri, S. (Eds.): *Service-Oriented Computing, Bd.* 8831. Berlin, Heidelberg: Springer (Lecture notes in computer science), pp. 184–198, 2014.

[Ebe+15] Ebeid, E., Medina, J., Quaglia, D. and Fummi, F., „Extensions to the UML Profile for MARTE for Distributed Embedded Systems,“ In: *2015 Forum on Specification and Design Languages (FDL):* IEEE, pp. 1–8, 2015.

[Ecl19a] Eclipse Foundation, 2019, „Sirius. Version 6.3.0“ [Online] Available: https://www.eclipse.org/sirius/overview.html, [Accessed: 10-02-20].

[Ecl19b] Eclipse Foundation, 2019, „Eclipse Modeling Framework (EMF). Version 2.18“ [Online] Available: https://www.eclipse.org/modeling/emf/, [Accessed: 23-08-19].

[Ecl19c] Eclipse Foundation, 2019, „Eclipse Modeling Tools. Version 2019-09“ [Online] Available: https://www.eclipse.org/downloads/packages/release/2019-09/r/eclipse-modeling-tools, [Accessed: 10-01-20].

[Ecl19d] Eclipse Foundation, „Eclipse BaSyx,“ [Online] Available: https://www.eclipse.org/basyx/, [Accessed: 01-04-19], 2019.

[Ecl19e] Eclipse Foundation, 2019, „Eclipse Mosquitto. Version 1.6.7“ [Online] Available: https://github.com/eclipse/mosquitto, [Accessed: 03-12-19].

[Ecl19f] Eclipse Foundation, 2019, „Graphiti. Version 0.16.1“ [Online] Available: https://www.eclipse.org/graphiti/, [Accessed: 10-02-20].

[Ecl19g] Eclipse Foundation, 2019, „Eclipse Acceleo. Version 3.7.8“ [Online] Available: https://projects.eclipse.org/projects/modeling.m2t.acceleo, [Accessed: 23-08-19].

[EFQ15] Ebeid, E., Fummi, F. and Quaglia, D., „Model-Driven Design of Network Aspects of Distributed Embedded Systems,“ In: *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol.* 34, no. 4, pp. 603–614, 2015.

[EGW18] Epple, U., Grothoff, J. A. and Wagner, C., „BaSys 4.0: Metamodell der Komponenten und ihres Aufbaus,“ 2018.

[Eur13] EN 61968-11, 2013, „Application Integration at Electric Utilities – System Interfaces for Distribution Management – Part 11: Common Information Model (CIM) Extensions for Distribution.

[Fay+15] Fay, A., Vogel-Heuser, B., Frank, T., Eckert, K., Hadlich, T. and Diedrich, C., „Enhancing a Model-based Engineering Approach for Distributed Manufacturing Automation Systems with Characteristics and Design Patterns,“ In: *J. Syst. Softw., vol.* 101, pp. 221–235, 2015.

[Fei+05] Feiler, P. H., Lewis, B., Vestal, S. and Colbert, E., „An Overview of the SAE Architecture Analysis & Design Language (AADL) Standard: A Basis for Model-Based Architecture-Driven Embedded Systems Engineering,“ In: Dissaux, P., Filali-Amine, M., Michel, P. and Vernadat, F. (Eds.): *Architecture Description Languages, Bd.* 176. New York: Springer-Verlag (IFIP The International Federation for Information Processing), pp. 3–15, 2005.

[Fel+15] Felter, W., Ferreira, A., Rajamony, R. and Rubio, J., „An Updated Performance Comparison of Virtual Machines and Linux Containers,“ In: *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS):* IEEE, pp. 171–172, 2015 - 2015.

[Fer+17] Ferrer, B. R., Mohammed, W. M., Chen, E. and Lastra, J. L. M., „Connecting Web-based IoT Devices to a Cloud-based Manufacturing Platform,“ In: *IECON 2017 – 43rd Annual Conference of the IEEE Industrial Electronics Society:* IEEE, pp. 8628–8633, 2017.

[Fer+18] Ferrer, B. R., Wael, M. M., Martínez Lastra, J. L., Villalonga, A., Beruvides, G., Castaño, F. and Haber, R. E., „Towards the Adoption of Cyber-Physical Systems of Systems Paradigm in Smart Manufacturing Environments," In: *IEEE 16th International Conference of Indistrual Informationcs (INDIN) 2018*, pp. 792–799, 2018.

[Fes08] Festo Didactic GmbH & Co. KG, „EasyPort USB. Manual," 2008.

[Fes20] Festo Corporation, „Automation Technology with MPS®," [Online] Available: https://www.festo.com/us/en/e/technical-education/learning-systems/factory-automation-and-industry-4-0/automation-technology-with-mps-r-id_31963/, [Accessed: 17-01-20], 2020.

[FG13] Feiler, P. H.; Gluch, D. P., „Model-based Engineering with AADL. An Introduction to the SAE Architecture Analysis & Design Language." Upper Saddle River, N.J: Addison-Wesley (The SEI series in software engineering), 2013.

[Fia+18] Fiaschetti, L., Antunez, M., Trapani, E., Valenzuela, L., Rubiales, A., Risso, M. and Boroni, G., „Monitoring and Controlling Energy Distribution. Implementation of a Distribution Management System based on Common Information Model," In: *Int. J. Elec. Power, vol.* 94, pp. 67–76, 2018.

[Fie00] Fielding, R. T., „Architectural Styles and the Design of Network-based Software Architectures," Dissertation, University of California, Irvine, U.S.A., 2000.

[FIPA02] Specification SC00084F, 2002, „FIPA Agent Message Transport Protocol for HTTP."

[FKF16a] Fleischmann, H., Kohl, J. and Franke, J., „A Modular Web Framework for Socio-CPS-based Condition Monitoring," In: *2016 IEEE World Conference on Factory Communication Systems (WFCS):* IEEE, pp. 1–8, 2016.

[FKF16b] Fleischmann, H., Kohl, J. and Franke, J., „A Reference Architecture for the Development of Socio-cyber-physical Condition Monitoring Systems," In: *2016 11th System of Systems Engineering Conference (SoSE):* IEEE, pp. 1–6, 2016.

[FL17a] Ferrer, B. R. and Lastra, J. L. M., „Private Local Automation Clouds Built by CPS. Potential and Challenges for Distributed Reasoning," In: *Adv. Eng. Inform., vol.* 32, pp. 113–125, 2017.

[FL17b] Ferrer, B. R. and Lastra, J. L. M., „An Architecture for Implementing Private Local Automation Clouds Built by CPS," In: *IECON 2017 – 43rd Annual Conference of the IEEE Industrial Electronics Society:* IEEE, pp. 5406–5413, 2017.

[Fle+16] Fleischmann, H., Kohl, J., Franke, J., Reidt, A., Duchon, M. and Krcmar, H., „Improving Maintenance Processes with Distributed Monitoring Systems," In: *2016 IEEE 14th International Conference on Industrial Informatics (INDIN):* IEEE, pp. 377–382, 2016.

[FLV06] Feiler, P. H., Lewis, B. A. and Vestal, S., „The SAE Architecture Analysis & Design Language (AADL) a Standard for Engineering Performance Critical Systems," In:

2006 IEEE Conference on Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control: IEEE, pp. 1206–1211, 2006.

[Fol+17] Folmer, J., Kirchen, I., Trunzer, E., Vogel-Heuser, B., Pötter, T., Graube, M., Heinze, S., Urbas, L., Atzmüller, M. and Arnu, D., „Big und Smart Data - Herausforderungen in der Prozessindustrie," In: *atp, vol.* 59, pp. 58–69, 2017.

[Fow15] Fowler, M., „Continuous Integration," [Online] Available: https://martinfowler.com/articles/continuousIntegration.html, [Accessed: 14-01-20], 2015.

[Fra14] Frank, T., „Entwicklung und Evaluation einer Modellierungssprache für den Architekturentwurf von verteilten Automatisierungsanlagen auf Basis der Systems Modeling Language (SysML)," Dissertation, Technical University of Munich, Munich, Germany. Institute of Automation and Information Systems, 2014.

[FS17] Fitzgerald, B. and Stol, K.-J., „Continuous software engineering: A roadmap and agenda," In: *J. Syst. Softw., vol.* 123, pp. 176–189, 2017.

[Gam11] Gamma, E., „Design Patterns. Elements of Reusable Object-oriented Software." Boston: Addison-Wesley (Addison-Wesley professional computing series), 2011.

[GB12] Geisberger, E.; Broy, M., „agendaCPS. Integrierte Forschungsagenda Cyber-Physical Systems." Berlin, Heidelberg: Springer, 2012.

[Geo+13] Georgantas, N., Bouloukakis, G., Beauche, S. and Issarny, V., „Service-Oriented Distributed Applications in the Future Internet. The Case for Interaction Paradigm Interoperability," In: Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J. M., Mattern, F., Mitchell, J. C. et al. (Eds.): *Service-Oriented and Cloud Computing, Bd.* 8135. Berlin, Heidelberg: Springer (Lecture notes in computer science), pp. 134–148, 2013.

[GF07] Greifeneder, J. and Frey, G., „DesLaNAS - a Language for Describing Networked Automation Systems," In: *2007 IEEE Conference on Emerging Technologies & Factory Automation (EFTA 2007):* IEEE, pp. 1053–1060, 2007.

[Git20] Git project, 2020, „Git. Version 2.25.0" [Online] Available: https://github.com/git/git, [Accessed: 14-01-20].

[Goo19a] Google Inc., 2019, „gRPC. Version 1.23.0" [Online] Available: https://github.com/grpc/grpc, [Accessed: 06-09-19].

[Goo19b] Google Inc., 2019, „ProtoBuf. Version 3.9.1" [Online] Available: https://github.com/protocolbuffers/protobuf, [Accessed: 06-09-19].

[Gor+14] Gorecky, D., Schmitt, M., Loskyll, M. and Zühlke, D., „Human-machine-interaction in the Industry 4.0 Era," In: *2014 12th IEEE International Conference on Industrial Informatics (INDIN):* IEEE, pp. 289–294, 2014.

[Gos+17] Gosewehr, F., Wernann, J., Borysch, W. and Colombo, A. W., „Specification and Design of an Industrial Manufacturing Middleware," In: *Proc. IEEE International*

Conference on Industrial Informatics (INDIN). Emden: IEEE Press, pp. 1160–1166, 2017.

[Gos+18] Gosewehr, F., Wermann, J., Borysch, W. and Colombo, A. W., „Apache Camel based Implementation of an Industrial Middleware Solution," In: *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*, pp. 523–528, 2018.

[Gra20] Grafana Labs, 2020, „Grafana. Version 6.5.3" [Online] Available: https://github.com/grafana/grafana, [Accessed: 17-01-20].

[Gre07] Greifeneder, J., „Formale Analyse des Zeitverhaltens netzbasierter Automatisierungssysteme," Dissertation, Technische Universität Kaiserslautern, Kaiserslautern. Fachbereich Elektrotechnik und Informationstechnik, 2007.

[Grö+16] Gröger, C., Kassner, L., Hoos, E., Königsberger, J., Kiefer, C., Silcher, S. and Mitschang, B., „The Data-driven Factory - Leveraging Big Industrial Data for Agile, Learning and Human-centric Manufacturing," In: *Proceedings of the 18th International Conference on Enterprise Information Systems:* SCITEPRESS - Science and and Technology Publications, pp. 40–52, 2016.

[Gro+99] Grosu, R., Broy, M., Selic, B. and Stefănescu, G., „What is Behind UML-RT?," In: Kilov, H., Rumpe, B. and Simmonds, I. (Eds.): *Behavioral Specifications of Businesses and Systems*. Boston, MA: Springer US, pp. 75–90, 1999.

[GTD12] Gama, K., Touseau, L. and Donsez, D., „Combining Heterogeneous Service Technologies for Building an Internet of Things Middleware," In: *Comput. Commun., vol.* 35, no. 4, pp. 405–417, 2012.

[Haa97] Haaß, W.-D., „Handbuch der Kommunikationsnetze." Berlin, Heidelberg: Springer, 1997.

[Had+12] Hadlich, T., Home, S., Diedrich, C., Eckert, K., Frank, T., Fay, A. and Vogel-Heuser, B., „Time as Non-functional Requirement in Distributed Control Systems," In: *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA 2012):* IEEE, pp. 1–6, 2012.

[Har+16] Harrand, N., Fleurey, F., Morin, B. and Husa, K. E., „ThingML," In: Baudry, B. and Combemale, B. (Eds.): *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems – MODELS ,16*. New York, New York, USA: ACM Press, pp. 125–135, 2016.

[Has+13] Hashemi Farzaneh, M., Feldmann, S., Legat, C., Folmer, J. and Vogel-Heuser, B., „Modeling Multicore Programmable Logic Controllers in Networked Automation Systems," In: *IECON 2013 – 39th Annual Conference of the IEEE Industrial Electronics Society:* IEEE, pp. 4398–4403, 2013.

[Has+15] Hashem, I. A. T., Yaqoob, I., Anuar, N. B., Mokhtar, S., Gani, A. and Ullah Khan, S., „The Rise of "Big Data" on Cloud Computing. Review and Open Research Issues," In: *Inf. Syst., vol.* 47, pp. 98–115, 2015.

[Hev+04] Hevner, A., March, S. T., Park, J. and Ram, S., „Design Science in Information Systems Research," In: *MIS Q, vol.* 28, no. 1, pp. 75, 2004.

[HFV13] Hufnagel, J., Frank, T. and Vogel-Heuser, B., „Framework for a Model-based, Cross-domain System Interconnection in Automation Technology," In: *2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA)*, pp. 1–9, 2013.

[HKV18] Hästbacka, D., Kannisto, P. and Vilkko, M., „Data-driven and Event-driven Integration Architecture for Plant-wide Industrial Process Monitoring and Control," In: *IECON 2018 – 44th Annual Conference of the IEEE Industrial Electronics Society:* IEEE, pp. 2979–2985, 2018.

[HMS19] HMS Industrial Networks, „Industrial Network Market Shares 2019 According to HMS," [Online] Available: https://www.hms-networks.com/news-and-insights/news-from-hms/2019/05/07/industrial-network-market-shares-2019-according-to-hms, [Accessed: 04-02-20], 2019.

[Hol+13] Holtewert, P., Wutzke, R., Seidelmann, J. and Bauernhansl, T., „Virtual Fort Knox Federative, Secure and Cloud-based Platform for Manufacturing," In: *Procedia CIRP, vol.* 7, pp. 527–532, 2013.

[HP88] Hatley, D. J.; Pirbhai, I. A., „Strategies for Real-time System Specification." New York, NY: Dorset House Publ, 1988.

[HR00] Harel, D. and Rumpe, B., „Modeling Languages: Syntax, Semantics and All That Stuff, Part I: The Basic Stuff," 2000.

[Hub+96] Huber, F., Schätz, B., Schmidt, A. and Spies, K., „AutoFocus — A Tool for Distributed Systems Specification," In: Goos, G., Hartmanis, J., Leeuwen, J., Jonsson, B. and Parrow, J. (Eds.): *Formal Techniques in Real-Time and Fault-Tolerant Systems, Bd.* 1135. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture notes in computer science), pp. 467–470, 1996.

[HV15] Hufnagel, J. and Vogel-Heuser, B., „Data Integration in Manufacturing Industry: Model-based Integration of Data Distributed from ERP to PLC," In: *2015 IEEE 13th International Conference on Industrial Informatics*, pp. 275–281, 2015.

[HX14] He, W. and Xu, L. D., „Integration of Distributed Enterprise Applications. A Survey," In: *IEEE Trans. Ind. Inf., vol.* 10, no. 1, pp. 35–42, 2014.

[IEC60802] IEC/IEEE CD 60802 D1.1, 2019, „TSN Profile for Industrial Automation."

[IEC61131] IEC 61131-3, 2013a, „Programmable Controllers – Part 3: Programming Languages."

[IEC61131] IEC 61131-10, 2019b, „Programmable Controllers – Part 10: PLC Open XML Exchange Format."

[IEC61131] IEC 61131-9, 2013c, „Programmable Controllers – Part 9: Single-drop Digital Communication Interface for Small Sensors and Actuators (SDCI)."

[IEC61158] IEC 61158-1, 2019, „Industrial Communication Networks – Fieldbus Specifications – Part 1: Overview and Guidance for the IEC 61158 and IEC 61784 Series.“

[IEC61784] IEC 61784, 2019, „Industrial Communication Networks – Profiles Part 1: Fieldbus Profiles.“

[IEC62264] IEC 62264-1, 2013, „Enterprise-control System Integration – Part 1: Models and Terminology.“

[IEC62541] IEC TR 62541-1, 2016, „OPC Unified Architecture – Part 1: Overview and Concepts.“

[IEC62714] IEC 62714-1, 2018, „Engineering data exchange format for use in industrial automation systems engineering – Automation Markup Language – Part 1: Architecture and general requirements.“

[IEEE2413] IEEE 2413, 2019, „IEEE Standard for an Architectural Framework for the Internet of Things (IOT).“

[IEEE802] IEEE 802.1Q-2018, 2018, „IEEE Standard for Local and Metropolitan Area Networks – Bridges and Bridged Networks.“

[Ift+18] Iftikhar, U., Wael, M. M., Ferrer, B. R. and Martínez Lastra, J. L., „A Framework for Data Collection, Transformation and Processing in Industrial Systems,“ In: *IEEE 16th International Conference of Indistrual Informationcs (INDIN) 2018*, pp. 707–712, 2018.

[IK16] Ismail, A. and Kastner, W., „A Middleware Architecture for Vertical Integration,“ In: *2016 1st International Workshop on Cyber-Physical Production Systems (CPPS):* IEEE, pp. 1–4, 2016.

[IK17] Ismail, A. and Kastner, W., „Surveying the Features of Industrial SOAs,“ In: *2017 IEEE International Conference on Industrial Technology (ICIT):* IEEE, pp. 1199–1204, 2017.

[IMP19] IMPROVE Project, „IMPROVE,“ [Online] Available: http://improve-vfof.eu/, [Accessed: 16-12-19], 2019.

[Ind17a] Industrial Internet Consortium, „Architecture Alignment and Interoperability. An Industrial Internet Consortium and Plattform Industrie 4.0 Joint Whitepaper,“ Industrial Internet Consortium, 2017.

[Ind17b] Industrial Internet Consortium, „The Industrial Internet of Things. Volume G1: Reference Architecture,“ Industrial Internet Consortium, 2017.

[Ind17c] Industrial Internet Consortium, „The Industrial Internet of Things. Volume G5: Connectivity Framework,“ Industrial Internet Consortium, 2017.

[Ins13] Institute of Automation and Information Systems, Technical University of Munich, „Industrie 4.0,“ [Online] Available: http://i40d.ais.mw.tum.de/, [Accessed: 17-01-20], 2013.

[Ins20] Institute of Automation and Information Systems, Technical University of Munich, „Hybrid Process Model," [Online] Available: http://www.ais.mw.tum.de/en/research/equipment/hybrid-process-model/, [Accessed: 17-01-20], 2020.

[ISA95] ANSI/ISA 95.00.01–2000, 2000, „Enterprise-Control System Integration – Part I: Models and Terminology."

[Ism18] Ismail, A., „Service Oriented Manufacturing Infrastructure," Dissertation, TU Wien, Vienna, Austria. Faculty of Informatics, 2018.

[ISO19464] ISO/IEC 19464, 2014, „Information Technology – Advanced Message Queuing Protocol (AMQP) v1.0 specification."

[ISO19508] ISO/IEC 19508, 2014, „Information Technology – Object Management Group Meta Object Facility (MOF) Core."

[ISO19514] ISO/IEC 19514, 2017, „Information Technology – Object Management Group Systems Modeling Language (OMG SysML)."

[ISO20922] ISO/IEC 20922, 2016, „Information Technology – Message Queuing Telemetry Transport (MQTT) v3.1.1."

[ISO30141] ISO/IEC 30141, 2018, „Information technology – Internet of Things Reference Architecture (IoT RA)."

[ISO42010] ISO/IEC/IEEE 42010, 2011, „Systems and Software Engineering – Architecture Description."

[ISO7498] ISO/IEC 7498-1, 1994, „Information Technology – Open Systems Interconnection - Basic Reference Model: The Basic Model."

[Iss+16] Issarny, V., Bouloukakis, G., Georgantas, N. and Billet, B., „Revisiting Service-Oriented Architecture for the IoT: A Middleware Perspective," In: Sheng, Q. Z., Stroulia, E., Tata, S. and Bhiri, S. (Eds.): *Service-Oriented Computing: 14th International Conference, ICSOC 2016, Banff, AB, Canada, October 10-13, 2016, Proceedings*. Cham: Springer International Publishing, pp. 3–17, 2016.

[ITK19] Ismail, A., Truong, H.-L. and Kastner, W., „Manufacturing process data analysis pipelines: a requirements analysis and survey," In: *J. Big Data, vol.* 6, no. 1, pp. 1–26, 2019.

[Izz09] Izza, S., „Integration of industrial information systems. From Syntactic to Semantic Integration Approaches," In: *Enterp. Inf. Syst., vol.* 3, no. 1, pp. 1–57, 2009.

[Jas+09] Jasperneite, J., Imtiaz, J., Schumacher, M. and Weber, K., „A Proposal for a Generic Real-Time Ethernet System," In: *IEEE Trans. Ind. Inf., vol.* 5, no. 2, pp. 75–85, 2009.

[Jes+17] Jeschke, S., Brecher, C., Meisen, T., Özdemir, D. and Eschert, T., „Industrial Internet of Things and Cyber Manufacturing Systems," In: Jeschke, S., Brecher, C., Song, H. and Rawat, D. B. (Eds.): *Industrial Internet of Things, Bd.* 54. Cham:

Springer International Publishing (Springer Series in Wireless Technology), pp. 3–19, 2017.

[Jha+14] Jha, S., Jha, M., O'Brien, L. and Wells, M., „Integrating Legacy System into Big Data Solutions. Time to make the change," In: *Asia-Pacific World Congress on Computer Science and Engineering:* IEEE, pp. 1–10, 2014.

[Jha+20] Jha, D. N., Alwasel, K., Alshoshan, A., Huang, X., Naha, R. K., Battula, S. K., Garg, S., Puthal, D., James, P., Zomaya, A., Dustdar, S. and Ranjan, R., „IoTSim-Edge: A Simulation Framework for Modeling the Behavior of Internet of Things and Edge Computing Environments," In: *Softw: Pract. Exper., vol.* 3, no. 1, pp. 11, 2020.

[JPG12] Jardim-Goncalves, R., Popplewell, K. and Grilo, A., „Sustainable Interoperability. The Future of Internet based Industrial Enterprises," In: *Comput. Ind., vol.* 63, no. 8, pp. 731–738, 2012.

[JS 19] JS Foundation, 2019, „Node-RED. Version 1.0.3" [Online] Available: https://github.com/node-red/node-red, [Accessed: 17-01-20].

[Kag15] Kagermann, H., „Change Through Digitization. Value Creation in the Age of Industry 4.0," In: Albach, H., Meffert, H., Pinkwart, A. and Reichwald, R. (Eds.): *Management of permanent change*. Wiesbaden: Springer Fachmedien Wiesbaden, pp. 23–45, 2015.

[Kar+14] Karnouskos, S., Colombo, A. W., Bangemann, T., Manninen, K., Camp, R., Tilly, M., Sikora, M., Jammes, F., Delsing, J., Eliasson, J., Nappey, P., Hu, J. and Graf, M., „The IMC-AESOP Architecture for Cloud-Based Industrial Cyber-Physical Systems," In: Colombo, A. W., Bangemann, T., Karnouskos, S., Delsing, J., Stluka, P., Harrison, R. et al. (Eds.): *Industrial Cloud-Based Cyber-Physical Systems, Bd.* 7. Cham: Springer International Publishing, pp. 49–88, 2014.

[Kas+17] Kassner, L., Gröger, C., Königsberger, J., Hoos, E., Kiefer, C., Weber, C., Silcher, S. and Mitschang, B., „The Stuttgart IT Architecture for Manufacturing," In: Hammoudi, S., Maciaszek, L. A., Missikoff, M. M., Camp, O. and Cordeiro, J. (Eds.): *Enterprise Information Systems, Bd.* 291. Cham: Springer International Publishing (Lecture Notes in Business Information Processing), pp. 53–80, 2017.

[Kat08] Katzke, U., „Spezifikation und Anwendung einer Modellierungssprache für die Automatisierungstechnik auf Basis der Unified Modeling Language (UML)," Dissertation, Universität Kassel, Kassel. Fachbereich Elektrotechnik / Informatik, 2008.

[KBD09] Karnouskos, S., Bangemann, T. and Diedrich, C., „Integration of Legacy Devices in the Future SOA-based Factory," In: *IFAC Proceedings Volumes, vol.* 42, no. 4, pp. 2113–2118, 2009.

[Ker19] Kernschmidt, K., „Interdisciplinary Structural Modeling of Mechatronic Production Systems using SysML4Mechatronics," Dissertation, Technical University of Munich, Munich, Germany. Institute of Automation and Information Systems, 2019.

[Kir+18] Kirmse, A., Kraus, V., Hoffmann, M. and Meisen, T., „An Architecture for Efficient Integration and Harmonization of Heterogeneous, Distributed Data Sources Enabling Big Data Analytics," In: *Proceedings of the 20th International Conference on Enterprise Information Systems*, pp. 175–182, 2018.

[KK19] Kuo, Y.-H. and Kusiak, A., „From Data to Big Data in Production Research: the Past and Future Trends," In: *Int. J. Prod. Res., vol.* 57, no. 15-16, pp. 4828–4853, 2019.

[Kle+17] Klettner, C., Tauchnitz, T., Epple, U., Nothdurft, L., Diedrich, C., Schröder, T., Goßmann, D., Banerjee, S., Krauß, M., Latrou, C. and Urbas, L., „Namur Open Architecture," In: *atp, vol.* 59, no. 01–02, pp. 17, 2017.

[KUG20147] Khan, M. A.-u.-d., Uddin, M. F. and Gupta, N., „Seven V's of Big Data Understanding Big Data to Extract Value," In: *Proceedings of the 2014 Zone 1 Conference of the American Society for Engineering Education:* IEEE, pp. 1–5, 20147.

[Kuh+18] Kuhn, T., Antonino, P. O., Damm, M., Morgenstern, A., Schulz, D., Ziesche, C. and Müller, T., „Industrie 4.0 Virtual Automation Bus," In: Crnkovic, I., Chaudron, M., Chechik, M. and Harman, M. (Eds.): *Proceedings of the 40th International Conference on Software Engineering Companion Proceeedings - ICSE ,18*. New York, New York, USA: ACM Press, pp. 121–122, 2018.

[KV05a] Katzke, U. and Vogel-Heuser, B., „UML-PA as an Engineering Model for Distributed Process Automation," In: *IFAC Proceedings Volumes, vol.* 38, no. 1, pp. 129–134, 2005.

[KV05b] Katzke, U. and Vogel-Heuser, B., „Design and Application of an Engineering Model for Distributed Process Automation," In: *Proceedings of the 2005, American Control Conference, 2005:* IEEE, pp. 2960–2965, 2005.

[KWH13] Kagermann, Henning; Wahlster, Wolfgang; Helbig, Johannes (Hg.), „Recommendations for Implementing the Strategic Initiative INDUSTRIE 4.0. Final Report of the Industrie 4.0 Working Group," acatech, 2013.

[KY13] Kim, E.-J. and Youm, S., „Machine-to-machine Platform Architecture for Horizontal Service Integration," In: *J. Wireless Com. Network., vol.* 2013, no. 1, pp. 9, 2013.

[LBK15] Lee, J., Bagheri, B. and Kao, H.-A., „A Cyber-Physical Systems Architecture for Industry 4.0-based Manufacturing Systems," In: *Manufacturing Letters, vol.* 3, pp. 18–23, 2015.

[LCK16] Leitão, P., Colombo, A. W. and Karnouskos, S., „Industrial Automation based on Cyber-physical Systems Technologies. Prototype Implementations and Challenges," In: *Comput. Ind., vol.* 81, pp. 11–25, 2016.

[LCR05] Leitão, P., Casais, F. and Restivo, F., „Holonic Manufacturing Control: A Practical Implementation," In: Camarinha-Matos, L.-M. (Ed.): *Emerging Solutions for Future Manufacturing Systems, Bd.* 159. Boston: Kluwer Academic Publishers (IFIP International Federation for Information Processing), pp. 33–44, 2005.

[Lei+13] Leitão, P., Barbosa, J., Vrba, P., Skobelev, P., Tsarev, A. and Kazanskaia, D., „Multi-agent System Approach for the Strategic Planning in Ramp-Up Production of Small Lots," In: *2013 IEEE International Conference on Systems, Man, and Cybernetics:* IEEE, pp. 4743–4748, 2013.

[Lei+15] Leitão, P., Barbosa, J., Papadopoulou, M.-E. C. and Venieris, I. S., „Standardization in Cyber-physical Systems. The ARUM Case," In: *2015 IEEE International Conference on Industrial Technology (ICIT):* IEEE, pp. 2988–2993, 2015.

[Lei+16] Leitão, P., Barbosa, J., Pereira, A., Barata, J. and Colombo, A. W., „Specification of the PERFoRM Architecture for the Seamless Production System Reconfiguration," In: *IECON 2016 – 42nd Annual Conference of the IEEE Industrial Electronics Society:* IEEE, pp. 5729–5734, 2016.

[Lei+17] Leitão, P., Barbosa, J., Foehr, M., Calà, A., Perlo, P., Iuzzolino, G., Petrali, P., Vallhagen, J. and Colombo, A. W., „Instantiating the PERFoRM System Architecture for Industrial Case Studies," In: Borangiu, T., Trentesaux, D., Thomas, A., Leitão, P. and Oliveira, J. B. (Eds.): *Service Orientation in Holonic and Multi-Agent Manufacturing, Bd.* 694. Cham: Springer International Publishing (Studies in Computational Intelligence), pp. 359–372, 2017.

[Lei04] Leitão, P., „An Agile and Adaptive Holonic Architecture for Manufacturing Control," Dissertation, University of Porto, Porto, Portugal. Faculty of Engineering, 2004.

[LG99a] Lauber, R.; Göhner, P., „Prozessautomatisierung 2." Berlin, Heidelberg: Springer, 1999.

[LG99b] Lauber, R.; Göhner, P., „Prozessautomatisierung 1." Berlin, Heidelberg: Springer, 1999.

[Lie+18] Liebel, G., Marko, N., Tichy, M., Leitner, A. and Hansson, J., „Model-based Engineering in the Embedded Systems Domain: an Industrial Survey on the State-of-practice," In: *Softw. Syst. Model., vol.* 17, no. 1, pp. 91–113, 2018.

[Lin+17] Lin, Y.-C., Hung, M.-H., Huang, H.-C., Chen, C.-C., Yang, H.-C., Hsieh, Y.-S. and Cheng, F.-T., „Development of Advanced Manufacturing Cloud of Things (AMCoT) - A Smart Manufacturing Platform," In: *2017 13 th IEEE Conference on Automation Science and Engineering (CASE)*, pp. 255–262, 2017.

[Liu+16] Liu, H., Guo, J., Yu, W., Zhu, L., Liu, Y., Xia, T., Sun, R. and Gardner, R. M., „The Design and Implementation of the Enterprise Level Data Platform and Big Data Driven Applications and Analytics," In: *2016 IEEE/PES Transmission and Distribution Conference and Exposition (T&D):* IEEE, pp. 1–5, 2016.

[Liu+18] Liu, Y.-Y., Hung, M.-H., Lin, Y.-C., Chen, C.-C., Gao, W.-L. and Cheng, F.-T., „A Cloud-based Pluggable Manufacturing Service Scheme for Smart Factory," In: *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE):* IEEE, pp. 1040–1045, 2018.

[LJ16] Liu, C. and Jiang, P., „A Cyber-physical System Architecture in Shop Floor for Intelligent Manufacturing," In: *Procedia CIRP, vol.* 56, pp. 372–377, 2016.

[LNP19] Longo, F., Nicoletti, L. and Padovano, A., „Ubiquitous Knowledge Empowers the Smart Factory: the Impacts of a Service-oriented Digital Twin on Enterprises' Performance," In: *Annu. Rev. Control*, pp. 221–236, 2019.

[LPJ10] Lankhorst, M. M., Proper, H. A. and Jonkers, H., „The Anatomy of the ArchiMate Language," In: *Int. J. Inf. Sys. Model. Des., vol.* 1, no. 1, pp. 1–32, 2010.

[LR06] Leitão, P. and Restivo, F., „ADACOR: A holonic Architecture for Agile and Adaptive Manufacturing Control," In: *Comput. Ind., vol.* 57, no. 2, pp. 121–130, 2006.

[LVF17] Lewin, M., Voigtländer, S. and Fay, A., „Method for Process Modelling and Analysis with Regard to the Requirements of Industry 4.0," In: *43rd Annual Conference of the IEEE Industrial Electronics Society*, pp. 3957–3962, 2017.

[Mat19] MathWorks, I., 2019, „MATLAB. Version 2019b Update 3" [Online] Available: https://de.mathworks.com/, [Accessed: 17-01-20].

[May+13] Mayer, F., Pantförder, D., Diedrich, C. and Vogel-Heuser, B., „Deutschlandweiter I4.0-Demonstrator," Lehrstuhl für Automatisierung und Informationssysteme, 2013.

[Maz+18] Mazak, A., Lüder, A., Wolny, S., Wimmer, M., Winkler, D., Kirchheim, K., Rosendahl, R., Bayanifar, H. and Biffl, S., „Model-based Generation of Run-time Data Collection Systems Exploiting AutomationML," In: *Automatisierungstechnik, vol.* 66, no. 10, pp. 819–833, 2018.

[MCV05] Mens, T., Czarnecki, K. and Van Gorp, P., „A Taxonomy of Model Transformations," In: Bezivin, J. and Heckel, R. (Eds.): *Language Engineering for Model-Driven Software Development*. Dagstuhl, Germany: Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2005.

[Mic19a] Microsoft Corporation, 2019, „Visio. Version 2019" [Online] Available: https://products.office.com/de-de/visio/flowchart-software, [Accessed: 09-09-19].

[Mic19b] Microsoft Corporation, 2019, „Visual Studio. Version 2019" [Online] Available: https://visualstudio.microsoft.com/de/vs/, [Accessed: 10-01-20].

[Mic19c] Microsoft Corporation, 2019, „Microsoft.CodeAnalysis.Metrics. Version 2.9.8" [Online] Available: https://www.nuget.org/packages/Microsoft.CodeAnalysis.Metrics/, [Accessed: 20-01-20].

[Mín12] Mínguez, J., „A Service-oriented Integration Platform for Flexible Information Provisioning in the Real-time Factory," Dissertation, Universität Stuttgart, Stuttgart, Germany. Institut für Parallele und Verteilte Systeme, 2012.

[MJG11] Maga, C., Jazdi, N. and Göhner, P., „Reusable Models in Industrial Automation: Experiences in Defining Appropriate Levels of Granularity," In: *IFAC Proceedings Volumes, vol.* 44, no. 1, pp. 9145–9150, 2011.

[MKB07] Mahambre, S. P., Kumar S.D., M. and Bellur, U., „A Taxonomy of QoS-Aware, Adaptive Event-Dissemination Middleware,“ In: *IEEE Internet Comput., vol.* 11, no. 4, pp. 35–44, 2007.

[Moc+12] Moctezuma, L. E. G., Jokinen, J., Postelnicu, C. and Lastra, J. L. M., „Retrofitting a Factory Automation System to Address Market Needs and Societal Changes,“ In: *IEEE 10th International Conference on Industrial Informatics:* IEEE, pp. 413–418, 2012.

[Mon+16] Monostori, L., Kádár, B., Bauernhansl, T., Kondoh, S., Kumara, S., Reinhart, G., Sauer, O., Schuh, G., Sihn, W. and Ueda, K., „Cyber-physical Systems in Manufacturing,“ In: *CIRP Annals, vol.* 65, no. 2, pp. 621–641, 2016.

[Mon19] MongoDB Inc., 2019, „MongoDB. Version 4.2.2“ [Online] Available: https://github.com/mongodb/mongo/releases, [Accessed: 24-01-20].

[Moo09] Moody, D., „The “Physics” of Notations. Toward a Scientific Basis for Constructing Visual Notations in Software Engineering,“ In: *IIEEE Trans. Software Eng., vol.* 35, no. 6, pp. 756–779, 2009.

[Mor17] Morabito, R., „Virtualization on Internet of Things Edge Devices With Container Technologies: A Performance Evaluation,“ In: *IEEE Access, vol.* 5, pp. 8835–8850, 2017.

[MTC18] MTConnect® Standard, 2018, „Part 1.0 – Overview and Fundamentals.“

[MW15] Marz, N.; Warren, J., „Big data. Principles and best practices of scalable real-time data systems.“ Shelter Island, NY: Manning, 2015.

[NE175] NAMUR recommendation NE 175 (draft), 2020, „NAMUR Open Architecture - NOA Concept.“

[Neu+18] Neumann, A., Wisniewski, L., Ganesan, R. S., Rost, P. and Jasperneite, J., „Towards Integration of Industrial Ethernet with 5G Mobile Networks,“ In: *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS):* IEEE, pp. 1–4, 2018 - 2018.

[Neu07] Neumann, P., „Communication in Industrial Automation—What is Going on?,“ In: *Control. Eng. Pract., vol.* 15, no. 11, pp. 1332–1347, 2007.

[OMG08] Object Management Group (OMG), „MOF Model to Text Transformation Language,“ Object Management Group (OMG), 2008.

[OMG11] Object Management Group (OMG), „UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems,“ Object Management Group (OMG), 2011.

[OMG12] Object Management Group (OMG), „Service Oriented Architecture Modeling Language (SoaML) Specification,“ Object Management Group (OMG), 2012.

[OMG14] Object Management Group (OMG), „Object Constraint Language," Object Management Group (OMG), 2014a.

[OMG14] Object Management Group (OMG), „Model Driven Architecture (MDA) MDA Guide rev. 2.0," Object Management Group (OMG), 2014b.

[OMG15] Object Management Group (OMG), „Data Distribution Service (DDS)," Object Management Group (OMG), 2015.

[OMG16] Object Management Group (OMG), „Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification," Object Management Group (OMG), 2016a.

[OMG16] Object Management Group (OMG), „OMG Meta Object Facility (MOF) Core Specification," Object Management Group (OMG), 2016b.

[OMG17] Object Management Group (OMG), „OMG Unified Modeling Language (OMG UML)," Object Management Group (OMG), 2017.

[OMG18] Object Management Group (OMG), „OPC UA/DDS Gateway," Object Management Group (OMG), 2018.

[OMG19] Object Management Group (OMG), „OMG Systems Modeling Language (OMG SysML)," Object Management Group (OMG), 2019.

[OPC03] OPC Specification, 2003, „OPC Data Access Custom Interface Specification."

[OPC18] OPC Unified Architecture Specification Part 14, 2018, „PubSub."

[OPC19] OPC Foundation, 2019, „UA-.NetStandard. Version 1.4.355.26" [Online] Available: https://github.com/OPCFoundation/UA-.NETStandard, [Accessed: 23-08-19].

[Ope18] Open Mobile Alliance, „Lightweight Machine to Machine Technical Specification. Core," 2018.

[Ope19] Open Group, „ArchiMate 3.1 Specification," 2019.

[Ope20] OpenSim Ltd., 2020, „OMNeT++. Version 5.6.1" [Online] Available: https://github.com/omnetpp/omnetpp, [Accessed: 27-04-20].

[OSI19] OSIsoft, „PI System, [Online] Available: https://www.osisoft.com/pi-system/, [Accessed: 26-09-19], 2019.

[Pah15] Pahl, C., „Containerization and the PaaS Cloud," In: *IEEE Cloud Comput., vol.* 2, no. 3, pp. 24–31, 2015.

[Pan+19] Panetto, H., Iung, B., Ivanov, D., Weichhart, G. and Wang, X., „Challenges for the Cyber-physical Manufacturing Enterprises of the Future," In: *Annu. Rev. Control*, 2019.

[Pei19] Pei Breivold, H., „Towards Factories of the Future: Migration of Industrial Legacy Automation Systems in the Cloud Computing and Internet-of-things Context," In: *Enterp. Inf. Syst.*, pp. 1–21, 2019.

[Pen+17] Penas, O., Plateaux, R., Patalano, S. and Hammadi, M., „Multi-scale Approach from Mechatronic to Cyber-Physical Systems for the Design of Manufacturing Systems," In: *Comput. Ind., vol.* 86, pp. 52–69, 2017.

[Per+14] Perera, C., Zaslavsky, A., Christen, P. and Georgakopoulos, D., „Context Aware Computing for The Internet of Things. A Survey," In: *IEEE Commun. Surv. Tutorials, vol.* 16, no. 1, pp. 414–454, 2014.

[Per+18] Peres, R. S., Dionisio Rocha, A., Leitao, P. and Barata, J., „IDARTS – Towards Intelligent Data Analysis and Real-time Supervision for Industry 4.0," In: *Comput. Ind., vol.* 101, pp. 138–146, 2018.

[PER16a] PERFoRM Project, „Deliverable 2.2 – Definition of the System Architecture," 2016.

[PER16b] PERFoRM Project, „Deliverable 3.2 – Real-time Process Information Exploitation," 2016.

[PER17] PERFoRM Project, „Deliverable 5.1 – The PERFoRM Integration Approach," 2017.

[Pet17] Petrasch, R., „Model-based Engineering for Microservice Architectures using Enterprise Integration Patterns for Inter-service Communication," In: *2017 14th International Joint Conference on Computer Science and Software Engineering (JCSSE):* IEEE, pp. 1–4, 2017.

[Pet18] Petrasch, R., „Message-Oriented Middleware for System Communication. A Model-Based Approach," In: Meesad, P., Sodsee, S. and Unger, H. (Eds.): *Recent Advances in Information and Communication Technology 2017, Bd.* 566. Cham: Springer International Publishing (Advances in Intelligent Systems and Computing), pp. 253–263, 2018.

[PGP16] Pfrommer, J., Gruner, S. and Palm, F., „Hybrid OPC UA and DDS. Combining Architectural Styles for the Industrial Internet," In: *2016 IEEE World Conference on Factory Communication Systems (WFCS):* IEEE, pp. 1–7, 2016.

[Piv19a] Pivotal Software Inc., 2019, „RabbitMQ. Version 3.8.2" [Online] Available: https://github.com/rabbitmq/rabbitmq-server, [Accessed: 03-12-19].

[Piv19b] Pivotal Software Inc., 2019, „RabbitMQ.Net client. Version 6.0.0-pre3" [Online] Available: https://github.com/rabbitmq/rabbitmq-dotnet-client, [Accessed: 23-08-19].

[PJM12] Panetto, H., Jardim-Goncalves, R. and Molina, A., „Enterprise Integration and Networking. Theory and Practice," In: *Annu. Rev. Control, vol.* 36, no. 2, pp. 284–290, 2012.

[PN09] Pereira, C. E. and Neumann, P., „Industrial Communication Protocols," In: Nof, S. Y. (Ed.): *Springer Handbook of Automation*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 981–999, 2009.

[Pos19] PostgreSQL Global Development Group, 2019, „PostgreSQL. Version 11.6“ [Online] Available: https://www.postgresql.org/, [Accessed: 17-01-20].

[Pre00] Prechelt, L., „Empirical Comparison of Seven Programming Languages,“ In: *Computer, vol.* 33, no. 10, pp. 23–29, 2000.

[PTD19] Pusztai, T. W., Tsigkanos, C. and Dustdar, S., „Engineering Heterogeneous Internet of Things Applications: From Models to Code,“ In: *2019 IEEE 5th International Conference on Collaboration and Internet Computing (CIC):* IEEE, pp. 222–231, 2019 - 2019.

[Pyt19] Python Software Foundation, 2019, „Python. Version 3.7.3“ [Online] Available: https://www.python.org/, [Accessed: 17-01-20].

[Qur+17] Qureshi, K. A., Mohammed, W. M., Ferrer, B. R., Lastra, J. L. M. and Agostinho, C., „Legacy Systems Interactions with the Supply Chain Through the C2NET Cloud-based Platform,“ In: *2017 IEEE 15th International Conference on Industrial Informatics (INDIN):* IEEE, pp. 725–731, 2017.

[Raj+10] Rajkumar, R., Lee, I., Sha, L. and Stankovic, J., „Cyber-physical Systems,“ In: Sapatnekar, S. (Ed.): *Proceedings of the 47th Design Automation Conference on - DAC '10*. New York, New York, USA: ACM Press, pp. 731–736, 2010.

[Ran+18] Ranjan, R., Rana, O., Nepal, S., Yousif, M., James, P., Wen, Z., Barr, S., Watson, P., Jayaraman, P. P., Georgakopoulos, D., Villari, M., Fazio, M., Garg, S., Buyya, R., Wang, L., Zomaya, A. Y. and Dustdar, S., „The Next Grand Challenges: Integrating the Internet of Things and Data Science,“ In: *IEEE Cloud Comput., vol.* 5, no. 3, pp. 12–26, 2018.

[Raz+16] Razzaque, M. A., Milojevic-Jevric, M., Palade, A. and Clarke, S., „Middleware for Internet of Things. A Survey,“ In: *IEEE Internet Things J., vol.* 3, no. 1, pp. 70–95, 2016.

[RD18] Rentschler, M. and Drath, R., „Vendor-independent Modeling and Exchange of Fieldbus Topologies with AutomationML,“ In: *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA):* IEEE, pp. 956–963, 2018.

[Reu+11] Reussner, R., Becker, S., Burger, E., Happe, J., Hauck, M., Koziolek, A., Koziolek, H., Krogmann, K. and Kuperberg, M., „The Palladio Component Model,“ 2011.

[Reu+16] Reussner, R., Becker,S., Happe, J., Heinrich,R., Koziolek, A. and Koziolek, H. (Eds.), „Modeling and Simulating Software Architectures. The Palladio Approach,“ Cambridge, Massachusetts, London, England: The MIT Press, 2016.

[RFC1006] RFC 1006, 1987, „ISO Transport Service on top of the TCP.“

[RFC7252] RFC 7252, 2014, „The Constrained Application Protocol (CoAP).“

[Rie+14a] Riedl, M., Lüder, A., Heines, B. and Drath, R., „Kommunikation mit AutomationML beschreiben,“ In: *atp, vol.* 56, no. 11, pp. 44–51, 2014.

[Rie+14b] Riedl, M., Zipper, H., Meier, M. and Diedrich, C., „Cyber-physical Systems Alter Automation Architectures," In: *Annu. Rev. Control, vol.* 38, no. 1, pp. 123–133, 2014.

[Riv19] Riverbed Technology, 2019, „Riverbed Modeler. Version 18.8.0" [Online] Available: https://www.riverbed.com/gb/products/steelcentral/steelcentral-riverbed-modeler.html, [Accessed: 27-04-20].

[Rod15] Rodrigues da Silva, A., „Model-driven Engineering: A Survey Supported by the Unified Conceptual Model," In: *Comput. Lang. Syst. Str., vol.* 43, pp. 139–155, 2015.

[RPC19] Raptis, T. P., Passarella, A. and Conti, M., „Data Management in Industry 4.0: State of the Art and Open Challenges," In: *IEEE Access, vol.* 7, pp. 97052–97093, 2019.

[SAEAS5506C] AS5506C, 2017, „Architecture Analysis & Design Language (AADL)."

[Sau07] Sauter, T., „The Continuing Evolution of Integration in Manufacturing Automation," In: *IEEE Ind. Electron. Mag., vol.* 1, no. 1, pp. 10–19, 2007.

[Sau10] Sauter, T., „The Three Generations of Field-Level Networks—Evolution and Compatibility Issues," In: *IEEE Trans. Ind. Electron., vol.* 57, no. 11, pp. 3585–3595, 2010.

[Sch+02] Schätz, B., Pretschner, A., Huber, F. and Philipps, J., „Model-Based Development of Embedded Systems," In: Goos, G., Hartmanis, J., van Leeuwen, J., Bruel, J.-M. and Bellahsene, Z. (Eds.): *Advances in Object-Oriented Information Systems, Bd.* 2426. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture notes in computer science), pp. 298–311, 2002.

[Sch+18] Schel, D., Henkel, C., Stock, D., Meyer, O., Rauhöft, G., Einberger, P., Stöhr, M., Daxer, M. A. and Seidelmann, J., „Manufacturing Service Bus: An Implementation," In: *Procedia CIRP, vol.* 67, pp. 179–184, 2018.

[Sch15] Schütz, D., „Automatische Generierung von Softwareagenten für die industrielle Automatisierungstechnik der Steuerungsebene des Maschinen- und Anlagenbaus auf Basis der Systems Modeling Language," Dissertation, Technical University of Munich, Munich, Germany. Institute of Automation and Information Systems, 2015.

[Sel98] Selic, B., „Using UML for Modeling Complex Real-time Systems," In: Goos, G., Hartmanis, J., van Leeuwen, J., Mueller, F. and Bestavros, A. (Eds.): *Languages, Compilers, and Tools for Embedded Systems, Bd.* 1474. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture notes in computer science), pp. 250–260, 1998.

[SGL15] Sola, J., Gonzalez, A. and Lazaro, O., „Leveraging IoT Interoperability for Enhanced Business Process in Smart, Digital and Virtual Factories," In: Lauras, M. (Ed.): *Enterprise interoperability.* Interoperability for Agility, Resilience and Plas-

ticity of Collaborations ; I-EASA'14 Proceedings. London: ISTE Ltd (Interoperability research for networked enterprises applications and software), pp. 43–48, 2015.

[SGW94] Selic, B.; Gullekson, G.; Ward, P. T., „Real-time Object-oriented Modeling." New York: Wiley (Wiley professional computing), 1994.

[SHW99] Scherff, B.; Haese, E.; Wenzek, H. R., „Feldbussysteme in der Praxis." Berlin, Heidelberg: Springer Berlin Heidelberg, 1999.

[SID19] SIDAP Project, „SIDAP," [Online] Available: www.sidap.de, [Accessed: 04-02-20], 2019.

[Sie19] Siemens AG, 2019, „Totally Integrated Automation Portal (TIA Portal). Version V16" [Online] Available: https://new.siemens.com/global/de/produkte/automatisierung/industrie-software/automatisierungs-software/tia-portal.html, [Accessed: 24-02-20].

[SK03] Sendall, S. and Kozaczynski, W., „Model Transformation: the Heart and Soul of Model-driven Software Development," In: *IEEE Softw., vol.* 20, no. 5, pp. 42–45, 2003.

[SLV19] Sollfrank, M., Loch, F. and Vogel-Heuser, B., „Exploring Docker Containers for Time-sensitive Applications in Networked Control Systems," In: *2019 IEEE 17th International Conference on Industrial Informatics (INDIN):* IEEE, pp. 1760–1765, 2019 - 2019.

[Spa13] Spath, Dieter (Hg.), „Produktionsarbeit der Zukunft - Industrie 4.0, Stuttgart: Fraunhofer-Verl.", 2013.

[Sta73] Stachowiak, H., „Allgemeine Modelltheorie." Wien: Springer, 1973.

[Str+09] Strasser, T., Rooker, M., Hegny, I., Wenger, M., Zoitl, A., Ferrarini, L., Dede, A. and Colla, M., „A Research Roadmap for Model-driven Design of Embedded Systems for Automation Components," In: *2009 7th IEEE International Conference on Industrial Informatics:* IEEE, pp. 564–569, 2009.

[Str+18] Strasser, T. I., Andren, F. P., Vrba, P., Suhada, R., Moulis, V., Farid, A. M. and Rohjans, S., „An Overview of Trends and Developments of Internet of Things Applied to Industrial Systems," In: *IECON 2018 – 44th Annual Conference of the IEEE Industrial Electronics Society:* IEEE, pp. 2853–2860, 2018.

[Str+20] Struhár, V., Behnam, M., Ashjaei, M. and Papadopoulos, A. V., „Real-Time Containers: A Survey," 2020.

[STV19] Sollfrank, M., Trunzer, E. and Vogel-Heuser, B., „Graphical Modeling of Communication Architectures in Network Control Systems with Traceability to Requirements," In: *IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society:* IEEE, pp. 6267–6273, 2019.

[SVF17] Sollfrank, M., Vogel-Heuser, B. and Fahimipirehgalin, M., „Integration of Safety Aspects in Modeling of Networked Control Systems," In: *Proc. IEEE International Conference on Industrial Informatics (INDIN)*. Emden: IEEE Press, pp. 405–412, 2017.

[Tac19] Tacke, C., 2019, „OpenNETCF MQTT. Version 1.0.17253" [Online] Available: https://github.com/ctacke/mqtt, [Accessed: 23-08-19].

[TC16] Thramboulidis, K. and Christoulakis, F., „UML4IoT—A UML-based Approach to Exploit IoT in Cyber-physical Manufacturing Systems," In: *Comput. Ind., vol.* 82, pp. 259–272, 2016.

[TÇK18] Tekinerdogan, B., Çelik, T. and Köksal, Ö., „Generation of Feasible Deployment Configuration Alternatives for Data Distribution Service based Systems," In: *Comput. Stand. Interfaces, vol.* 58, pp. 126–145, 2018.

[Ter+18] Terzić, B., Dimitrieski, V., Kordić, S., Milosavljević, G. and Luković, I., „Development and evaluation of MicroBuilder. A Model-Driven Tool for the Specification of REST Microservice Software Architectures," In: *Enterp. Inf. Syst., vol.* 3, no. 5, pp. 1–24, 2018.

[The+16] Theorin, A., Bengtsson, K., Provost, J., Lieder, M., Johnsson, C., Lundholm, T. and Lennartson, B., „An Event-driven Manufacturing Information System Architecture for Industry 4.0," In: *Int. J. Prod. Res., vol.* 55, no. 5, pp. 1297–1311, 2016.

[TLV18] Trunzer, E., Lötzerich, S. and Vogel-Heuser, B., „Concept and Implementation of a Software Architecture for Unifying Data Transfer in Automated Production Systems," In: Niggemann, O. and Schüller, P. (Eds.): *IMPROVE - Innovative Modelling Approaches for Production Systems to Raise Validatable Efficiency, Bd.* 8. Berlin, Heidelberg: Springer Berlin Heidelberg (Technologien für die intelligente Automation), pp. 1–17, 2018.

[Tru+17] Trunzer, E., Kirchen, I., Folmer, J., Koltun, G. and Vogel-Heuser, B., „A Flexible Architecture for Data Mining from Heterogeneous Data Sources in Automated Production Systems," In: *2017 IEEE International Conference on Industrial Technology (ICIT):* IEEE, pp. 1106–1111, 2017.

[Tru+19a] Trunzer, E., Weiß, I., Pötter, T., Vermum, C., Odenweller, M., Unland, S., Schütz, D. and Vogel-Heuser, B., „Big Data trifft Produktion. Neun Pfeiler der industriellen Smart-Data-Analyse," In: *Automatisierungstechnische Praxis (atp), vol.* 61, no. 1–2, pp. 90–98, 2019.

[Tru+19b] Trunzer, E., Prata, P., Vieira, S. and Vogel-Heuser, B., „Concept and Evaluation of a Technology-independent Data Collection Architecture for Industrial Automation," In: *IECON 2019 – 45th Annual Conference of the IEEE Industrial Electronics Society:* IEEE, 2830-2836, 2019.

[Tru+19c] Trunzer, E., Calà, A., Leitão, P., Gepp, M., Kinghorst, J., Lüder, A., Schauerte, H., Reifferscheid, M. and Vogel-Heuser, B., „System Architectures for Industrie 4.0 Applications," In: *Prod. Eng. Res. Devel., vol.* 13, no. 2, pp. 247–257, 2019.

[Tru+20a] Trunzer, E., Schilling, T., Müller, M. and Vogel-Heuser, B., „Comparison of Communication Technologies for Industrial Middlewares and DDS-based Realization," In: *21st IFAC World Congress*, 8 (in Press), 2020.

[Tru+20b] Trunzer, E., Vogel-Heuser, B., Folmer, J. and Pötter, T., „Smart Data Architekturen. Vertikale und horizontale Integration," In: Vogel-Heuser, B. (Ed.): *Handbuch Industrie 4.0 Bd.5*. Berlin, Heidelberg: Springer, 2020.

[TVS18] Thramboulidis, K., Vachtsevanou, D. C. and Solanos, A., „Cyber-Physical Microservices," In: *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*, pp. 232–239, 2018.

[TWV20] Trunzer, E., Wullenweber, A. and Vogel-Heuser, B., „ Graphical Modeling Notation for Data Collection and Analysis Architectures in Cyber-physical Systems of Systems," In: *J. Ind. Inf. Integration, vol.* 19, pp. 100155, 2020.

[Var+17] Varga, P., Blomstedt, F., Ferreira, L. L., Eliasson, J., Johansson, M., Delsing, J. and Martínez de Soria, I., „Making System of Systems Interoperable – The Core Components of the Arrowhead Framework," In: *J. Netw. Comput. Appl., vol.* 81, pp. 85–95, 2017.

[VBF12] Vogel-Heuser, B., Bayrak, G. and Frank, U., „Forschungsfragen in „Produktionsautomatisierung der Zukunft," acatech, 2012.

[VDI2657] VDI/VDE Guideline 2657 Part 1, 2013, „Middleware in Industrial Automation - Fundamentals."

[VDI3687] VDI/VDE Guideline 3687, 19999, „Selection of Field Bus-systems by Evaluating their Performance Characteristics for Industrial Applications."

[VDI5600] VDI-Richtlinie 5600 Blatt 3, 2013, „Fertigungsmanagementsysteme (Manufacturing Execution Systems – MES)."

[VH08] Varga, A. and Hornig, R., „An Overview of the OMNeT++ Simulation Environment," In: Molnár, S. and Heath, J. (Eds.): *Proceedings of the First International ICST Conference on Simulation Tools and Techniques for Communications Networks and Systems:* ICST, 2008 - 2007.

[VH16] Vogel-Heuser, B. and Hess, D., „Guest Editorial Industry 4.0–Prerequisites and Visions," In: *IEEE Trans. Automat. Sci. Eng., vol.* 13, no. 2, pp. 411–413, 2016.

[Vog+09] Vogel-Heuser, B., Kegel, G., Bender, K. and Wucherer, K., „Global Information Architecture for Industrial Automation," In: *Automatisierungstechnische Praxis (atp), vol.* 51, no. 1-2, pp. 108–115, 2009.

[Vog+11] Vogel-Heuser, B., Feldmann, S., Werner, T. and Diedrich, C., „Modeling Network Architecture and Time Behavior of Distributed Control Systems in Industrial Plant

Automation,“ In: *IECON 2011 – 37th Annual Conference of the IEEE Industrial Electronics Society:* IEEE, pp. 2232–2237, 2011.

[Vog+12] Vogel-Heuser, B., Folmer, J., Frey, G., Liu, L., Hermanns, H. and Hartmanns, A., „Modeling of Networked Automation Systems for simulation and model checking of time behavior,“ In: *International Multi-Conference on Systems, Sygnals & Devices:* IEEE, pp. 1–5, 2012.

[Vog+14a] Vogel-Heuser, B., Schütz, D., Frank, T. and Legat, C., „Model-driven Engineering of Manufacturing Automation Software Projects – A SysML-based Approach,“ In: *Mechatronics, vol.* 24, no. 7, pp. 883–897, 2014.

[Vog+14b] Vogel-Heuser, B., Diedrich, C., Fay, A., Jeschke, S., Kowalewski, S., Wollschlaeger, M. and Göhner, P., „Challenges for Software Engineering in Automation,“ In: *JSEA, vol.* 07, no. 05, pp. 440–451, 2014.

[Vog+14c] Vogel-Heuser, B., Legat, C., Folmer, J. and Feldmann, S., „Researching Evolution in Industrial Plant Automation. Scenarios and Documentation of the Pick and Place Unit,“ Institute of Automation and Information Systems, Technical University of Munich, 2014.

[Vog+14d] Vogel-Heuser, B., Diedrich, C., Pantforder, D. and Göhner, P., „Coupling Heterogeneous Production Systems by a Multi-agent based Cyber-physical Production System,“ In: *2014 12th IEEE International Conference on Industrial Informatics (INDIN):* IEEE, pp. 713–719, 2014.

[Vog+15] Vogel-Heuser, B., Fay, A., Schaefer, I. and Tichy, M., „Evolution of Software in Automated Production Systems: Challenges and research directions,“ In: *J. Syst. Softw., vol.* 110, pp. 54–84, 2015.

[Vog+20] Vogel-Heuser, B., Trunzer, E., Sollfrank, M. and Hujo, D., „(Re-)Deployment of Smart Algorithms in Cyber-Physical Production Systems using DSL4hDNCS,“ In: *Proc. IEEE,* 12 (submitted), 2020.

[VR18] Vogel-Heuser, B. and Ribeiro, L., „Bringing Automated Intelligence to Cyber-Physical Production Systems in Factory Automation,“ In: *14th IEEE International Conference on Automation Science and Engineering (CASE)*. Munich, Germany, pp. 347–352, 2018.

[VWT17] Vogel-Heuser, B., Wildermann, S. and Teich, J., „Towards the Co-evolution of Industrial Products and its Production Systems by Combining Models from Development and Hardware/software Deployment in Cyber-physical Systems,“ In: *Prod. Eng. Res. Devel., vol.* 11, no. 6, pp. 687–694, 2017.

[Vya13] Vyatkin, V., „Software Engineering in Industrial Automation: State-of-the-Art Review,“ In: *IEEE Trans. Ind. Inf., vol.* 9, no. 3, pp. 1234–1249, 2013.

[Wan+15] Wang, Z., Dai, W., Wang, F., Deng, H., Wei, S., Zhang, X. and Liang, B., „Kafka and Its Using in High-throughput and Reliable Message Distribution,“ In: *2015 8th*

International Conference on Intelligent Networks and Intelligent Systems (ICINIS): IEEE, pp. 117–120, 2015.

[WDF18] Witte, M. E., Diedrich, C. and Figalist, H., „Model-based Development in Automation," In: *at - Automatisierungstechnik, vol.* 66, no. 5, pp. 360–371, 2018.

[WMW18] Wolny, S., Mazak, A. and Wally, B., „An Initial Mapping Study on MDE4IoT," In: *Proceedings of MODELS 2018 Workshops: ModComp, MRT, OCL, FlexMDE, EXE, COMMitMDE, MDETools, GEMOC, MORSE, MDE4IoT, MDEbug, MoDeVVa, ME, MULTI, HuFaMo, AMMoRe, PAINS co-located with ACM/IEEE 21st International Conference on Model Driven Engineering Languages and Systems (MODELS 2018), Copenhagen, Denmark, October, 14, 2018*, pp. 524–529, 2018.

[Wol+18] Wolny, S., Mazak, A., Wimmer, M., Konlechner, R. and Kappel, G., „Model-Driven Time-Series Analytics," In: International Journal of Conceptual Modeling, vol. 13, pp. 252–261, 2018.

[Wol+20] Wolny, S., Mazak, A., Wimmer, M. and Huemer, C., „Model-driven Runtime State Identification," In: Mayr, H., Rinderle-Ma, S. and Strecker, S. (Eds.): *40 Years EMISA 2019*. Bonn: Gesellschaft für Informatik e.V, pp. 29–44, 2020.

[Wor+20] Wortmann, A., Barais, O., Combemale, B. and Wimmer, M., „Modeling Languages in Industry 4.0: an Extended Systematic Mapping Study," In: *Softw Syst Model, vol.* 19, no. 1, pp. 67–94, 2020.

[Wor15] World Economic Forum, „Industrial Internet of Things. Unleashing the Potential of Connected Products and Services," 2015.

[WSJ17] Wollschlaeger, M., Sauter, T. and Jasperneite, J., „The Future of Industrial Communication. Automation Networks in the Era of the Internet of Things and Industry 4.0," In: *IEEE Ind. Electron. Mag., vol.* 11, no. 1, pp. 17–27, 2017.

[Wu+14] Wu, X., Zhu, X., Wu, G.-Q. and Ding, W., „Data Mining with Big Data," In: *IEEE Trans. Knowl. Data Eng., vol.* 26, no. 1, pp. 97–107, 2014.

[WV08] Witsch, D. and Vogel-Heuser, B., „Modellierungsansatz für Zeitanforderungen und Kommunikationsnetze," In: *atp edition,* no. 6, pp. 44–52, 2008.

[XD18] Xu, L. D. and Duan, L., „Big Data for Cyber Physical Systems in Industry 4.0. A Survey," In: *Enterp. Inf. Syst., vol.* 35, no. 1, pp. 1–22, 2018.

[YK15] Yin, S. and Kaynak, O., „Big Data for Modern Industry: Challenges and Trends [Point of View]," In: *Proc. IEEE, vol.* 103, no. 2, pp. 143–146, 2015.

[Zha+12] Zhang, L., Luo, Y., Tao, F., Li, B. H., Ren, L., Zhang, X., Guo, H., Cheng, Y., Hu, A. and Liu, Y., „Cloud Manufacturing. A New Manufacturing Paradigm," In: *Enterp. Inf. Syst., vol.* 8, no. 2, pp. 167–187, 2012.

11. List of Figures

12. List of Tables

13. List of References to the Application Example

Parts

Figures

14. List of Abbreviations

Abbreviation	Description
AADL	Architecture Analysis and Design Language
ADS	Automation Device Specification
AE	Application example
AES	Advanced Encryption Standard
AML	Automation Markup Language
AMQP	Advanced Message Queuing Protocol
ANSI	American National Standards Institute
API	Application programming interface
App	Application
aPS	automated Production System
ATL	ATLAS Transformation Language
AutomationML	Automation Markup Language
BaSys4.0	Project “Basissystem Industrie 4.0”
BC	Bus coupler
Bins	Binaries
CAN	Controller Area Network
CAx	Computer-aided x
CI	Continuous integration
CoAP	Constrained Application Protocol
CNC	Computerized numerical control
COCOMO	Constructive Cost Model
COM	Component Object Model
CPPS	Cyber-physical Production Systems
CPS	Cyber-physical Systems
CPSoS	Cyber-physical Systems of Systems
CPU	Central processing unit
DCS	Decentralized control system
DDS	Data Distribution Service
DFD	Data flow diagram
DIN	Deutsches Institut für Normung
DLL	Dynamic-link libraries
DSL	Domain-specific language
DSL4hDNCS	Domain-specific language for heterogeneous distributed networked control systems
EMF	Eclipse Modeling Framework
ERP	Enterprise resource planning
ESB	Enterprise Service Bus

Abbreviation	Description
EtherCAT	Ethernet for Control Automation Technology
FIPA	Foundation for Intelligent Physical Agents
GPML	General-purpose modeling language
gRPC	gRPC Remote Procedure Calls
H	Hypothesis
HART	Highway Addressable Remote Transducer
HMI	Human-machine interface
HTTP	Hypertext Transfer Protocol
I 4.0	Industrie 4.0
I/O	In-/output
IDE	Integrated development environment
IDL	Interface description language
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IETC	Internet Engineering Task Force
IIoT	Industrial Internet of Things
IIRA	Industrial Internet Reference Architecture
IMPROVE	Project “Innovative modeling approaches for production systems to raise validatable efficiency”
IoT	Internet of Things
IoT RA	Internet of Things Reference Architecture
IP	Internet Protocol
IPC	Industrial PC
ISA	International Society of Automation
ISO	International Organization for Standardization
IT	Information technology
KPI	Key performance indicator
Libs	Libraries
LoC	Lines of code
LwM2M	Lightweight Machine to Machine
M+O	Monitoring and optimization (from NOA)
M2M	Model-to-model
M2T	Model-to-text
MARTE	Modeling and Analysis of Real Time and Embedded Systems
MDA	Model-driven Architecture
MDD	Model-driven development
MES	Manufacturing execution system
Modbus/RTU	Modbus Remote Terminal Unit
Modbus/TCP	Modbus Transmission Control Protocol

Abbreviation	Description
MOF	Meta Object Facility
MOFM2T	MOF Model to Text Transformation Language
MPS	Modular production system
MQTT	Message Queuing Telemetry Transport
MSB	Manufacturing Service Bus
MTC	MTConnect Institute
NAMUR	Interessengemeinschaft Automatisierungstechnik der Prozessindustrie
NOA	NAMUR Open Architecture
OCL	Object Constraint Language
OEM	Original equipment manufacturer
OMG	Object Management Group
OPC	Open Platform Communications
OPC DA	OPC Data Access
OPC UA	OPC Unified Architecture
OS	Operating system
OSI	Open Systems Interconnection
OT	Operational technology
P2P	Peer-to-peer
PC	Personal computer
PCM	Palladio Component Model
PERFoRM	Project “Production harmonized reconfiguration of flexible robots and machinery”
PLC	Programmable logic controller
PROFIBUS	Process Field Bus
PROFIBUS DP	Process Field Bus Decentralized Automation
PROFIBUS PA	Process Field Bus Process Automation
PROFINET	Process Field Net
PS	Publish-subscribe
QoS	Quality of Service
QVT	Query View Transformation
RAM	Random-access memory
RAMI	Reference Architecture Model Industrie 4.0
Req	Requirement
Req-A	Requirements of category “data collection system architecture”
Req-G	Requirements of category “model-driven generation of data collection architectures”
Req-M	Requirements of category “domain-specific language for architecture modeling”
Req-SF	Requirements of category “interoperability of systems and architecture software framework”
REST	Representational State Transfer
RFC	Request for Comments (from IETF)

Abbreviation	Description
RFID	Radio-frequency identification
ROOM	Real-Time Object-Oriented Modeling
RPC	Remote procedure calls
RR	Request-response
SA	Structured analysis
SA/RT	SA for real-time systems
SAE	Society of Automotive Engineers
SCADA	Supervisory control and data acquisition
SIDAP	Project “Scalable integration concept for data aggregation, analysis and preparation of big data volumes in process industry”
SME	Small and medium-sized enterprises
SoaML	Service-oriented architecture Modeling Language
SOAP	Simple Object Access Protocol
SysML	Systems Modeling Language
SysML-vAT	SysML for distributed automation systems
T2M	Text-to-model
TCP	Transmission Control Protocol
TSN	Time-Sensitive Networking
UADP	UA Datagram Protocol (from OPC UA)
UDP	User Datagram Protocol
UID	Unique identifier
UML	Unified Modeling Language
UML-PA	UML Process Automation
UML-RT	UML for Real Time
VAB	Virtual Automation Bus
VDE	Verband der Elektrotechnik, Elektronik und Informationstechnik e.V.
VDI	Verein Deutscher Ingenieure e.V.
VLAN	Virtual Local Area Network
VM	Virtual machine
VSB	eVolution Service Bus
WLAN	Wireless Local Area Network
xADL	extensible Architecture Description Language
XML	Extensible Markup Language
xPPU	Extended Pick and Place Unit
XSB	Extensible service bus

Appendix A. Graphical Models of Use-Cases B to D

In the following Appendix the graphical models related to the industrial Use-Cases B to C from Section 7.2 can be found.

Appendix A.1 Use-Case B Anomaly Detection

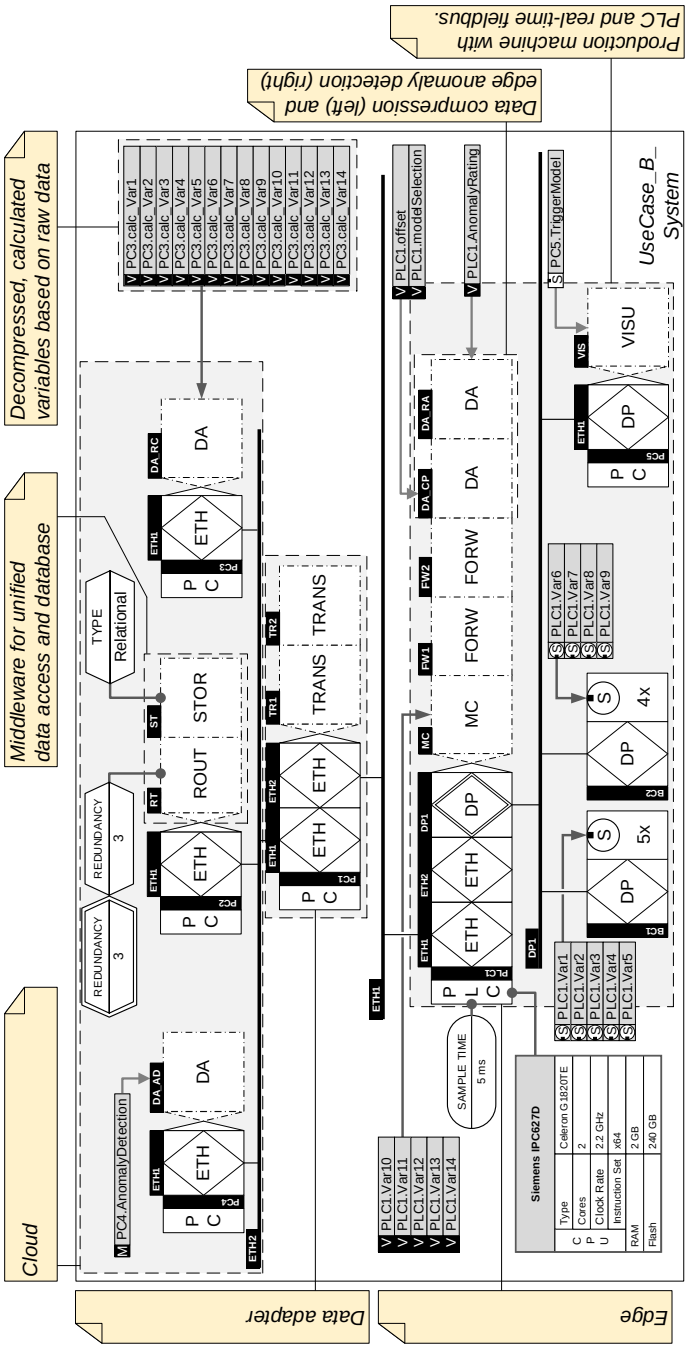


Figure 55: Combined edge and cloud architecture (Use-Case B) in the system viewpoint (adapted from [TWV20]). Production machine with anomaly detection on edge level, cloud environment for model training, and data adapter in between to translate protocol and semantics.

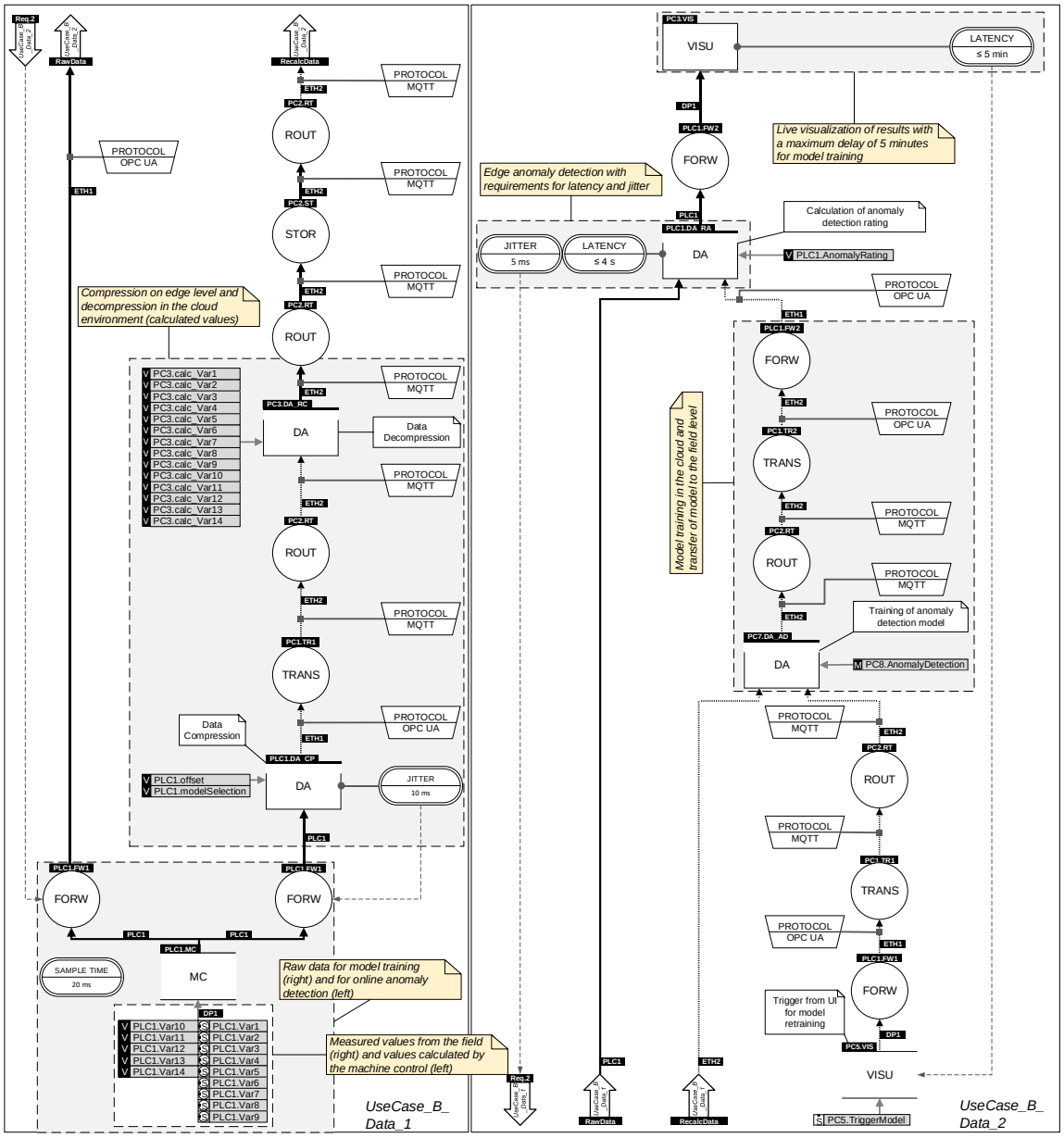


Figure 56: Data flow of Use-Case B modeled in the data flow viewpoint (adapted from [TWV20]). The diagram is distributed over two sheets for better overview, arrows link the two sheets.

Appendix A.2 Use-Case C Alarm Management

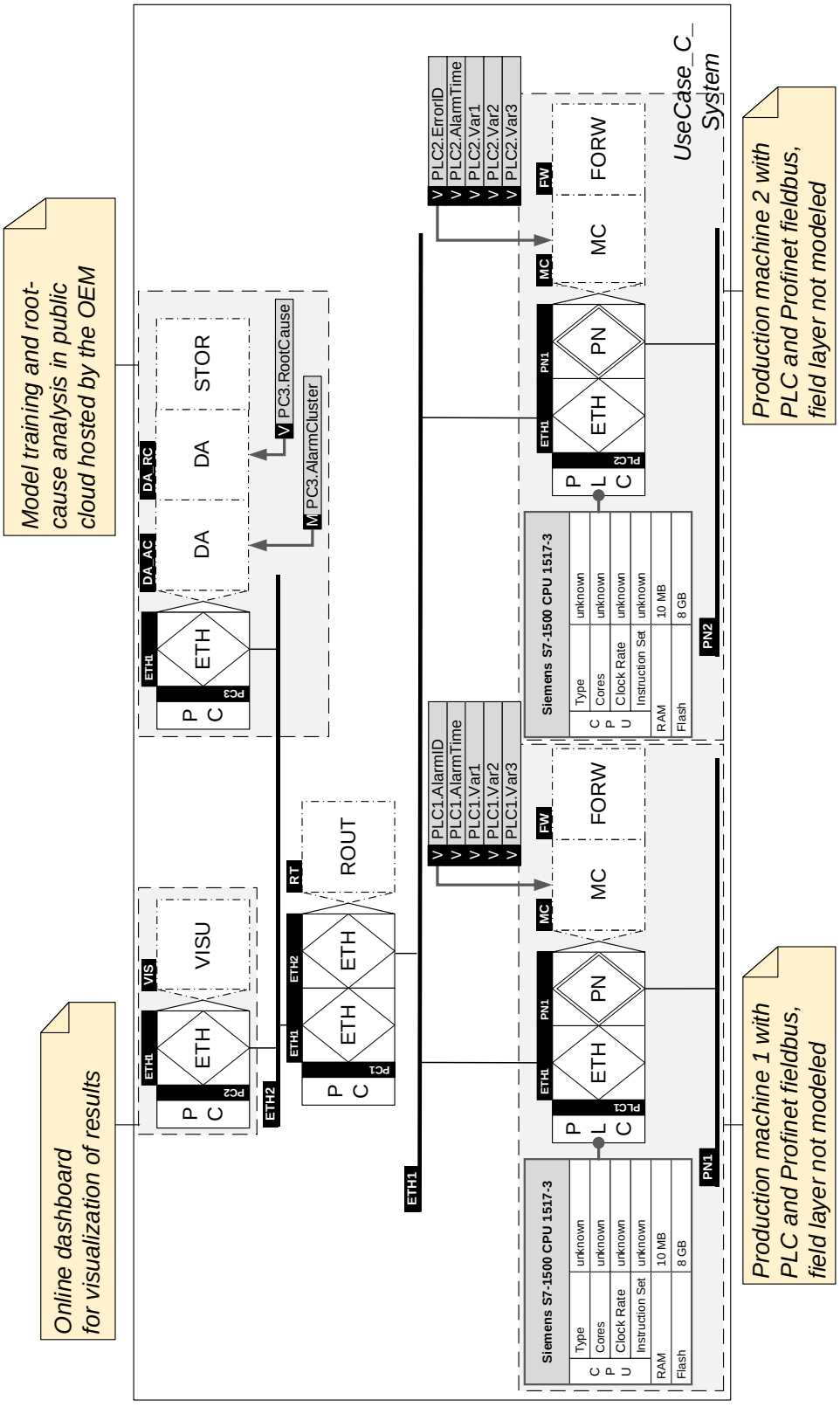


Figure 57: Public cloud architecture for alarm analysis and management (Use-Case C) modeled in the system viewpoint (adapted from [TWV20]). Two production machines connected to an online dashboard and analysis solution hosted in a public cloud.

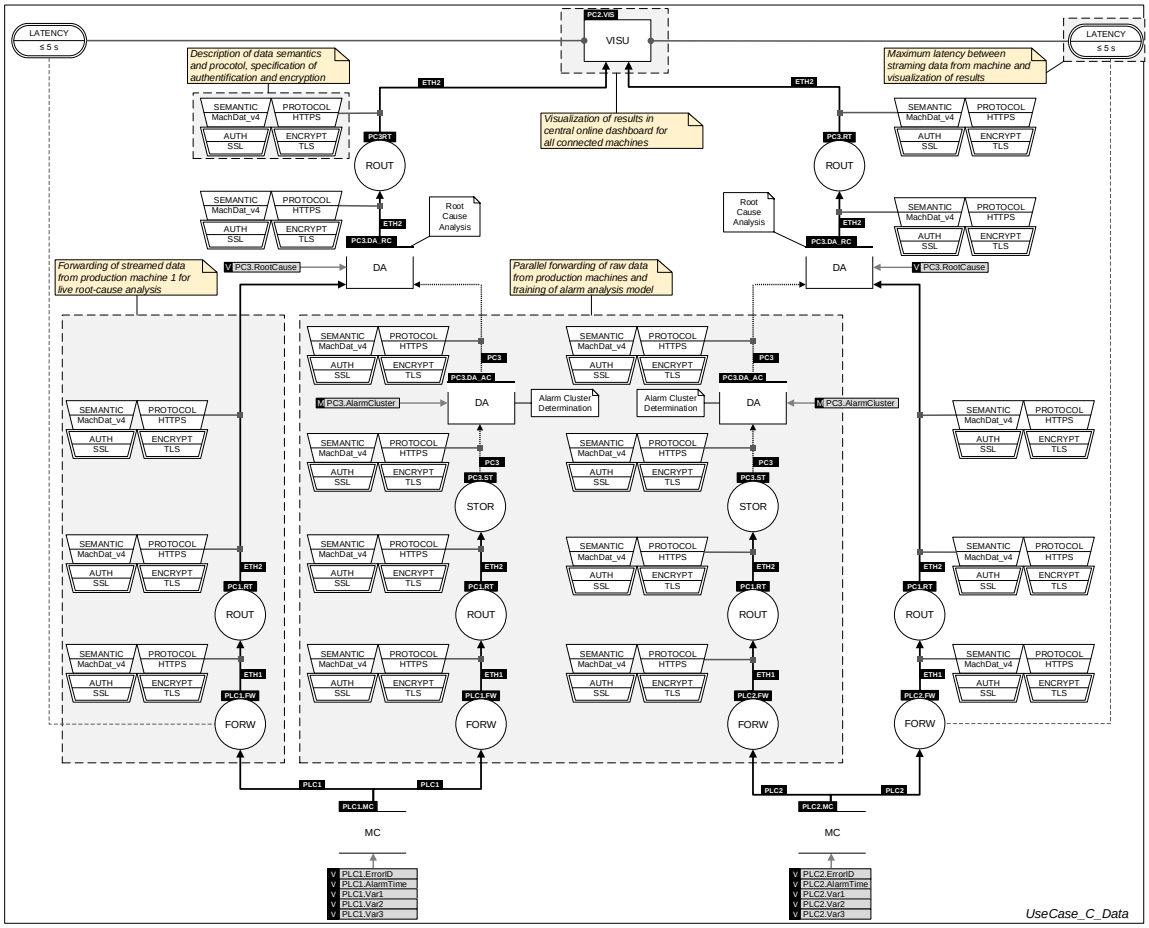


Figure 58: Data flow of Use-Case C covering the alarm analysis and management in a public cloud (adapted from [TWV20]). Differentiation between the collection of historic data for model training (center) and the collection of streamed data for live root-cause analysis (left and right).

Appendix A.3 Use-Case D Alarm Management

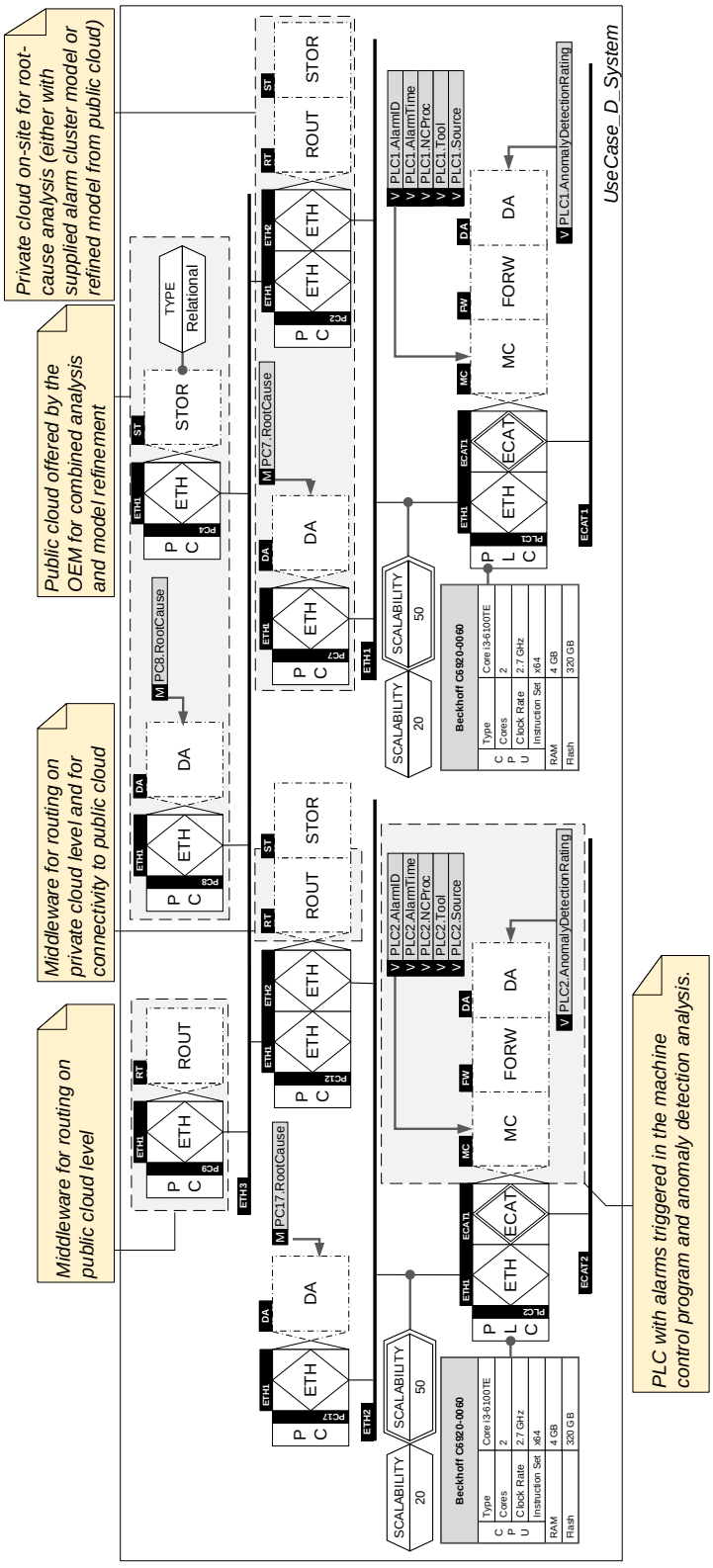


Figure 59: Alarm management system hosted private and public clouds of Use-Case D modeled in the system viewpoint (adapted from [TWV20]). Hybrid cloud setup to guarantee confidentiality to customers and increase performance of the analysis, while allowing a fleet-management across multiple clients.

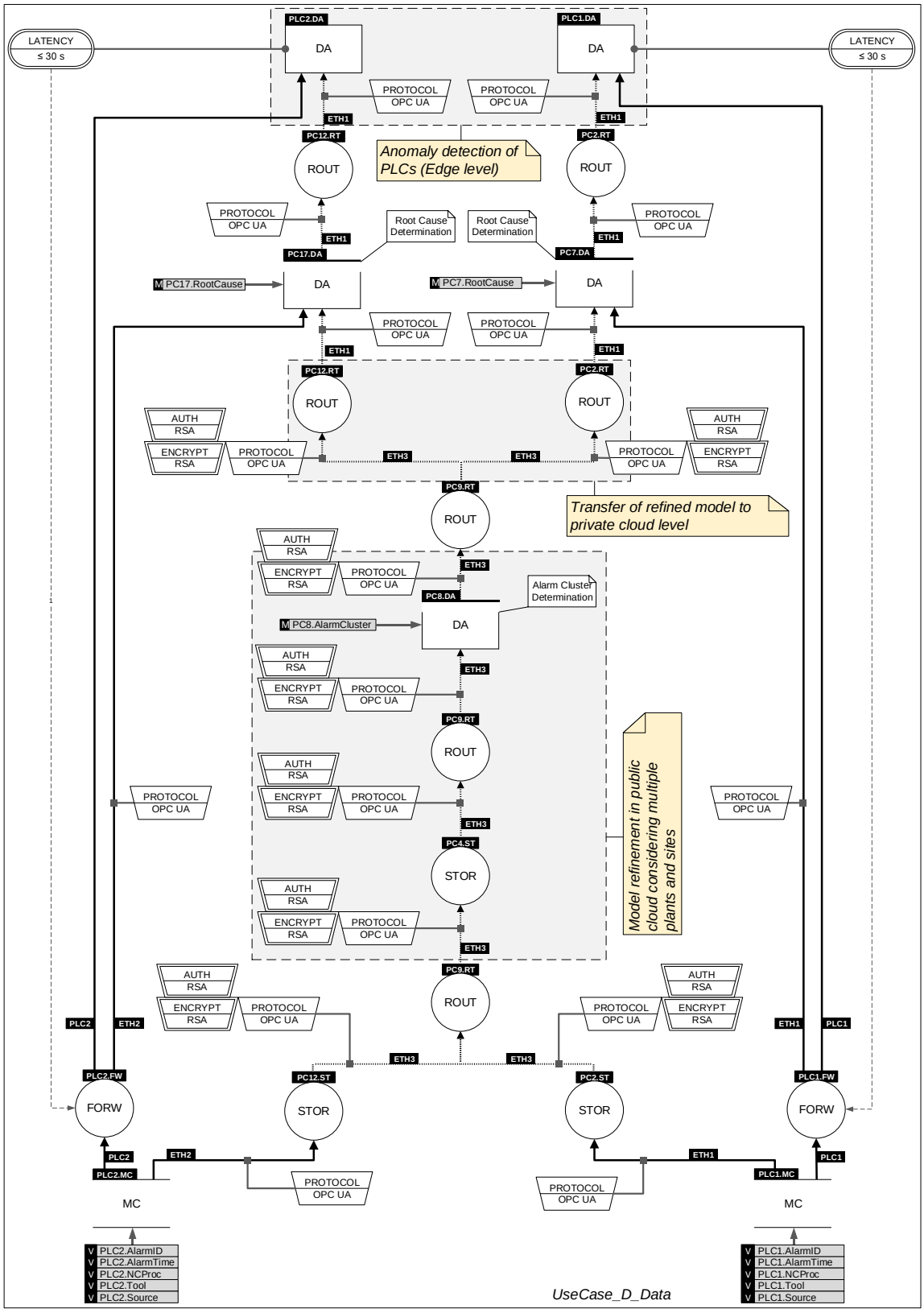


Figure 60: Alarm management system of Use-Case D modeled in the data flow viewpoint (adapted from [TWV20]). Model refinement based on data from multiple sources (center) and subsequent execution of pre-trained models to diagnose machines.

Appendix B. Graphical Models of Lab-scale Study

All graphical models related to the lab-scale feasibility study from Section 7.3 are given in this Chapter. Figures 61 to 63 contain the system diagrams, while Figures 64 to 67 the corresponding data flow diagrams.

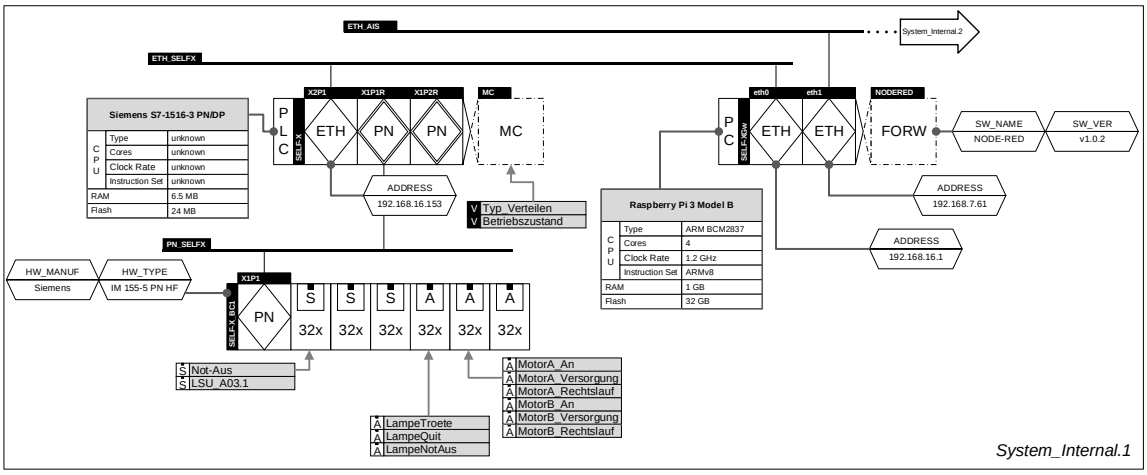


Figure 61: First sheet of the system diagram of the internal feasibility study.

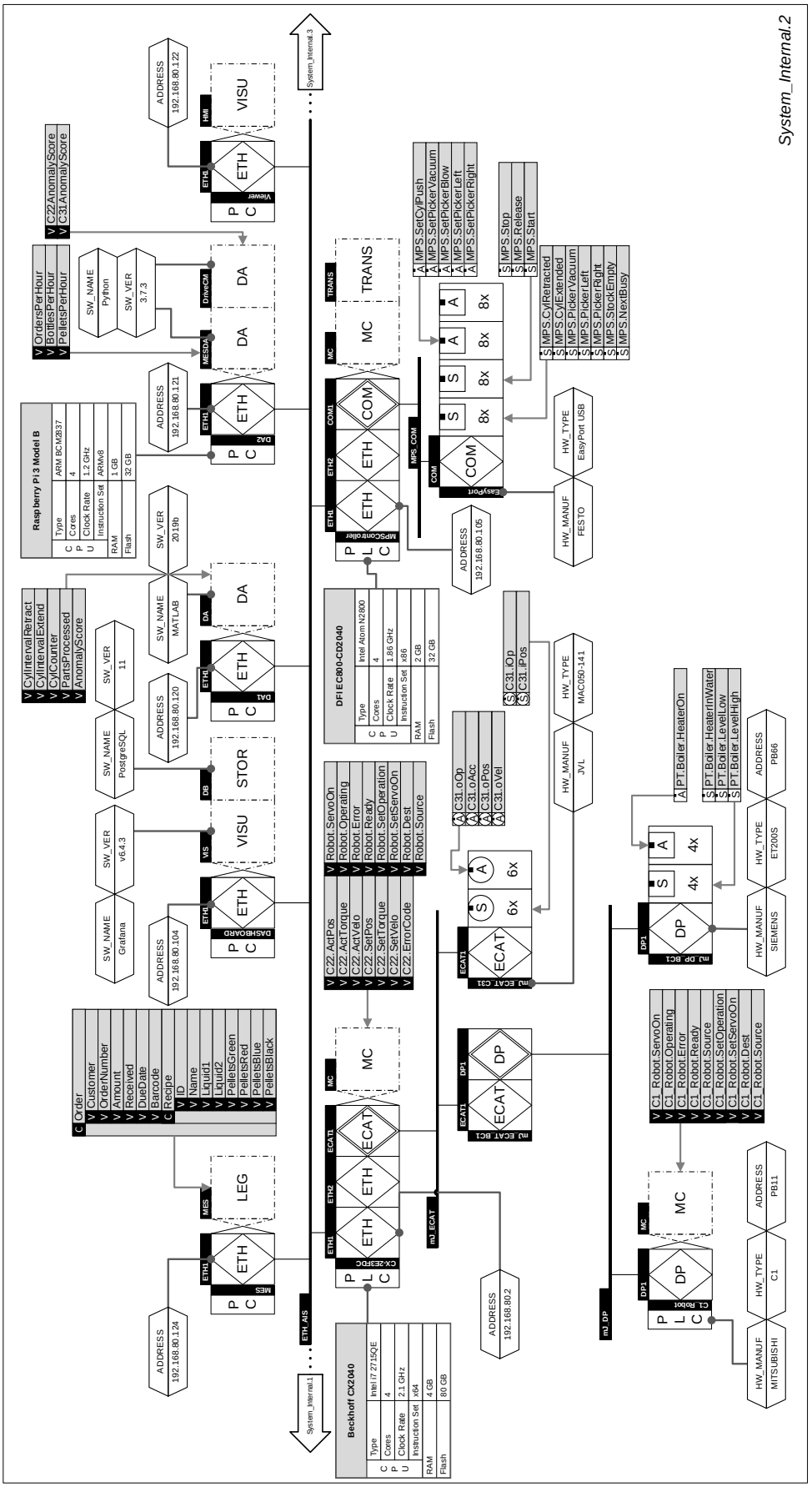


Figure 62: Second sheet of the system diagram of the internal feasibility study.

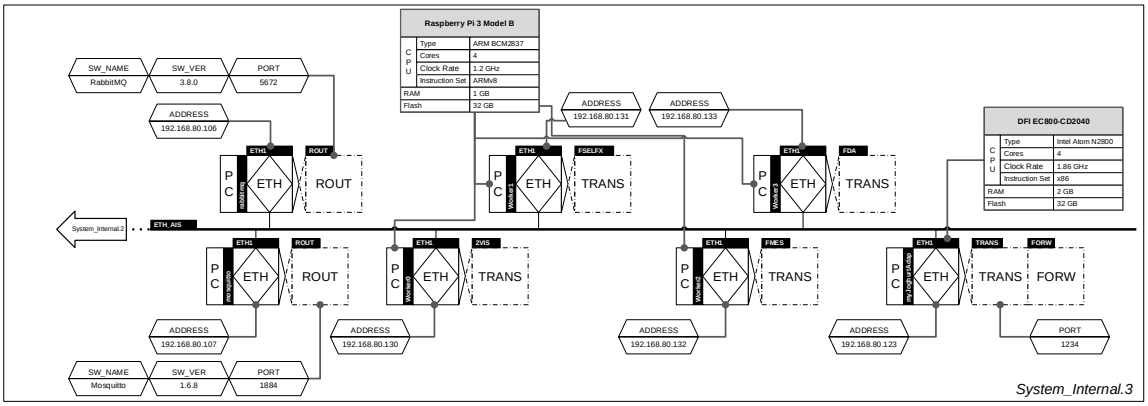


Figure 63: Third sheet of the system diagram of the internal feasibility study.

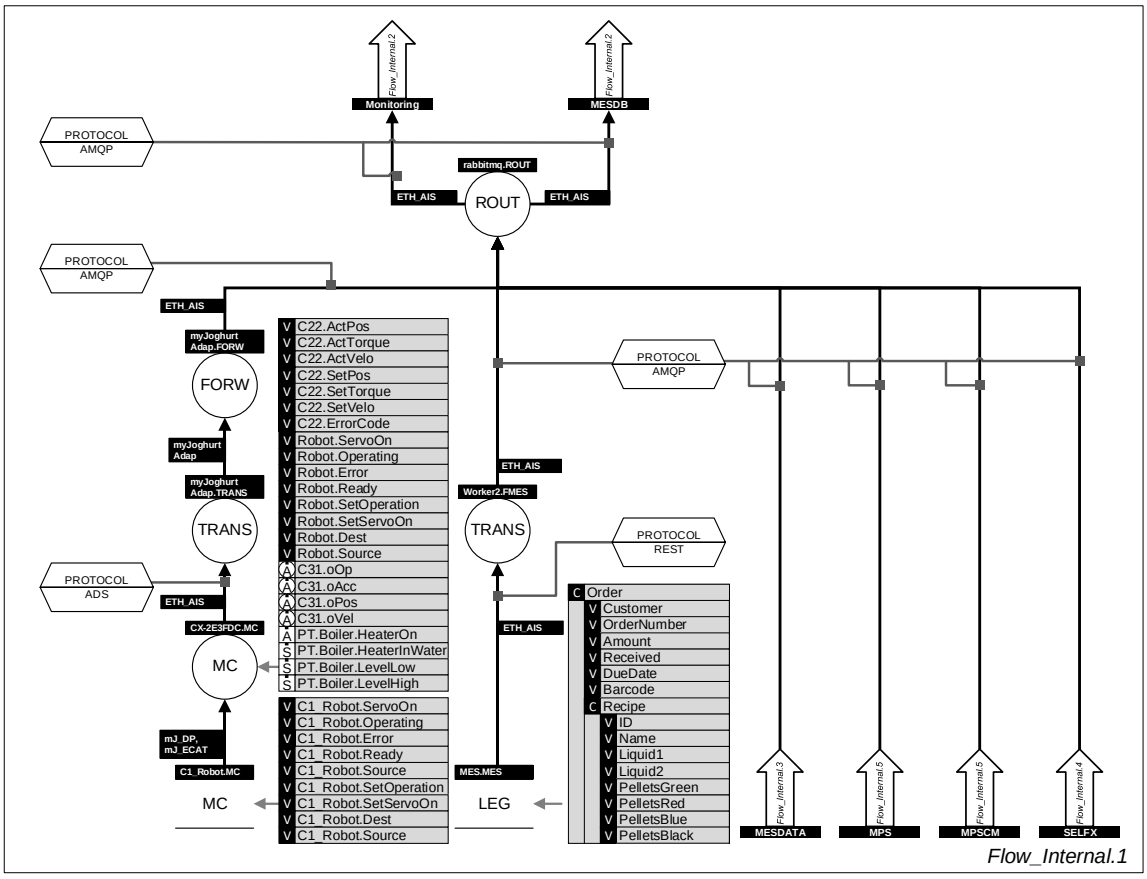


Figure 64: First sheet of the data flow diagram of the internal feasibility study.

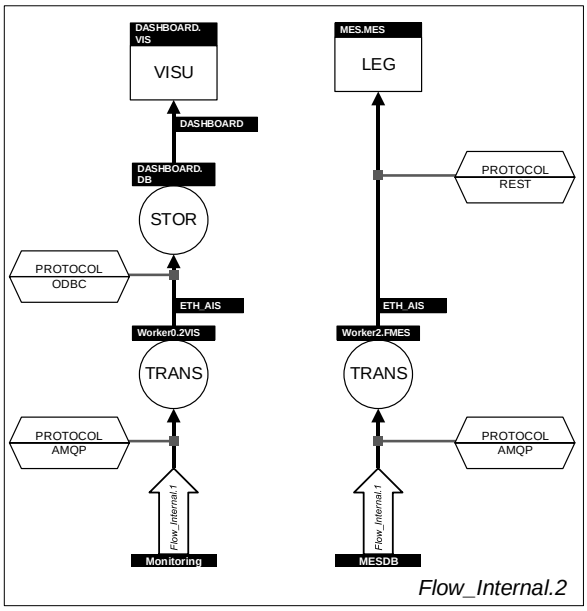


Figure 65: Second sheet of the data flow diagram of the internal feasibility study.

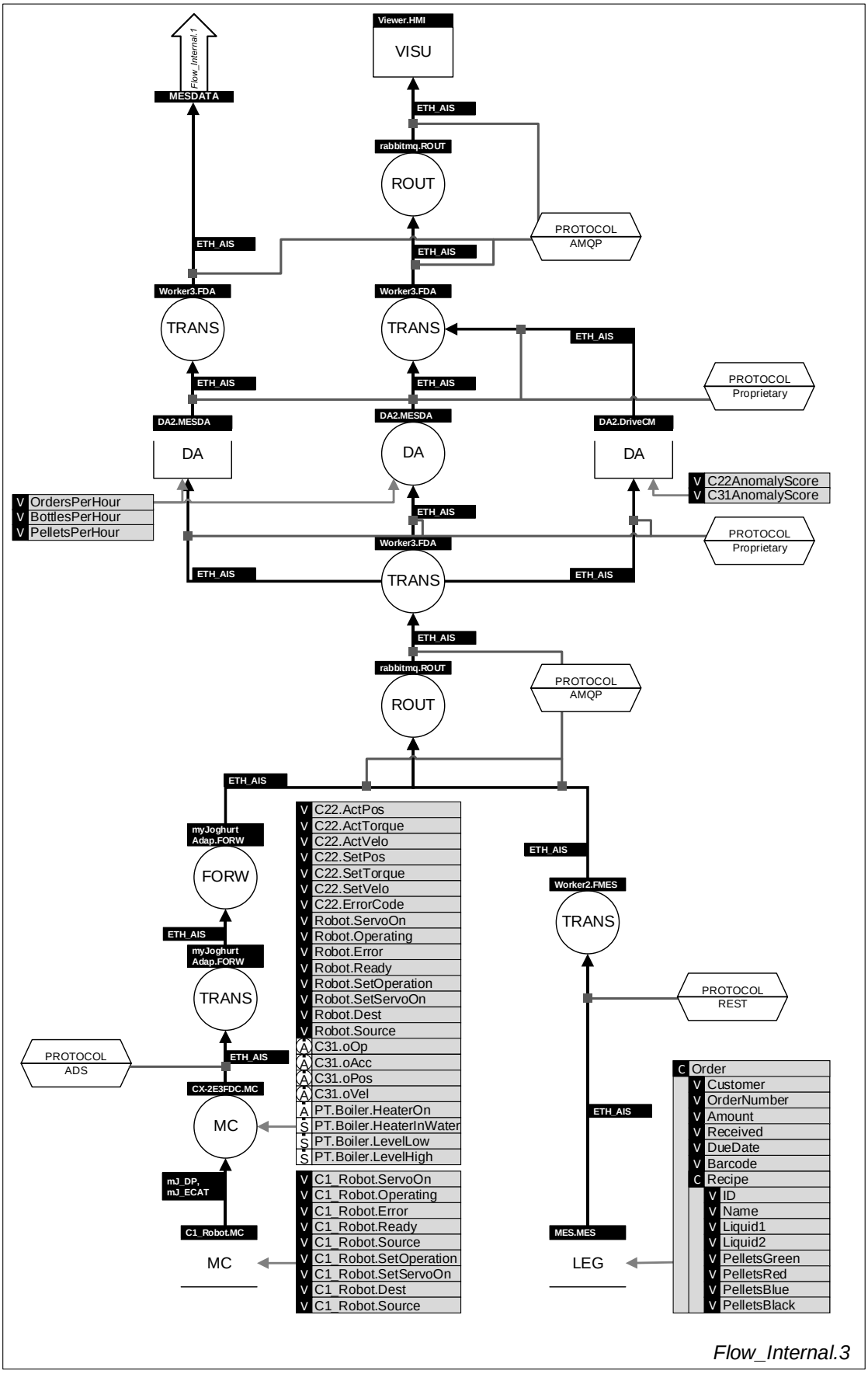


Figure 66: Third sheet of the data flow diagram of the internal feasibility study.

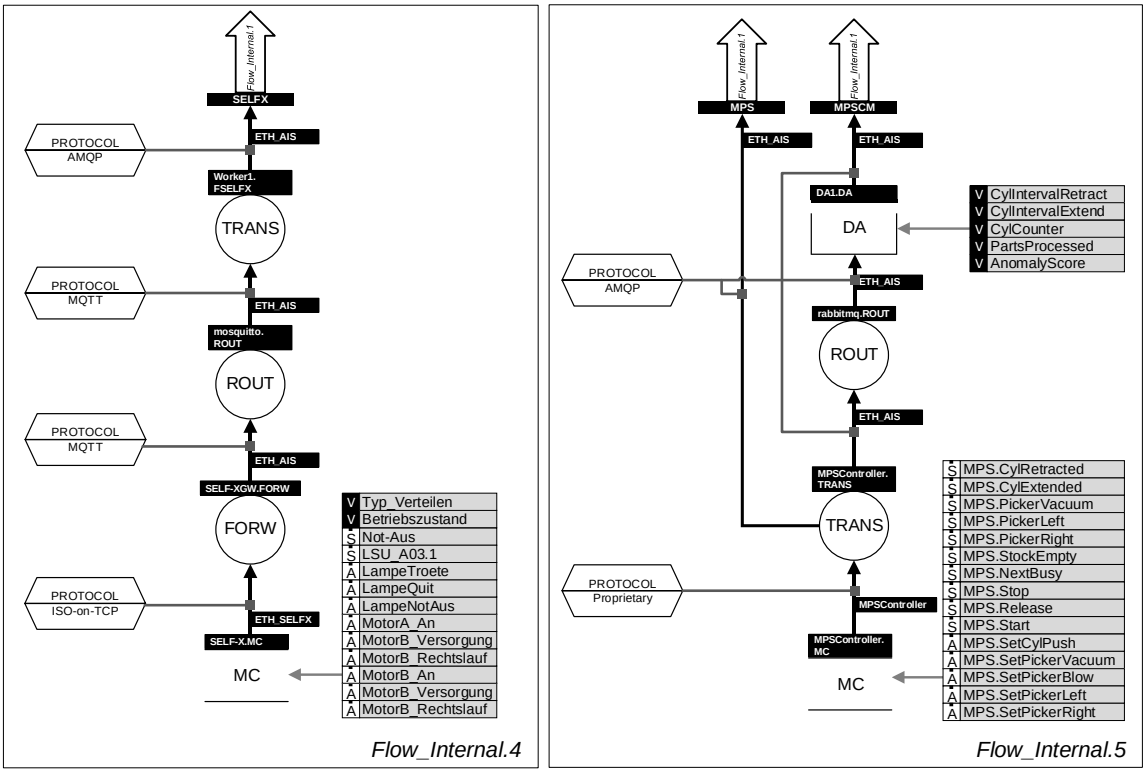


Figure 67: Fourth and fifth sheets of the data flow diagram of the internal feasibility study.

Appendix C. Code Snippets Extrapolation Case-Study

The following Chapter lists the source codes for the minimal publishers and subscriber functionalities for the extrapolation case-study (cf. Section 7.5). The lines of code (LoC) metrics were evaluated in Visual Studio 2019 using the Microsoft.CodeAnalysis.Metrics package in version 2.9.8 [Mic19c]. The analysis counts all lines of code for the implementation of the respective classes including comments and empty lines. The using directives at the top of each listing, the namespace declaration, as well as the surrounding brackets of the namespace are not counted. Due to width limitations of this printed document, some additional line breaks were introduced but do not influence the LoC metric. The line numbers on the left of each listing reflect this introduction of arbitrary line breaks by not counting these additional lines. Lines of codes in the captions reflect raw, uncorrected numbers directly from the code metric analysis.

Listing 1: Minimal publisher for AMQP protocol (LoC = 28).

```
 1 using System;
 2 using RabbitMQ.Client;
 3 using System.Text;
 4 namespace MinimalExample
 5 {
 6     public class PublisherManualAmqpMinClient
 7     {
 8         public static void Main()
 9         {
10             var client = new PublisherManualAmqpMin("192.168.80.214", 5672,
                                    "SimplPub", "SimplePass");
11             client.TransmitData("TestByte", 127);
12         }
13     }
14     public class PublisherManualAmqpMin
15     {
16         private IModel _Channel;
17         public PublisherManualAmqpMin(string host, uint port, string user, string password)
18         {
19             var factory = new ConnectionFactory
20             {
21                 HostName = host,
22                 Port = (int)port,
23                 UserName = user,
24                 Password = password
25             };
26             _Channel = factory.CreateConnection().CreateModel();
27         }
28         public void TransmitData(string channel, object data)
29         {
30             _Channel.QueueDeclare(channel, false, false, false, null);
31             _Channel.BasicPublish("", channel, null,
                                    Encoding.UTF8.GetBytes(data.ToString()));
```

```
        }
    }
}
```

Listing 2: Minimal publisher for Beckhoff ADS protocol (LoC = 30).

```
using System;
using TwinCAT.Ads;
using System.Collections.Generic;
namespace MinimalExampleAds
{
    public class PublisherManualAdsClient
    {
        public static void Main()
        {
            var client = new PublisherManualAdsMin("5.46.63.220.1.1", 851);
            client.TransmitData("TestByte", 127);
        }
    }
    public class PublisherManualAdsMin
    {
        private readonly TcAdsClient _client;
        private Dictionary<string, int> _knownHandles;
        public PublisherManualAdsMin(string amsNetId, uint port)
        {
            _client = new TcAdsClient();
            _client.Connect(new AmsAddress(amsNetId + ":" + port));
            _knownHandles = new Dictionary<string, int>();
        }
        public void TransmitData<T>(string channel, T data)
        {
            int handle;
            if(_knownHandles.TryGetValue(channel, out var h)) handle = h;
            else
            {
                handle =  _client.CreateVariableHandle(channel);
                _knownHandles.Add(channel, handle);
            }
            _client.WriteAny(handle, data);
        }
    }
}
```

Listing 3: Minimal publisher for Kafka protocol (LoC = 27).

```
using System;
using Confluent.Kafka;
namespace MinimalExample
{
    public class PublisherManualKafkaMinClient
    {
        public static void Main()
        {
            var client = new PublisherManualKafkaMin("192.168.80.214", 1883,
```

```
                                    "SimplPub", "SimplePass");
10              client.TransmitData("TestByte", 127);
11          }
12      }
13      public class PublisherManualKafkaMin
14      {
15          private IProducer<Ignore, string> _client;
16          public PublisherManualKafkaMin(string host, uint port, string user, string password)
17          {
18              var conf = new ProducerConfig
19              {
20                  BootstrapServers = host + ":" + port,
21                  SaslUsername = user,
22                  SaslPassword = password,
23                  SecurityProtocol = SecurityProtocol.SaslPlaintext
24              };
25              _client = new ProducerBuilder<Ignore, string>(conf).Build();
26          }
27          public void TransmitData(string channel, object data)
28          {
29              _client.Produce(channel, new Message<Ignore, string> {Value = data.ToString()});
30          }
31      }
32  }
```

Listing 4: Minimal publisher for MQTT protocol (LoC = 21).

```
 1  using System;
 2  using OpenNETCF.MQTT;
 3  namespace MinimalExample
 4  {
 5      public class PublisherManualMqttMinClient
 6      {
 7          public static void Main()
 8          {
 9              var client = new PublisherManualMqttMin("192.168.80.214", 1883,
                                    "SimplPub", "SimplePass");
10              client.TransmitData("TestByte", 127);
11          }
12      }
13      public class PublisherManualMqttMin
14      {
15          private MQTTClient _client;
16          public PublisherManualMqttMin(string host, uint port, string user, string password)
17          {
18              _client = new MQTTClient(host, (int)port);
19              _client.Connect("SimplePub", user, password);
20          }
21          public void TransmitData(string channel, object data)
22          {
23              _client.Publish(channel, data.ToString(), QoS.FireAndForget, false);
24          }
25      }
26  }
```

Listing 5: Minimal publisher for OPC UA protocol (LoC =53).

```
 1 using System;
 2 using Opc.Ua.Client;
 3 using Opc.Ua;
 4 namespace MinimalExample
 5 {
 6     public class PublisherManualOpcUaMinClient
 7     {
 8         public static void Main()
 9         {
10             var client = new PublisherManualOpcUaMin("192.168.80.215", 5672);
11             client.TransmitData("ns=2;s=TestByte", 127);
12         }
13     }
14     class PublisherManualOpcUaMin
15     {
16         private Session m_session;
17         public PublisherManualOpcUaMin(string host, uint port)
18         {
19             var opcClientConfig = new ApplicationConfiguration()
20             {
21                 ApplicationName = "OPC UA Data Adapter Pub",
22                 ApplicationType = ApplicationType.Client,
23                 ApplicationUri = "urn:" + Utils.GetHostName() + ":AIS:DataAdapterPub",
24                 SecurityConfiguration = new SecurityConfiguration()
25                 {
26                     ApplicationCertificate = new CertificateIdentifier()
27                     {
28                         StoreType = CertificateStoreType.Directory,
29                         StorePath = "OPC_UA_DataAdapter_Pub\\UA_MachineDefault",
30                         SubjectName = "OPA UA Data Adapter",
31                     },
32                     TrustedPeerCertificates = new CertificateTrustList()
33                     {
34                         StoreType = CertificateStoreType.Directory,
35                         StorePath = "OPC_UA_DataAdapter_Pub\\UA_Applications"
36                     }
37                 },
38                 ClientConfiguration = new ClientConfiguration()
39             };
40             opcClientConfig.Validate(ApplicationType.Client).Wait();
41             var serverEndpoint = CoreClientUtils.SelectEndpoint
                        ("opc.tcp://" + host + ":" + port, false);
42             var server = new ConfiguredEndpoint
                        (serverEndpoint.Server, EndpointConfiguration.Create(opcClientConfig));
43             server.Update(serverEndpoint);
44             m_session = Session.Create
                        (opcClientConfig, server, true, opcClientConfig.ApplicationName,
                         3600, new UserIdentity(new AnonymousIdentityToken()), null).Result;
45         }
46         public void TransmitData(string channel, object data)
47         {
48             WriteValue valueToWrite = new WriteValue
49             {
```

```
50                 NodeId = channel,
51                 AttributeId = Attributes.Value
52             };
53             valueToWrite.Value.Value = data;
54             valueToWrite.Value.SourceTimestamp = DateTime.Now;
55             var valuesToWrite = new WriteValueCollection { valueToWrite };
56             m_session.Write(null, valuesToWrite, out _, out _);
57         }
58     }
59 }
```

Listing 6: Minimal subscriber for protocol AMQP (LoC = 43).

```
 1 using System;
 2 using System.Text;
 3 using RabbitMQ.Client;
 4 using RabbitMQ.Client.Events;
 5 namespace MinimalExample
 6 {
 7     public class SubscriberManualAmqpMinClient
 8     {
 9         public static void Main()
10         {
11             var client = new SubscriberManualAmqpMin("192.168.80.215", 5672,
                                    "SimplePub", "SimplePass");
12             client.Subscribe("TestByte", ReceivedHandler);
13             Console.ReadLine();
14         }
15         public static void ReceivedHandler(string message)
16         {
17         }
18     }
19     public class SubscriberManualAmqpMin
20     {
21         private readonly IModel _Channel;
22         public delegate void ReceivedHandler(string message);
23         public event ReceivedHandler TestByteReceived;
24         public SubscriberManualAmqpMin(string host, uint port, string user, string password)
25         {
26             var factory = new ConnectionFactory
27             {
28                 HostName = host,
29                 Port = (int)port,
30                 UserName = user,
31                 Password = password
32             };
33             _Channel = factory.CreateConnection().CreateModel();
34         }
35         public void Subscribe(string channel, ReceivedHandler handler)
36         {
37             _Channel.QueueDeclare(channel, false, false, false, null);
38             _Channel.QueueBind(channel, "", channel);
39             var consumer = new EventingBasicConsumer(_Channel);
40             consumer.Received += (model, ea) =>
```

```
41            {
42                if (ea.RoutingKey != channel) return;
43                var body = ea.Body;
44                var message = Encoding.UTF8.GetString(body);
45                handler(message);
46            };
47            _Channel.BasicConsume(channel, true, consumer);
48        }
49    }
50 }
```

Listing 7: Minimal subscriber for Beckhoff ADS protocol (LoC = 55).

```
 1 using System;
 2 using System.Collections.Generic;
 3 using System.Text;
 4 using TwinCAT.Ads;
 5 namespace MinimalExampleAds
 6 {
 7     public class SubscriberManualAdsClient
 8     {
 9         public static void Main()
10         {
11             var client = new SubscriberManualAdsMin("5.46.63.220.1.1", 851);
12             client.Subscribe<byte>("TestByte", ReceivedHandler);
13             Console.ReadLine();
14         }
15         public static void ReceivedHandler(string message)
16         {
17         }
18     }
19     public class SubscriberManualAdsMin
20     {
21         public delegate void ReceivedHandler(string message);
22         private readonly TcAdsClient _client;
23         private readonly Dictionary<uint, AdsNotificationEventHandler> _subscriptions;
24         private readonly Dictionary<string, int> _knownVariableHandles;
25         public event ReceivedHandler TestByteReceived;
26         public SubscriberManualAdsMin(string amsNetId, uint port)
27         {
28             _subscriptions = new Dictionary<uint, AdsNotificationEventHandler>();
29             _knownVariableHandles = new Dictionary<string, int>();
30             _client = new TcAdsClient();
31             _client.Connect(new AmsAddress(amsNetId + ":" + port));
32             _client.AdsNotification += (s, e) =>
33             {
34                 if (_subscriptions.TryGetValue((uint)e.NotificationHandle, out var handler))
35                     handler.Invoke(s, e);
36             };
37         }
38         private int GetVariableHandle(string varName)
39         {
40             if (_knownVariableHandles.TryGetValue(varName, out var handle)) return handle;
41             else
```

```
42            {
43                var newHandle = _client.CreateVariableHandle(varName);
44                _knownVariableHandles.Add(varName, newHandle);
45                return newHandle;
46            }
47        }
48        public void Subscribe<T>(string channel, ReceivedHandler handler)
49        {
50            var settings = new NotificationSettings(AdsTransMode.OnChange, 10, 20);
51            var errorCode = _client.TryAddDeviceNotification(channel, new AdsStream(),
                                    0, 40, settings, null, out uint handle);
52            if (errorCode != AdsErrorCode.NoError)
                      throw new Exception("subscription failed with error code" + errorCode);
53            _subscriptions.Add(handle, (s, e) =>
54            {
55                object value;
56                if (typeof(T) == typeof(string)) value =
                      _client.ReadAnyString(GetVariableHandle(channel), 80, Encoding.Default);
57                else value = _client.ReadAny(GetVariableHandle(channel), typeof(T));
58                handler.Invoke(value.ToString());
59            });
60        }
61    }
62 }
```

Listing 8: Minimal subscriber for Kafka protocol (LoC = 51).

```
 1 using Confluent.Kafka;
 2 using System;
 3 using System.Collections.Generic;
 4 using System.Threading;
 5 namespace MinimalExample
 6 {
 7     public class SubscriberManualKafkaMinClient
 8     {
 9         public static void Main()
10         {
11             var client = new SubscriberManualKafkaMin("192.168.80.216", 9092,
                                    "SimplePub", "SimplePass");
12             client.Subscribe("TestByte", ReceivedHandler);
13             Console.ReadLine();
14         }
15         public static void ReceivedHandler(string message)
16         {
17         }
18     }
19     public class SubscriberManualKafkaMin
20     {
21         public delegate void ReceivedHandler(string message);
22         private IConsumer<Ignore, string> _client;
23         private Dictionary<string, ReceivedHandler> _channels;
24         public event ReceivedHandler TestByteReceived;
25         public SubscriberManualKafkaMin(string host, uint port, string user, string password)
26         {
```

```
27              var conf = new ConsumerConfig
28              {
29                  GroupId = "AIS",
30                  BootstrapServers = host + ":" + port,
31                  SaslUsername = user,
32                  SaslPassword = password,
33                  SecurityProtocol = SecurityProtocol.SaslPlaintext,
34              };
35              _client = new ConsumerBuilder<Ignore, string>(conf).Build();
36              _channels = new Dictionary<string, ReceivedHandler>();
37              new Thread(Receive).Start();
38          }
39          public void Subscribe(string channel, ReceivedHandler handler)
40          {
41              _client.Subscribe(channel);
42              _channels.Add(channel, handler);
43          }
44          private void Receive()
45          {
46              while (true)
47              {
48                  try
49                  {
50                      var res = _client.Consume(TimeSpan.FromMilliseconds(100));
51                      if (!_channels.TryGetValue(res.Topic, out ReceivedHandler handler))
                                        return;
52                      handler.Invoke(res.Message.Value);
53                  }
54                  catch(Exception e) { }
55              }
56          }
57      }
58  }
```

Listing 9: Minimal subscriber for MQTT protocol (LoC = 34).

```
 1 using System;
 2 using OpenNETCF.MQTT;
 3 using System.Collections.Generic;
 4 using System.Text;
 5 namespace MinimalExample
 6 {
 7     public class SubscriberManualMqttMinClient
 8     {
 9         public static void Main()
10         {
11             var client = new SubscriberManualMqttMin("192.168.80.214", 1883,
                                        "SimplPub", "SimplePass");
12             client.Subscribe("TestByte", ReceivedHandler);
13             Console.ReadLine();
14         }
15         public static void ReceivedHandler(string message)
16         {
17         }
```

```
18      }
19      public class SubscriberManualMqttMin
20      {
21          private readonly MQTTClient _client;
22          public delegate void ReceivedHandler(string message);
23          private Dictionary<string, ReceivedHandler> _channels;
24          public event ReceivedHandler TestByteReceived;
25          public SubscriberManualMqttMin(string host, uint port, string user, string password)
26          {
27              _client = new MQTTClient(host, (int)port);
28              _client.Connect("SimplePub", user, password);
29              _client.MessageReceived += (channel, qos, payload) =>
30              {
31                  if (!_channels.TryGetValue(channel, out ReceivedHandler handler)) return;
32                  handler.Invoke(Encoding.UTF8.GetString(payload));
33              };
34              _channels = new Dictionary<string, ReceivedHandler>();
35          }
36          public void Subscribe(string channel, ReceivedHandler handler)
37          {
38              if (!_channels.ContainsKey(channel)) _channels.Add(channel, handler);
39          }
40      }
41  }
```

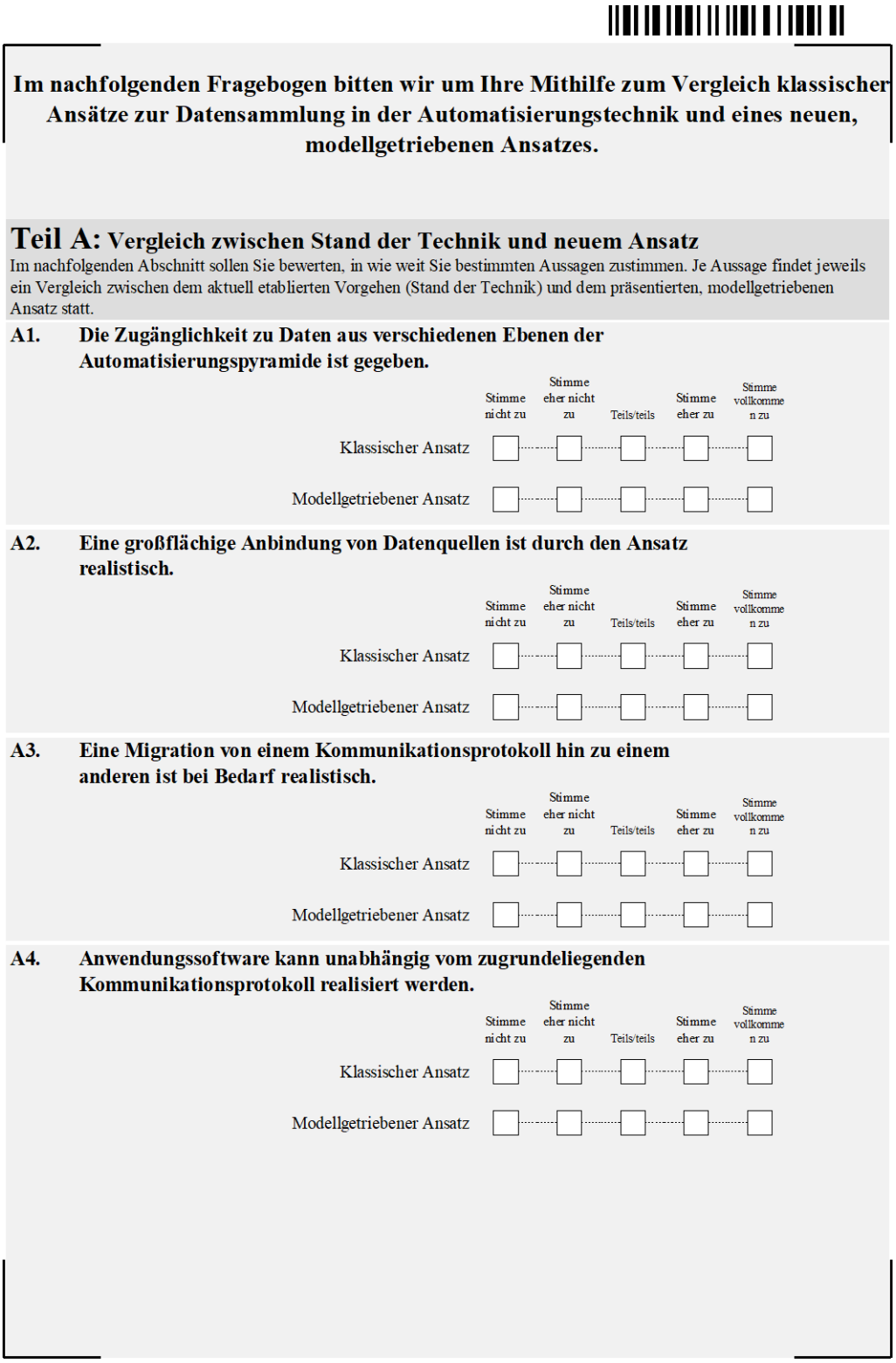
Listing 10: Minimal subscriber for OPC UA protocol (LoC = 65).

```
 1  using System;
 2  using Opc.Ua;
 3  using Opc.Ua.Client;
 4  namespace MinimalExample
 5  {
 6      public class SubscriberManualOpcUaMinClient
 7      {
 8          public static void Main()
 9          {
10              var client = new SubscriberManualOpcUaMin("desktop-o6ueut2", 50000);
11              client.Subscribe("ns=2;s=TestByte", ReceivedHandler);
12              Console.ReadLine();
13          }
14          public static void ReceivedHandler(string message)
15          {
16          }
17      }
18      public class SubscriberManualOpcUaMin
19      {
20          public delegate void ReceivedHandler(string message);
21          private readonly Session m_session;
22          public SubscriberManualOpcUaMin(string host, uint port)
23          {
24              var opcClientConfig = new ApplicationConfiguration()
25              {
26                  ApplicationName = "OPC UA Data Adapter",
27                  ApplicationType = ApplicationType.Client,
```

```
28                 ApplicationUri = "urn:" + Utils.GetHostName() + ":AIS:DataAdapter",
29                 SecurityConfiguration = new SecurityConfiguration()
30                 {
31                     ApplicationCertificate = new CertificateIdentifier()
32                     {
33                         StoreType = CertificateStoreType.Directory,
34                         StorePath = "OPC_UA_DataAdapter\\UA_MachineDefault",
35                         SubjectName = "OPA UA Data Adapter",
36                     },
37                     TrustedPeerCertificates = new CertificateTrustList()
38                     {
39                         StoreType = CertificateStoreType.Directory,
40                         StorePath = "OPC_UA_DataAdapter\\UA_Applications"
41                     }
42                 },
43                 ClientConfiguration = new ClientConfiguration()
44             };
45             opcClientConfig.Validate(ApplicationType.Client).Wait();
46             var serverEndpoint = CoreClientUtils.SelectEndpoint
                            ("opc.tcp://" + host + ":" + port, false);
47             var serverConfiguration = EndpointConfiguration.Create(opcClientConfig);
48             var server = new ConfiguredEndpoint(serverEndpoint.Server, serverConfiguration);
49             server.Update(serverEndpoint);
50             m_session = Session.Create
                            (opcClientConfig, server, true, opcClientConfig.ApplicationName,
                            3600, new UserIdentity(new AnonymousIdentityToken()), null).Result;
51         }
52         public void Subscribe(string channel, ReceivedHandler handler)
53         {
54             var item = new MonitoredItem()
55             {
56                 DisplayName = channel,
57                 StartNodeId = channel
58             };
59             item.Notification += (itm, args) =>
60             {
61                 if (itm.DisplayName == channel)
62                     foreach (var val in itm.DequeueValues())
63                         handler.Invoke(val?.Value?.ToString());
64             };
65             var subscription = new Subscription(m_session.DefaultSubscription);
66             subscription.AddItem(item);
67             m_session.AddSubscription(subscription);
68             subscription.Create();
69         }
70     }
71 }
```

Appendix D. Expert Questionnaire and Results

The two pages of the expert questionnaire and detailed results for Section 7.6 can be found below.



Im nachfolgenden Fragebogen bitten wir um Ihre Mithilfe zum Vergleich klassischer Ansätze zur Datensammlung in der Automatisierungstechnik und eines neuen, modellgetriebenen Ansatzes.

Teil A: Vergleich zwischen Stand der Technik und neuem Ansatz

Im nachfolgenden Abschnitt sollen Sie bewerten, in wie weit Sie bestimmten Aussagen zustimmen. Je Aussage findet jeweils ein Vergleich zwischen dem aktuell etablierten Vorgehen (Stand der Technik) und dem präsentierten, modellgetriebenen Ansatz statt.

A1. Die Zugänglichkeit zu Daten aus verschiedenen Ebenen der Automatisierungspyramide ist gegeben.

	Stimme nicht zu	Stimme eher nicht zu	Teils/teils	Stimme eher zu	Stimme vollkommen zu
Klassischer Ansatz	☐	☐	☐	☐	☐
Modellgetriebener Ansatz	☐	☐	☐	☐	☐

A2. Eine großflächige Anbindung von Datenquellen ist durch den Ansatz realistisch.

	Stimme nicht zu	Stimme eher nicht zu	Teils/teils	Stimme eher zu	Stimme vollkommen zu
Klassischer Ansatz	☐	☐	☐	☐	☐
Modellgetriebener Ansatz	☐	☐	☐	☐	☐

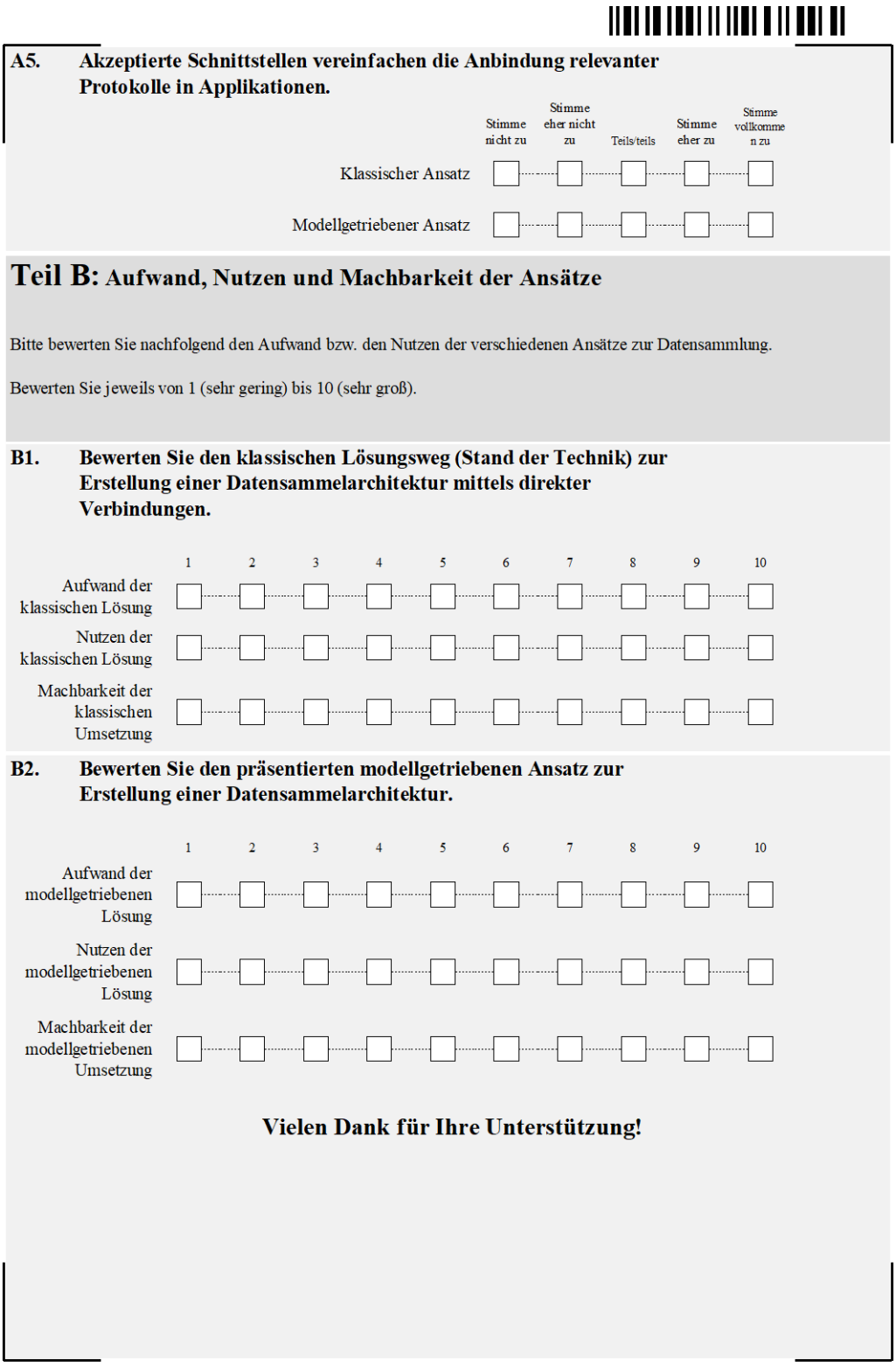
A3. Eine Migration von einem Kommunikationsprotokoll hin zu einem anderen ist bei Bedarf realistisch.

	Stimme nicht zu	Stimme eher nicht zu	Teils/teils	Stimme eher zu	Stimme vollkommen zu
Klassischer Ansatz	☐	☐	☐	☐	☐
Modellgetriebener Ansatz	☐	☐	☐	☐	☐

A4. Anwendungssoftware kann unabhängig vom zugrundeliegenden Kommunikationsprotokoll realisiert werden.

	Stimme nicht zu	Stimme eher nicht zu	Teils/teils	Stimme eher zu	Stimme vollkommen zu
Klassischer Ansatz	☐	☐	☐	☐	☐
Modellgetriebener Ansatz	☐	☐	☐	☐	☐

Figure 68: First page of the expert questionnaire in German.



A5. Akzeptierte Schnittstellen vereinfachen die Anbindung relevanter Protokolle in Applikationen.

	Stimme nicht zu	Stimme eher nicht zu	Teils/teils	Stimme eher zu	Stimme vollkommen zu
Klassischer Ansatz	☐	☐	☐	☐	☐
Modellgetriebener Ansatz	☐	☐	☐	☐	☐

Teil B: Aufwand, Nutzen und Machbarkeit der Ansätze

Bitte bewerten Sie nachfolgend den Aufwand bzw. den Nutzen der verschiedenen Ansätze zur Datensammlung.

Bewerten Sie jeweils von 1 (sehr gering) bis 10 (sehr groß).

B1. Bewerten Sie den klassischen Lösungsweg (Stand der Technik) zur Erstellung einer Datensammelarchitektur mittels direkter Verbindungen.

	1	2	3	4	5	6	7	8	9	10
Aufwand der klassischen Lösung	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Nutzen der klassischen Lösung	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Machbarkeit der klassischen Umsetzung	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

B2. Bewerten Sie den präsentierten modellgetriebenen Ansatz zur Erstellung einer Datensammelarchitektur.

	1	2	3	4	5	6	7	8	9	10
Aufwand der modellgetriebenen Lösung	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Nutzen der modellgetriebenen Lösung	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Machbarkeit der modellgetriebenen Umsetzung	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

Vielen Dank für Ihre Unterstützung!

Figure 69: Second page of the expert questionnaire in German.

Table 23: Detailed results of the expert assessment of the dimensions feasibility, total effort, and benefit for classical, manually implemented P2P network and model-driven, middleware-based approach. Scale from 1 (very low) to 10 (very high).

Aspect	Classical Approach		Proposed Model-driven Approach		Number of answers n
	Mean $\bar{X}$	Standard deviation σ_X	Mean $\bar{X}$	Standard deviation σ_X	
Feasibility	5.1	2.4	7.0	1.3	14
Effort	8.0	1.8	5.1	1.7	14
Benefit	7.5	1.8	8.6	0.9	14

Table 24: Detailed, normalized results of the expert evaluation per question (-1 = Disagreeing, 1 = Agreeing). Question texts in Figure 54.

Question	Classical Approach		Proposed Model-driven Approach		Number of answers n
	Mean $\bar{X}$	Standard deviation σ_X	Mean $\bar{X}$	Standard deviation σ_X	
Q1	-0.18	0.45	0.14	0.55	14
Q2	-0.32	0.45	0.50	0.19	14
Q3	-0.36	0.35	0.21	0.45	14
Q4	0.14	0.52	0.54	0.23	14
Q5	0.54	0.54	0.67	0.24	12