A Methodology for the Development of Cyber-physical Systems of Systems in Aviation

Vom Promotionsausschuss der
Technischen Universität Hamburg

zur Erlangung des akademischen Grades

Doktor-Ingenieur (Dr.-Ing.)

genehmigte Dissertation (Monografie)

von
Oliver Constantin Eichmann

aus
Pinneberg

2024

1. Gutachter: Prof. Dr. Ralf God
2. Gutachter: Prof. Dr.-Ing. Dieter Krause
Tag der mündlichen Prüfung: 17. Juli 2024
https://doi.org/10.15480/882.13617
https://orcid.org/0000-0003-2064-3436

Contents

Abstract **3**

Kurzfassung **5**

1 Introduction **7**
1.1 Motivation . . . 7
1.2 Solution Approach . . . 9
1.3 Thesis Structure . . . 12

2 The Phenomenon of Cyber-physical Systems of Systems **15**
2.1 Evolution of Systems and System Characteristics . . . 15
2.2 History of Process Models in System Development . . . 27
2.3 The Rise and Spread of Cyber-Physical Systems of Systems in Aviation . . . 34

3 Development Life Cycle of Cyber-physical Systems of Systems **39**
3.1 Concept Phase . . . 40
3.1.1 Approaches for the Concept Phase . . . 41
3.1.2 Evaluation of Approaches for the Concept Phase . . . 55
3.2 Function and Architecture Phases . . . 56
3.2.1 Approaches for the Function and Architecture Phases . . . 57
3.2.2 Evaluation of Approaches for the Function and Architecture Phases . . . 68
3.3 Design Phase . . . 70
3.3.1 Approaches for the Design Phase . . . 70
3.3.2 Evaluation of Approaches for the Design Phase . . . 76
3.4 Implementation Phase . . . 78
3.5 Further Phases . . . 82

4 Methods, Processes, and Tools for the Design of Cyber-physical Systems of Systems **85**
4.1 CPSoS Mission and Operational Scenarios Definition . . . 86
4.1.1 Mission and Business Requirements Definition . . . 86
4.1.2 Status Quo Documentation . . . 88
4.1.3 Behavior Description . . . 92
4.1.4 Non-Functional Requirements and Key Performance Indicators Definition 95
4.2 CPSoS Function Identification and Architecture Development . . . 97
4.2.1 Use Case Refinement . . . 97
4.2.2 Functional Grouping and Architecture Development . . . 99
4.2.3 Logical Architecture Development . . . 101

4.3 CPSoS Architecture Evaluation . . . 103
4.3.1 Evaluation Rules Definition . . . 104
4.3.2 Architecture Evaluation . . . 106
4.3.3 Evaluation Result Calculation and Architecture Draft Selection . . . 107
4.4 CPS Function Identification and Functional Architecture Development . . . 108
4.4.1 CPS Use Case Refinement . . . 109
4.4.2 Functional Grouping and CPS Architecture Development . . . 111
4.5 CPSoS Communication Architecture Implementation . . . 115
4.5.1 Modeling of Components for Contract-based Communication . . . 116
4.5.2 Communication Architecture Design and Implementation . . . 118
4.5.3 Modeling of CPSs and Simulation of Interaction with Communication Architecture . . . 121
4.6 Methodology for the Design of Cyber-physical Systems of Systems . . . 124

5 Application and Validation 127
5.1 Validating CPSoS Mission and Operational Scenarios Definition . . . 131
5.2 Validating CPSoS Function Identification and Architecture Development . . . 135
5.3 Validating CPSoS Architecture Evaluation . . . 141
5.4 Validating CPS Function Identification and Functional Architecture Development . . . 146
5.5 Validating CPSoS Communication Architecture Implementation . . . 149
5.6 Extensive Validation Results of the Developed Methodology . . . 155

6 Summary 163

Bibliography 167

List of Abbreviations 175

List of Figures 177

List of Tables 183

A Appendix 185
A.1 Communication Architecture for Message Brokers in the Galley and the Cabin Server . . . 185
A.2 Chronology of SoS Definitions, Taxonomy, Characteristics, and CPSoS Definition in Literature . . . 186
A.3 Further Disciplines for the CPSoS Design . . . 188

Acknowledgments

This thesis was written during five very interesting and exciting years at the Institute of Aircraft Cabin Systems at the TU Hamburg. During this time I was able to expand my knowledge and gain a lot of valuable experience. I would like to take this opportunity to thank my colleagues, project partners, students, family, and friends for their support.

My special thanks go to my supervisor Prof. Dr. Ralf God for the elaboration of the topic, the support during the work on the thesis and at the institute, and for the effort and time invested in the supervision. I would also like to thank Prof. Dr.-Ing. Dieter Krause for his support as second examiner and Prof. Dr.-Ing. Hermann Lödding for taking over the chairmanship of the examination.

Many thanks to my colleagues at the institute for the great team spirit, fruitful discussions, and support. I would particularly like to emphasize the collaboration with Hartmut Hintze, Fabian Maximilian Giertzsch, Marvin Blecken, and Steffen Rüsch. Thanks should also go to my student assistants for the discussions and their work - especially Caspar van der Hoek and Abdallah Altalmas.

Next, I would like to express my gratitude to the German Academic Scholarship Foundation for granting the doctoral scholarship and providing a network of PhD students for interdisciplinary exchange.

Finally, I am deeply grateful to my family for their support and encouragement over the years.

Oliver C. Eichmann

Abstract

Informatization and connectivity drive the transformation of the aviation ecosystem into a Cyber-physical System of Systems (CPSoS). A CPSoS consists of multiple independent Cyber-physical Systems (CPSs). The cooperation of these CPSs enables new business models, improved passenger experience, and efficiency improvements. One example of efficiency improvements is more accurate predictions of in-flight catering demand to reduce food waste and unnecessary weight. These predictions are based on inventory data from aircraft galleys and the catering production sites. To take advantage of this data, aircraft systems, as well as systems of catering providers and airlines, must cooperate. An example for better passenger experience is improved passenger flow management by recording passenger movements at the gate and in the aircraft as well as on the way to the airport through the cooperation of systems at the airport, in the aircraft, and other modes of transport. For reasons of safety, established development approaches in aviation are based on the segregation of systems and tend to be no longer sufficient due to the informatization, connectivity, and associated complexity of the development task.

Hence, the objective of this thesis is a methodology for the specification and the design of CPSoS in aviation. For this purpose, the characteristic properties of CPSoS as well as existing process models and approaches for their development are collected from the literature. These approaches are evaluated with regard to their suitability for the development of CPSoS on the basis of the collected characteristic properties of CPSoS. Suitable components of the existing approaches for CPSoS development are then combined and extended with newly developed elements, resulting in a methodology consisting of five methods. All methods use the Systems Modeling Language (SysML) to address the complexity of the development task.

The methods begin with the definition of the CPSoS mission and associated operational scenarios in order to capture the expectations of all stakeholders. Based on the operational scenarios, the required CPSoS functions are derived and a functional CPSoS architecture is designed. These CPSoS functions are then assigned to the CPSs contained in the CPSoS in so-called logical architecture drafts. Since the functions can be assigned to the CPSs in different ways, alternative logical architectural drafts are created that can be evaluated within the SysML model so that the most suitable logical CPSoS architecture draft can be selected. After the selection, the implementation of a communication architecture of the CPSoS is supported. The design of the CPSoS affects the architecture of the included CPSs. Therefore, in order to consider the CPSoS functions also during the design of the CPSs, the methodology supports the design of functional CPS architectures that meet both the original use cases of the CPS and the use cases that are added through integration into the CPSoS.

For validation, the methods are applied to projects for the development of an electronic passenger flow control CPSoS, an aviation industry supply chain CPSoS, and a data-driven catering CPSoS. Based on the characteristic features of CPSoS elaborated in the introductory research, validation criteria for the methods are derived and applied. Finally, the work ends with an outlook on aspects for further research.

Kurzfassung

Informatisierung und Konnektivität fördern die Transformation des Lufttransportsystems in ein cyber-physisches System von Systemen (engl.: Cyber-physical System of Systems, CPSoS). Ein CPSoS setzt sich aus voneinander unabhängigen cyber-physischen Systemen (engl.: Cyber-physical Systems, CPSs) zusammen. Die Kooperation dieser CPSs ermöglicht neue Geschäftsmodelle, ein verbessertes Passagiererlebnis und Effizienzsteigerungen. Ein Beispiel sind verbesserte Cateringbedarfsprognosen zur Vermeidung von Lebensmittelverschwendung und unnötigem Gewicht an Bord von Flugzeugen durch die Nutzung von automatisiert erhobenen Inventurdaten aus Flugzeugküchen und der Produktion von Catering sowie deren Nutzung für die Vorhersage des tatsächlichen Cateringbedarfs. Zu diesem Zweck müssen Systeme in Flugzeugküchen, im Flugzeug und am Boden bei Cateringanbietern und Fluggesellschaften miteinander kooperieren. Ein weiteres Beispiel ist eine verbesserte Passagierflusssteuerung durch das Erfassen von Passagierbewegungen am Gate und im Flugzeug sowie auf dem Weg zum Flughafen durch die Kooperation von Systemen am Flughafen, im Flugzeug und bei den Betreibern anderer Verkehrsträger. In der Luftfahrt etablierte Entwicklungsansätze basieren aus Gründen der Betriebssicherheit (engl.: Safety) auf der Segregation von Systemen und stoßen durch die zuvor beschriebene Informatisierung, Konnektivität und damit einhergehende Komplexität der Entwicklungsaufgabe an ihre Grenzen.

Das Ziel dieser Arbeit ist daher die Entwicklung einer Methodik für die Spezifikation und den Entwurf von CPSoS in der Luftfahrt. Zu diesem Zweck werden zunächst die charakteristischen Eigenschaften von CPSoS sowie bestehende Vorgehensmodelle und Ansätze für ihre Entwicklung recherchiert. Diese Ansätze werden anhand der charakteristischen Eigenschaften von CPSoS im Hinblick auf ihre Eignung für deren Entwicklung bewertet. Anschließend werden für die CPSoS-Entwicklung geeignete Bestandteile der bestehenden Ansätze kombiniert und um neu entwickelte Bausteine erweitert, sodass eine aus fünf Methoden bestehende Methodik entsteht. Alle Methoden nutzen die Modellierungssprache SysML, um der Komplexität der Entwicklungsaufgabe zu begegnen.

Die Anwendung der Methoden beginnt mit der Definition der Mission des CPSoS und zugehöriger operationeller Szenarien, um die Erwartungen sämtlicher Interessenseigner zu erfassen. Im Anschluss daran werden erforderliche Funktionen des CPSoS abgeleitet und eine funktionale Architektur entworfen. Die Funktionen werden dann den im CPSoS enthaltenen CPSs in so genannten logischen Architekturentwürfen zugewiesen. Da die Funktionen unterschiedlichen CPSs zugewiesen werden können, entstehen alternative logische Architekturentwürfe, die innerhalb des SysML-Modells bewertet werden können, sodass der am besten geeigneten Entwurf ausgewählt werden kann. Um die Funktionen auch bei der

Entwicklung und Anpassung der in dem CPSoS enthaltenen CPSs berücksichtigen zu können, unterstützt eine Methode bei dem Entwurf funktionaler CPS-Architekturen, die sowohl die ursprünglichen Anwendungsfälle des CPS berücksichtigt, als auch die durch die Integration in ein CPSoS hinzukommenden Anwendungsfälle. Nach der Auswahl des am besten geeigneten logischen Architekturentwurfs wird die Implementierung einer Kommunikationsarchitektur des CPSoS durch die fünfte Methode unterstützt.

Die enthaltenen Methoden werden zur Evaluation in einem Projekt zur verbesserten Passagierflusssteuerung und für ein verbessertes Reisererlebnis, in einem Projekt zur Produktion von Luftfahrzeugausrüstung und in einem Projekt für die Entwicklung eines datenzentrierten In-Flight Catering-CPSoS angewendet. Basierend auf den charakteristischen Eigenschaften von CPSoS, die in der einleitenden Recherche ausgearbeitet wurden, werden Bewertungskriterien für die Methoden erarbeitet und für die Evaluation herangezogen. Schließlich endet die Arbeit mit einem Ausblick auf Anknüpfungspunkte für die weitere Forschung.

1 Introduction

In recent decades there is no technical development that has influenced society more than electronics on the one hand and wireless connectivity on the other. Fourteen years after introducing the Apple iPhone in 2007, 89 % of German inhabitants aged over fourteen used a smartphone [15]. Users appreciate location-independent possibilities for communication and information availability [32]. Smartphones can serve as a typical system element in so-called *Cyber-physical Systems* (CPSs) that are characterized by computation, communication, information, and control [71, 105, 111]. In our age of digital transformation, the rise of CPSs is a direct result of global connectivity, wide-ranging digital communication, and spatiotemporally almost unlimited access to information.

1.1 Motivation

The explosive spread and use of smartphones also illustrates informatization in the field of aeronautics enabled by miniaturized electronics and digital human-machine interfaces, as shown in Figure 1.1. Meanwhile passenger smartphones have become an indispensable interface between passengers and various systems of the aviation ecosystem and enable flight booking, check-in, boarding, etc. Furthermore, in-flight connectivity allows internet access for passengers' smartphones and other portable electronic devices even in the sky.

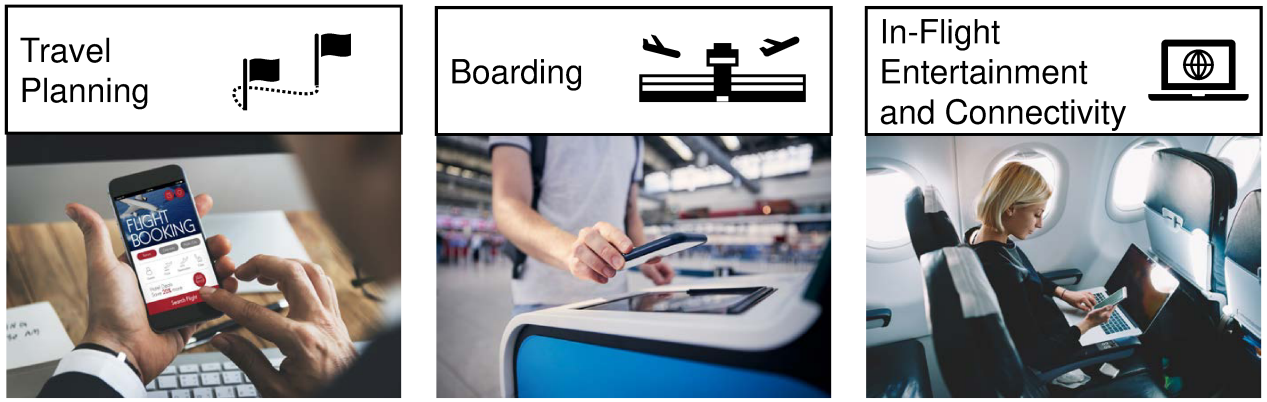


Figure 1.1: Informatization in aeronautics: Digitally supported passenger journey and passenger connectivity in the aircraft cabin

The passenger journey is only one prominent process in aeronautics that is digitally supported and enhanced by the spread of CPSs in aviation. However, this trend affects all aspects of the overall system. As an example, in the area of Cabin Operations increased connectivity between systems for passenger flow management and passenger identification

can support the development of new, more efficient boarding concepts. Furthermore, services and entertainment for passengers can benefit from connectivity, *e.g.* by internet access during flight or by digital payment processes on board.

Besides passenger-related processes, there is great potential for the digitization of further operational processes in aeronautics (*cf.* Figure 1.2). Predictive maintenance can contribute to safety and efficiency by replacing systems and components before they fail. For this purpose, data on aircraft system health status can be collected, preprocessed, transmitted to ground, and analyzed with regard to possible faults. Based on the analyses, initiation of maintenance measures to prevent faults from propagating into failures of systems is possible. Connectivity also allows the modernization of catering, baggage, and freight-handling processes. In conventional catering processes, the amount of catering consumed during the flight is unknown as the manual inventory is too time-consuming, resulting in catering overload and waste. Automated inventory solutions in aircraft galleys can transmit inventory results to ground stations in a timely manner, even during flight, to improve catering consumption estimates.

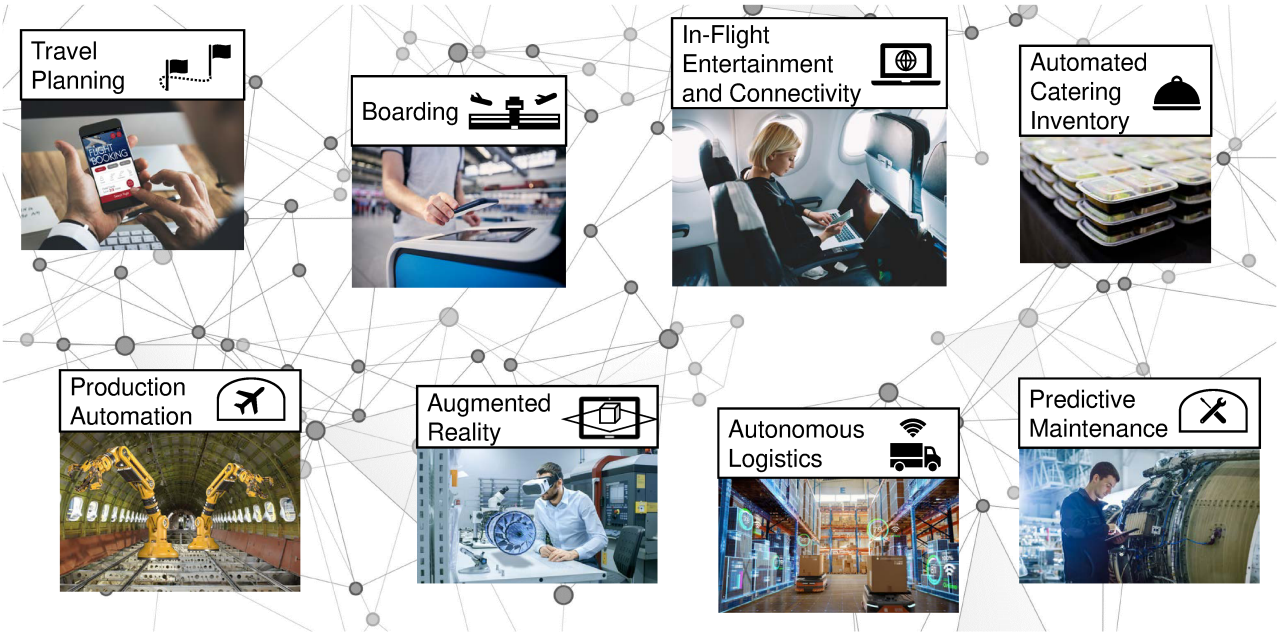


Figure 1.2: In today's aviation Cyber-physical Systems are merging into one extensive networking Cyber-Physical System of Systems (CPSoS) [51, 63, 64, 84].

Previously described benefits of increased connectivity and informatization can be considered as the result of formerly isolated systems working together in a so-called *System of Systems* (SoS). Individual systems, being part of an SoS, are denoted as *Constituent Systems* (CS) [48, 84]. If these CS have the attributes of CPSs, the overall system is called a Cyber-physical System of Systems (CPSoS) [63, 64], existing both in the physical world and in cyberspace.

Spatial distribution and diversity of functions make the development of CPSs and CPSoS a challenging task. Not only are the systems spatially distributed, but the development teams are also interdisciplinary and work at different locations. Process models should support mastering the complex specification and design challenges of CPSs and CPSoS. However, due to increasing functionality and complexity, even the sophisticated V model [16] as a prominent process model typically used to master demanding development tasks tends to be no longer sufficient for the design challenges associated with the spread of CPSoS and CPSs.

1.2 Solution Approach

With publication of the semi-formal *Systems Modeling Language* (SysML) by the Object Management Group (OMG) in 2007 for model-based support of specification, analysis, design, verification, and validation of systems [4], the so-called Model-based Systems Engineering (MBSE) is suggested and explored [67, 69, 109]. A widely accepted definition for MBSE is provided in the Systems Engineering Vision 2020 that defines MBSE to be “The formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases” [5]. Hence, the use of MBSE in combination with the V model, well known in aviation, is a promising approach to cope with the aforementioned development challenges. Thus, in this thesis, application of MBSE is extensively studied and evaluated and finally an MBSE methodology consisting of methods, processes, and tools is developed. An encountered MBSE methodology for future CPSoS development should support development teams in management of increased diversity of functions, the design of CPS cooperation along widespread spatial distribution, and implementation of the required communication between involved systems.

The SysML can represent a suitable tool for the methodology as it meets the demands for a general and interdisciplinary modeling language. Related work and an interesting starting point for more recent SoS research and development is the SysML-based *Unified Architecture Framework* (UAF) [8] for enterprise architecture modeling. This is because of large enterprises with multiple divisions and extensive product portfolios exhibit SoS characteristics. Figure 1.3 shows UAF view specifications in the form of a comprehensive matrix. Each matrix row represents a domain and matrix columns represent model kinds. Domains structure the information in the model, *e.g.* with respect to strategy, operations, personnel, resources, security considerations, and so on. Model kinds are means of representing domain-specific information. They consider, as an example, taxonomy, structure, and connectivity in the respective domains. Each intersection of a domain row and a model kind column represents a view specification for that UAF defines model elements and associated relationships. As an example, the view specification *Strategic Taxonomy* in the enlarged segment in Figure 1.3 describes taxonomy, composition, dependencies, and

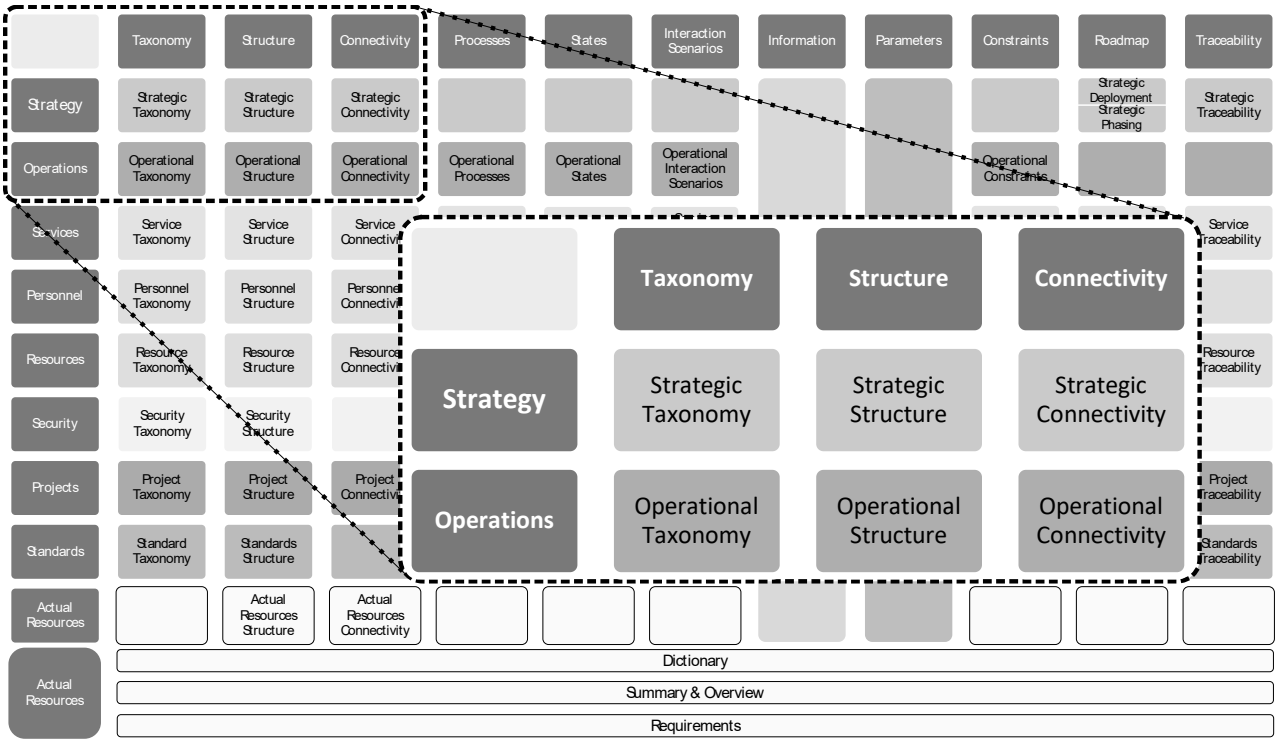


Figure 1.3: Unified Architecture Framework for Enterprise Architecture [8]

evolution of enterprise capabilities. The UAF is only one example of SoS modeling based on the SysML. However, the UAF offers potential for improvement considering CPSoS modeling in aeronautics as it focuses on enterprise architecture modeling. For this reason, the model elements and relationships proposed by UAF are enterprise architecture specific and not fundamentally suitable for modeling other types of CPSoS.

Using some of the pioneering ideas and concepts of the UAF, this thesis proposes methods and processes for CPSoS development. These adapt and enhance current approaches for Systems Engineering and System of Systems Engineering with respect to CPSoS properties in aviation. The Aerospace Recommended Practice (ARP) 4754A “Guidelines for Development of Civil Aircraft and Systems” (previous title: “Certification Considerations for Highly-Integrated Or Complex Aircraft Systems”) [6] describes the development life cycle phases of *Concept*, *Function*, *Architecture*, *Design*, *Implementation*, and *Production and Operation* of civil aircraft and systems. Existing Systems Engineering and System of Systems Engineering approaches for these life cycle phases from other domains are analyzed with regard to their suitability for CPSoS development and form the basis for the methods and processes proposed in this thesis.

The use of MBSE is often combined with the use of a process model [69, 109]. A process model provides structured phases for the system development process and corresponding methods and techniques [102]. The V model [3, 42, 68] is an established process model for the development of self-contained systems with few interfaces to other systems and covers

all life cycle phases described in the ARP 4754A. As shown in Figure 1.4, the V model is structured into a design (left side of the V), an implementation (bottom of the V), and an integration part (right side of the V). The main physical and logical operating characteristics of the system are established in system design, in that the functions are identified and divided into subfunctions. Operating principles and solution elements are selected for these functions, resulting in a general solution concept for the system. This general solution concept is further refined in the involved development domains during the implementation phase. Results of the implementation are gradually integrated in the subsequent system integration phase. During this integration phase, system properties defined in the system design serve as the basis for verification of integration results, as indicated by the horizontal arrows (assurance of properties) in Figure 1.4.

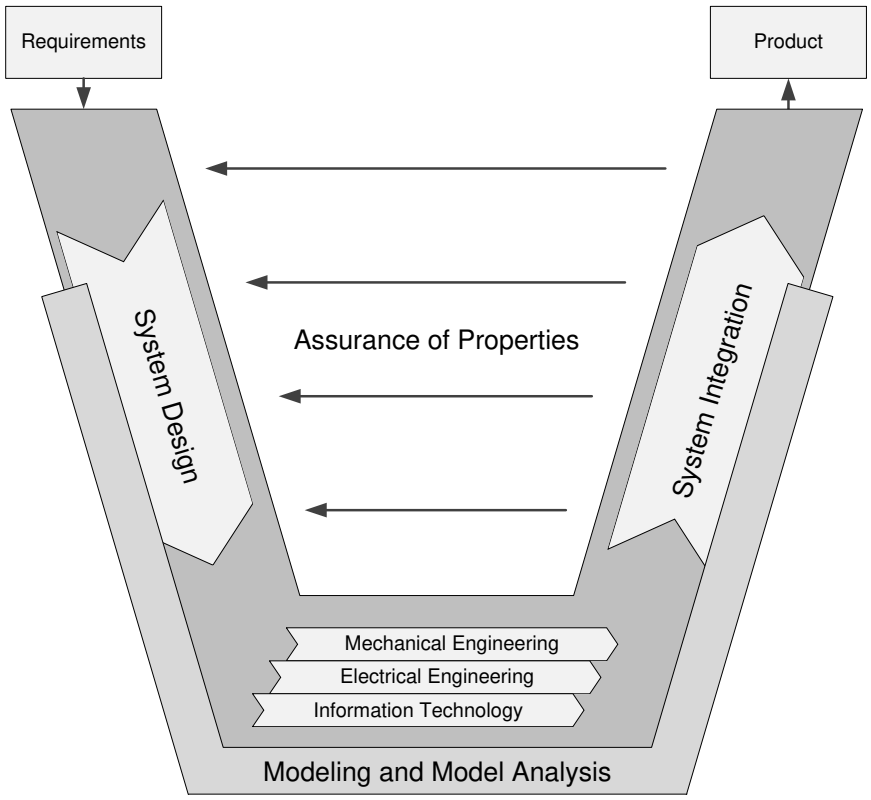


Figure 1.4: The V model for the development of mechatronic systems [3]

The well-established V model in Figure 1.4 from the Association of German Engineers (Verein Deutscher Ingenieure, VDI) [3] is a process model for the specification and design of mechatronic systems. Hence, few interfaces to external systems are considered but a comprehensive view of a CPSoS with associated connectivity and communication between CPSs is not part of the V model. To enable the V model for the development of CPSs and CPSoS, the methodology proposed in this thesis uses the SysML and concepts from the UAF as a starting point for CPSoS development. The V model is extended by a CPSoS development level for that methods and processes are elaborated.

1.3 Thesis Structure

After this introduction in **Chapter 1**, **Chapter 2** describes the phenomenon of CPSoS and its impact on the need for developing new process models. First, the evolution of system types ranging from mechanical over mechatronic to CPSs, SoS, and CPSoS is presented to provide an overview of the system characteristics. Established process models for the development of the aforementioned system types are provided subsequently. The chapter concludes with a depiction of CPSoS in aviation and aviation-specific process models.

Consistent with the development life cycle phases of concept, function, architecture, design, implementation, and further phases, **Chapter 3** describes results of a literature review about existing development approaches according to the respective phases. As the focus of this thesis is the specification and the design of CPSs and CPSoS, the concept, function, and architecture life cycle phases are mainly considered. For these phases, CPS- and CPSoS-specific evaluation criteria are identified and used for the assessment of existing approaches.

Estefan [65] defines a methodology as a combination of methods, processes, and tools, as shown in Figure 1.5. Following his definition, methods, processes, and tools are elaborated in this thesis and integrated into a methodology for the design of CPSoS and CPSs, presented in **Chapter 4**. The methods adapt and extend suitable aspects of previously introduced development approaches and cover CPSoS mission and operational scenarios definition, the identification of functions, and the design of functional, logical, and communication architectures. The transition from CPSoS development level to CPS development level is supported by a method for CPS function identification and functional architecture design.

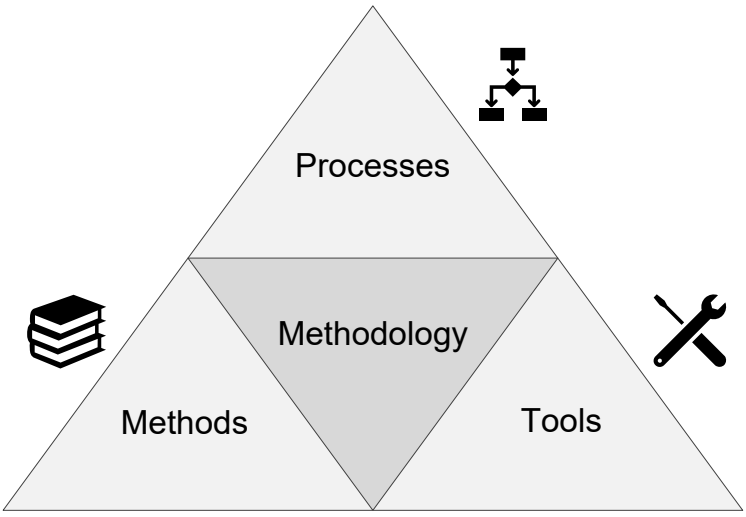


Figure 1.5: A methodology is the combination of processes, methods, and tools [3]

In **Chapter 5** the results from application and validation of the methods are given. The proposed methods are used for mission and operational scenarios definition as well as function identification and architecture design in a project for the improvement of passenger flow control and seamless travel experience. For this purpose, systems at departure and destination airports, airline systems, and aircraft systems must cooperate and form a comprehensive CPSoS. Another large CPSoS includes production systems in the aviation industry supply chain. The method for CPSoS communication architecture implementation is applied to a project for improving design, production, and information management processes in the aviation industry supply chain. The third and last project for application and validation of all methods of the proposed methodology is the design of an extensive data-driven catering CPSoS for more precise catering demand predictions. The results from application to the projects is used for an evaluation of the elaborated methodology.

Finally, **Chapter 6** provides a conclusion of the thesis as well as an outlook and this chapter is followed by the bibliography, lists of abbreviations, figures, tables, and the appendix.

2 The Phenomenon of Cyber-physical Systems of Systems

To better understand the phenomenon of CPSoS and new services that emerge with CPSoS, two main aspects should be taken into account, *i.e.* system types and their characteristics as well as related process models for their specification and design. The following Section 2.1 presents the evolution of system characteristics based on the system types mechanical systems, mechatronic systems, CPSs, SoS, and CPSoS. Subsequently, process models for the specification and the design of these system types are provided in Section 2.2 on page 27. As Systems Engineering in aviation is particularly characterized by many non-functional safety and security requirements to achieve a highly deterministic behavior of the systems, the rise and spread of CPSoS challenges systems engineers working with existing process models. Therefore, the final Section 2.3 of this chapter on page 34 is dealing with the phenomenon of CPSoS in aviation, aviation-specific process models, and the resulting research needs.

2.1 Evolution of Systems and System Characteristics

Before the advent of CPSoS there were several stages in the evolution of system types and their characteristics. This evolution can be observed in the major changes in production systems known as industrial revolutions, as illustrated in Figure 2.1.

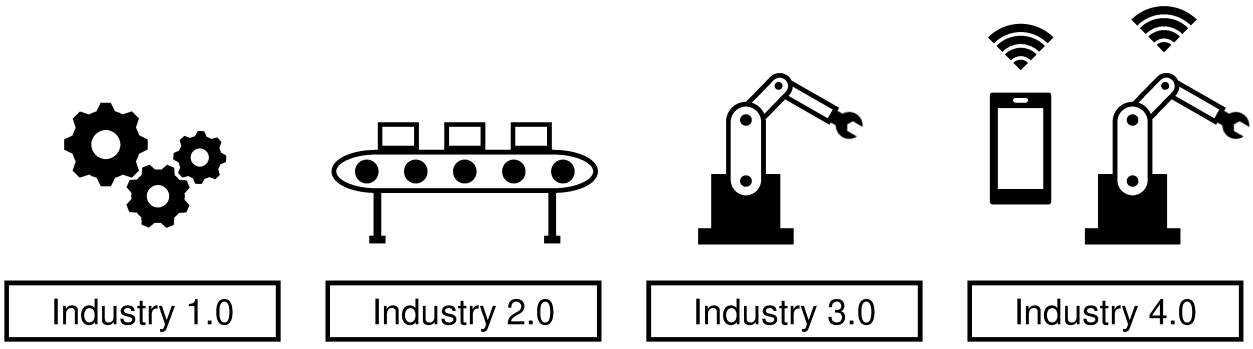


Figure 2.1: Industrial revolutions, illustration by [13]

The first industrial revolution began in the second half of the 18th century and was triggered by the mechanization of manual work by machines, mechanical production, conversion of energy, and the massive use of coal and iron as raw materials [112]. These production systems correspond to mechanical systems and were further developed in the second industrial revolution. Georges Friedmann dates the second industrial revolution to the

decades around 1900 with intensified mechanization, electrification, and mass production of goods [70]. While the first and second industrial revolutions are enabled by mechanical systems, the third industrial revolution is driven by mechatronic systems and the wide distribution of microelectronics and embedded systems in the mid-1970s [36]. The indicator for the fourth industrial revolution is the intensive digital communication and networking of these mechatronic systems in cyberspace, which leads to their classification as CPSs [47].

However, evolution of system properties and capabilities did not only affect production systems, but all types of systems. Tools and methods for the development of CPSoS must take into account CPSoS-specific properties. Hence, evolution of system types and their attributes are presented in the following. The evolution includes mechanical systems, mechatronic systems, CPSs, SoS, and CPSoS.

The first technical systems were mechanical systems that are still the basis for the physical part of CPSs. According to Feldhusen and Grote [66] in mechanical engineering, a system is defined as a technical entity that is connected to its environment through inputs and outputs that cross the system boundary. Aspects for system boundary definition are, *e.g.* system functions and considerations for production and installation. A system can be divided into subsystems and be part of a higher-level system [66].

The addition of electronic components and software to mechanical systems and the close interaction between mechanics and electronics leads to the characterization “mechatronics.” The term mechatronics is a combination of mechanisms, mechanics, or mechanical engineering with electronics and electrical engineering [75]. With the proliferation of microelectronics and microprocessors, information technology is having a significant impact on mechatronic systems. Mechatronic systems are characterized by the functional and physical integration of sensors, actuators, information processing, and a basic system. The basic system usually consists of mechanical structures but can also contain fluid-technical, chemical, or biological structures. Mechatronic systems have in common that sensor data is processed and provides information about the environment and the system status. Based on this information, system actuators are controlled so that a mechatronic system can respond to environmental influences and its current system state. This ability is denoted as adaptability and enables responses to changes in the environment and critical operating states as well as process optimization [3].

Ubiquitous connectivity and increasing available bandwidths enable the interconnection of previously isolated mechatronic systems so that distributed networks with various interacting systems emerge. If independent systems are connected to each other, *e.g.* through the internet, so-called Systems of Systems (SoS) emerge. Jaradat, Keating, and Bradley evaluated 500 publications about SoS from the period 1950 to 2011. Contributions to definitions for SoS and System of Systems Engineering (SoSE), about SoS characteristics, SoS methodologies as well as principles and axioms for SoS were considered. The evaluation revealed three

time periods in SoS research, *i.e. Recognition of Complex Systems*, *Exploration of SoS*, and *Revolution of SoS*. These periods are part of the overview about periods in SoS research in Figure 2.2. Subsequent periods are identified in a supplementary literature review about SoS definitions, taxonomy, characteristics, and the evolution of CPSoS in the course of this thesis. A more detailed presentation about identified literature and references is provided in the appendix A.2 on page 186.

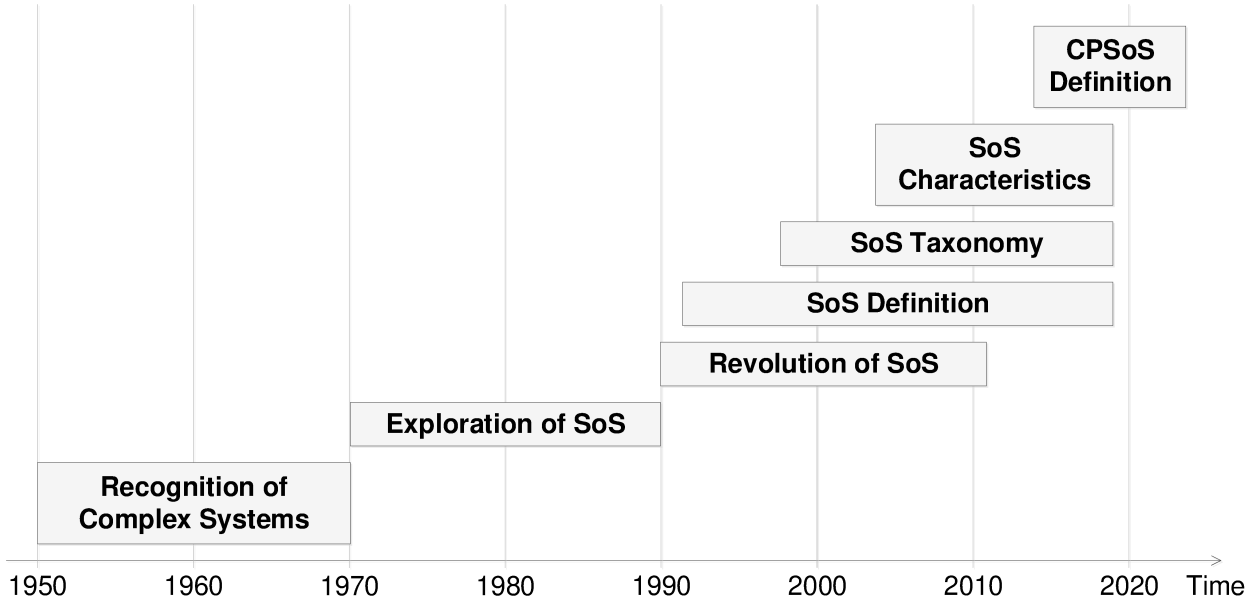


Figure 2.2: Time periods in SoS research

The first period between 1950 and 1969 is denoted as *Recognition of Complex Systems* and characterized by consideration of holistic approaches in order to deal with more complex systems. It was recognized that traditional reductionist approaches for the consideration of self-contained systems are not suitable for more complex and evolving systems. Evolution of complex systems was mentioned for the first time. Today, evolution is an important and accepted SoS property [23,41,103]. It has to be noted that the term SoS was not established during the first period. Instead, there were different terms for the description of SoS. This also becomes apparent with the introduction of the term “holon” for the description of the whole and parts of a system. Berry introduced the term “system of cities” that are “systems within systems” [39] in 1964. Berry analyzes cities from a socio-economic viewpoint and considers cities as a composition of systems that have dependencies on other cities, thus forming a system of cities [39].

The frequency of SoS-relevant publications increased in the second period called *Exploration of SoS* between 1970 and 1989. The idea of analyzing a complete system rather than breaking it down into parts also matured in the second period. More definitions appeared that describe SoS indirectly, *e.g.* Ackoff describes “a set of interrelated (or integrated) elements” as a so-called “integrated set.” [27] In addition, first system-based methodologies addressing

the SoS problem domain were developed. Beer introduced the term “metasystem” [37] for the integration of systems and developed the “viable system model” (VSM) comprising five main functions of a complex system. These main functions are the *productive* function for the produced output of the system, the *coordination* function for coordination of subsystems and oscillation avoidance, the *operation* function for decisions in daily operations, the *development* function for surrounding environment exploration and system evolution to ensure the existence of the system in the future, and the *identity* function that links the previously described functions [37].

Revolution of SoS occurred in the third period between 1990 and 2011. The publication frequency and the number of definitions, taxonomies, foundational principles, and proposed methodologies increased significantly. Accompanying the increased attention given to SoS, symposiums, journals, and conferences concerning SoS were established and some consensus regarding SoS characteristics was achieved [85]. Parallel to the *Revolution of SoS*, the first SoS definitions were published and discussed, beginning the period of *SoS Definitions*. Eisner is one of the first authors who mentions the SoS term in a publication [62] in 1991. In his publication he does not provide a reference for SoS definition, which suggests that he is one of the first authors to use the SoS term. Instead, Eisner deduces the need for an SoS engineering discipline because of growing size and complexity of systems being developed by different organizations with individual SE processes. One example is the aviation system including airports, airways, aircraft, and air traffic control. Another is the Tactical Warning and Attack Assessment Systems by the Department of Defense (DoD) in that ballistic missiles, atmospheric and space sensors, command and control centers as well as communication systems cooperate to warn of possible attacks by missiles and enemy aircraft [62].

Maier emphasizes geographic distribution and complexity as a taxonomy for SoS in 1998 [89] based on Shenhar’s and Eisner’s contribution from 1993. However, these characteristics are not exhaustive according to Maier. Instead, Maier suggests using the degree of centralization of control in the design and operation of CS as a taxonomy for SoS so that the period of *SoS Taxonomy* begins. The telephone system and the IBM System Network Architecture are examples of centrally developed SoS. In contrast, the internet is a decentralized SoS where CS collaborate. Operational and managerial independence of CS differs in both examples. Operational independence characterizes components that, when separated from the SoS, are still able to operate independently and “fulfill customer-operator purposes on their own.” [89] In addition, components of an SoS are managerially independent and therefore “maintain a continuing operational existence independent of the system-of-systems.” [89] Different degrees of managerial and operational control in SoS enable categorization. Maier distinguishes between directed, collaborative, and virtual SoS. Directed SoS are designed and managed in order to fulfill a specific purpose. They are centrally managed during long-term operations even though their components are managerially and operationally independent. Air defense networks are an example of directed SoS with independent CS,

e.g. combat planes that are centrally managed to fulfill the purpose of defending a region from enemy systems. In collaborative SoS, the central management does not have coercive power to run the CS. The internet is an example of a collaborative SoS in that CS voluntarily collaborate. In case of virtual SoS, both a central management authority and voluntary agreement are missing. A high degree of emergent behavior may occur as a result. The World Wide Web and national economies are examples of virtual SoS with a large physical and managerial distribution [89].

Ten years later in 2008, Dahmann and Baldwin take up Maier's definition for SoS, *i.e.* directed, collaborative, and virtual, and add acknowledged as an additional SoS type. Many SoS appear to be directive because of shared objectives, funding, and management even though their CS retain their individual ownership, objectives, and resources. Acknowledged SoS combine properties of directed and collaborative SoS. This SoS type is often chosen as a classification for SoS in that independently developed CS are operated as closely as in a directed SoS but have a higher degree of independence comparable to that of a collaborative SoS [49]. A smart city can be classified as an acknowledged SoS if it has a recognized objective and a designated manager. The CS include buildings and their heating systems as well as systems for energy production and storage. A managing authority distributes power between CS depending on the power available in the power grid based on the common objective of energy efficiency. However, CS retain individual ownership and management.

Meanwhile, the taxonomy and the four SoS types, *i.e.* directed, acknowledged, collaborative, and virtual, are widely accepted and integrated into ISO/IEC/IEEE standard 21841:2019 [24]. Ncube and Lim propose a graphical representation of the SoS types and relationships between organizations and systems that is depicted in Figure 2.3. Organizations and systems are represented by circles and squares respectively. Managing organizations in directed and acknowledged SoS direct system operation and are marked yellow. In collaborative and virtual SoS, organizations operate systems that are part of the SoS. The relationships between organizations are represented by dashed lines and become less strict as the degree of freedom of the SoS types increases. The managing organization defines the design in directed SoS and has contractual relationships in acknowledged SoS. In collaborative SoS, there are peer-to-peer links between organizations, and virtual SoS do not have any connections between organizations.

Parallel to the development of an SoS taxonomy, discussions on SoS definitions continue. Most publications that offer SoS definitions are based on the SoS taxonomy by Maier, Dahmann, and Baldwin. Dahmann and Rebovich's contribution to Systems Engineering for capabilities defines an SoS as "a set or arrangement of systems that results when independent and useful systems are integrated into a larger system that delivers unique capabilities." [48] This definition is also presented in the Department of Defense (DoD) Defense Acquisition Guidebook [50]. The International Council on Systems Engineering

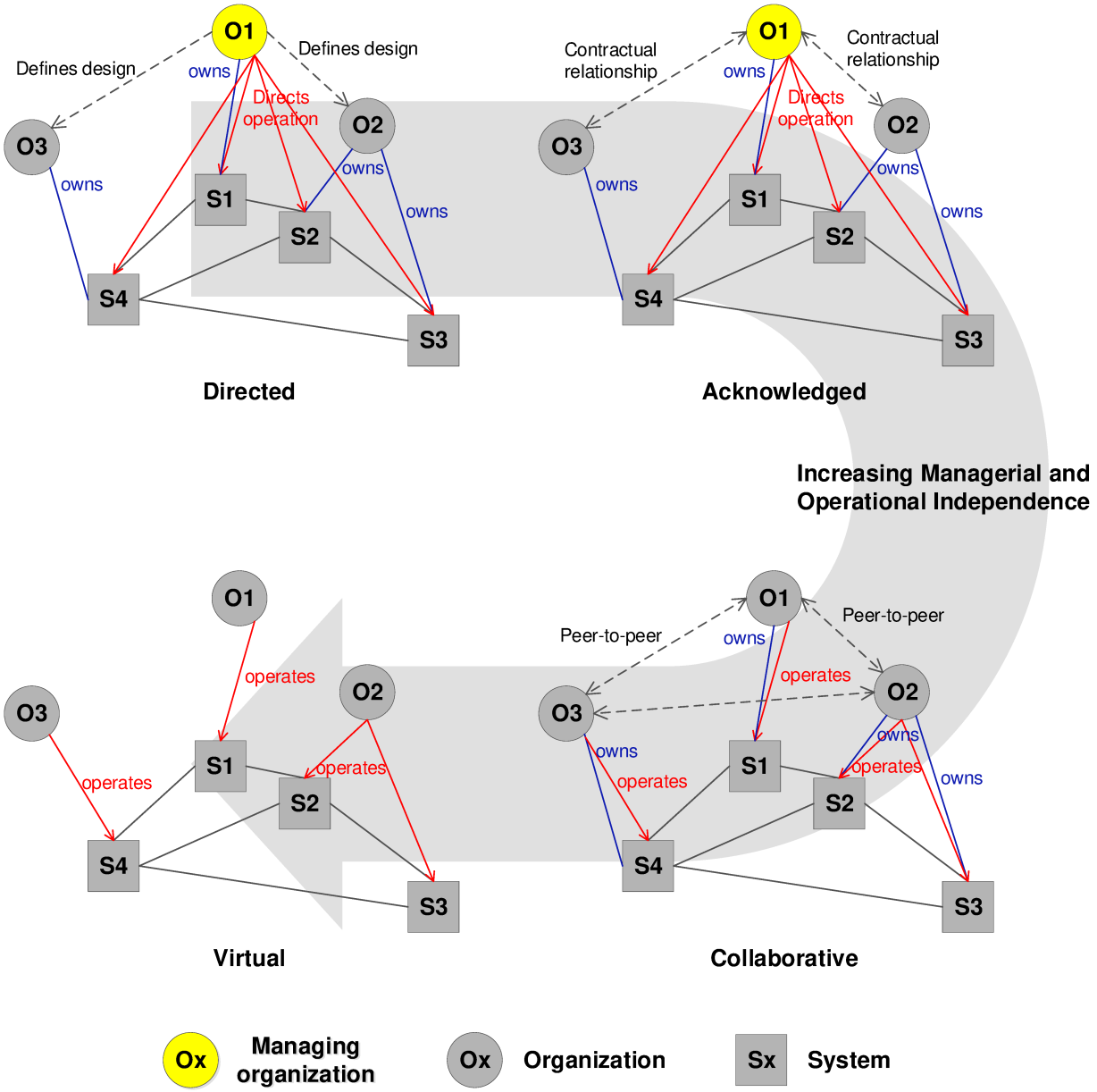


Figure 2.3: SoS types, arranged according to increasing managerial and operational independence [94]

(INCOSE) offers a similar definition in its Systems Engineering Handbook. The term SoS "applies to a system-of-interest whose system elements are themselves systems; typically these entail large scale inter-disciplinary problems with multiple, heterogeneous, distributed systems" according to INCOSE [108]. Another popular SoS definition is provided by Jamshidi in his book *System of Systems Engineering* from 2009 in that SoS are defined as "large-scale integrated systems that are heterogeneous and independently operable on their own, but are networked together for a common goal." According to Jamshidi, this definition is derived from various definitions taking into account SoS characteristics and SoS Engineering in enterprise and military contexts [84].

The definitions in the DoD Defense Acquisition Guidebook, the INCOSE Systems Engineering Handbook, and System of Systems Engineering by Jamshidi describe an SoS as a system consisting of systems. However, there are minor differences in composition and purpose of SoS. Regarding SoS composition, the DoD Defense Acquisition Guidebook and Jamshidi emphasize CS independence, and this is also an important aspect in Maier's SoS taxonomy [89]. Regarding the purpose of SoS, the DoD Defense Acquisition Guidebook emphasizes unique capabilities of SoS that come with emergent behavior. INCOSE does not take into account SoS purposes in its definition, while Jamshidi identifies a common goal of CS.

SoS definitions already consider some special SoS characteristics, *e.g.* unique capabilities [48], heterogeneity [84, 108], and distribution in [108]. Since 2004, more contributions have been published about SoS properties, marking the starting point for the *SoS Characteristics* period. Bar-Yam et al. collected SoS characteristics in biology, sociology, and military domains in 2004. Common characteristics of these domains are

- evolutionary development,
- emergent behavior,
- self-organization,
- adaptation,
- complex systems,
- individual specialization, and
- synergy [29].

A contribution by Boardman and Sauser from 2006 [41] provides a comprehensive literature review about SoS characteristics, compares SoS characteristics to properties of self-contained systems, and also cites the publication by Bar-Yam. According to Boardman and Sauser, main SoS characteristics are

- autonomy,
- belonging,
- connectivity,
- diversity, and
- emergence.

Autonomy describes freedom of choice of CS to join an SoS. The integration of CS into an SoS is expressed through belonging and differs from the traditional integration of parts into a self-contained system. While parts cannot operate independently when removed from the system, CS continue to function with respect to their individual purposes when removed from an SoS. In contrast to systems with a limited number of communication links between integrated parts, the degree of connectivity between CS is drastically increased in SoS. Diversity describes the heterogeneous composition of SoS from CS with individual contexts and pasts. Diverse CS in an SoS should be able to respond to different problems by taking advantage of diversity in the form of emergent behavior and resilient capabilities. Emergence describes behavior that only emerges through the combination of systems in

an SoS and that cannot be achieved by individual, self-contained systems. Each CS has its original purpose and is additionally integrated into an SoS, creating a new purpose [41]. Sauser, Boardman, and Verma take up the publication from 2006 [41] in 2010. Aside from source citations, there are no differences between the identified SoS characteristics, but the recent publication offers an explanation for the occurrence of SoS, *i.e.* intensely networked systems and interconnection of networks [103].

SoS characteristics were not only described in publications about the nature of SoS. An SoS definition and SoS properties are also provided in a contribution about SysML for Systems Engineering by Holt and Perry in 2014 [79]. They refer to Maier's SoS taxonomy from 1998 [89] and to Dahman and Rebovich's publication from 2008 [48]. Holt and Perry define an SoS as "a special type of system whose elements are one or more Constituent Systems and which deliver unique functionality not deliverable by any single Constituent System." This definition is similar to that of Dahman and Rebovich [48]. Differences in Holt and Perry's definition are the view that an SoS is also a system and the emphasis on unique functionality rather than unique capabilities. In addition to the SoS definition, Holt and Perry also provide SoS characteristics, *i.e.* independence, emergence, and evolution [79].

The increasing attention to SoS is also visible in three new standards that include SoS considerations and complement the ISO/IEC/IEEE standard 15288:2015 (Systems and software engineering – System life cycle processes). ISO/IEC/IEEE 15288:2015 describes system life cycle processes for self-contained systems. The new standards describe SoS considerations in life cycle stages of a system in ISO/IEC/IEEE 21839:2019 [22], guidelines for the utilization of ISO/IEC/IEEE 15288 in the context of SoS in ISO/IEC/IEEE 21840:2019 [23], and SoS taxonomy in ISO/IEC/IEEE 21841:2019 [24]. These standards are based on the previously introduced publications and are also included in the overview in the appendix A.2 on page 186. Further information about the included processes is provided in Section 3.1 about the concept life cycle phase on page 40 and Figure 3.2 on page 42 presents applicability of standards. The publication of standards suggests that there is a broad consensus regarding the definition and taxonomy of SoS. ISO/IEC/IEEE standard 21839:2019 defines SoS similarly to Dahman and Rebovich in [48] as a "set of systems or system elements that interact to provide a unique capability that none of the CS can accomplish on its own." According to the standard, CS are independent and form a part of an SoS [22]. The SoS taxonomy in ISO/IEC/IEEE 21841:2019 corresponds to Maier's SoS taxonomy in [89], *i.e.* directed, collaborative, virtual, and adds the acknowledged SoS type by Dahman and Baldwin [49].

As soon as systems are intensively connected with the cyberspace, they are also referred to as immersive. This type of system is called cyber-physical system (CPS) [71]. Many definitions emphasize the properties communication, computation, information, and control of CPSs [71, 105, 111]. However, the definitions differ in the composition of CPSs. While

some definitions assume that CPSs contain systems, other definitions do not address composition. An example of the consideration of multiple systems in CPSs is provided by Zhang [111]. Zhang defines CPSs as systems "in which physical systems and cyber systems not only are converged, but also computing, communication, and control technologies are tightly integrated." [111] This definition also states that most CPSs contain several subsystems [111]. Song et al. do not address composition and state that CPSs "integrate cyber components (namely, sensing, computation, control, and networking) into physical components (namely, physical objects, infrastructure, and human users), connecting them to the internet and to each other." [105] They characterize CPSs by a high degree of capability, adaptability, scalability, resiliency, safety, security, and usability [105]. Engell does not comment on composition and defines CPSs as "large complex physical systems that are interacting with a considerable number of distributed computing elements for monitoring, control and management which can exchange information between them and with human users." [63] This definition emphasizes the interaction with other systems using information exchange in a distributed environment. All definitions consider communication, networking, or interaction. These properties enable the interaction of multiple CPSs in a larger network. As the CPSs are individual and independent systems, the system network can be characterized as a System of Systems (SoS) [89] or, more precisely, a CPSoS [46, 63]. In the following, results from a literature review about the evolution of SoS and CPSoS research are provided.

Both system types, CPSs and SoS, are often characterized as smart systems if they are able to monitor their environment and change their behavior accordingly. Definitions for smart systems differ in the consideration of self-contained systems and aggregated systems and can be divided into two groups. One group [1, 28] defines self-contained systems as smart if they are equipped with sensors, actuators, and elements for coordination, communication, and signal processing that enable the smart system to monitor its environment and adjust its behavior based on its observations. This group of definitions corresponds to the definition of the CPS type. The other group considers aggregated systems as smart systems, *e.g.* smart cities [100], smart energy grids [98], and smart logistics [99]. Systems covered by these examples are those that are managerially and operationally independent; the second group of definitions for smart systems corresponds to the definition of SoS. In summary, the term smart does not define a new system type besides CPSs and SoS. Instead, the term smart emphasizes the capability of CPSs and SoS to monitor their environment and adapt their behavior accordingly.

Before the start of the period *CPSoS Definition* in 2014 (*cf.* Figure 2.2 on page 17), Sauser, Boardman, and Verma already mentioned SoS occurrence in "cyberphysical ecosystems." [103] Engell is one of the first authors who uses the term *Cyber-physical System of Systems* in 2014 and provides a definition [63]. According to Engell, CPSoS are CPS "which exhibit the features of SoS." [63] The features mentioned by Engell are congruent with the previously described SoS characteristics, *i.e.* distributed systems with distributed control, supervision,

and management, autonomy as well as dynamic reconfiguration and evolution. In addition, Engell refers to Jamshidi's SoS definition and emphasizes Maier's definition of managerial and operational independence of CS. Engell does not refer to another source for CPS definition. According to Engell and as already described during CPS definition above, CPSs are systems "where real-time computing and physical systems interact tightly." [63] The same definition is cited by Engell, Paulen, Reniers, Sonntag, and Thompson in [64] from 2015. Additional explanation by Engell et al. in [64] states that CPSoS consist of many interacting physical elements and distributed information technology (IT) systems. These IT systems are used to monitor, control, optimize, and interact with human operators and managers. Communication between IT systems is established by means of communication networks. Examples of CPSoS are railway systems, electric grids, and production plants [64].

For the development of a methodology for the design of CPSoS, the following SoS and CP-SoS properties identified in the literature review are primarily considered. These properties are also used for the validation of the methodology in the Section 5.6 on page 155.

- **Complexity**
- **Managerial and operational independence of CPSs**
- **CPS diversity**
- **Multiple communication paths between involved CPSs**
- **Distributed control**
- **Evolution**

Bondavelli, Bouchenak, and Kopetz published a book [44] with the title *Cyber-physical Systems of Systems* independently from CPSoS considerations in Engell's publications [63, 64] in 2016. Chapters in this book are written by different authors and are about CPSoS nature and Systems Engineering in CPSoS. Bondavelli, Bouchenak, and Kopetz describe how CPS can observe their environment by means of sensors and influence their environment using actuators. New services can be offered when isolated CPSs are connected. If these services cannot be provided by a single CPS, the set of constituent CPSs is denoted as CPSoS. The emphasis of new services enabled by the cooperation of CPSs demonstrates the transition from conventional Systems Engineering to Service Engineering. Consequently, one objective of this thesis is the systematic identification of CPSoS services and resulting functions. Like the authors of many previously cited publications, Bondavalli et al. cite Maier's publication from 1998 [89] in the first chapter to introduce basic SoS concepts and to determine the differences between self-contained systems and SoS. This chapter on basic SoS concepts also refers to Jamshidi's SoS definition from [84] and modifies the definition slightly to "an integration of a finite number of constituent systems (CS) which are independent and operable, and which are networked together for a period of time to achieve a certain higher goal." [44]

The distinctive feature of CPSs is the immersion into cyberspace through the linkage to the internet [71, 105, 111] that enables the emergence of CPSoS [63, 64]. Figure 2.4 presents a concept for the relationships between SoS, CPS, and CPSoS types and Figure 2.5 describes the SysML elements for modeling this concept.

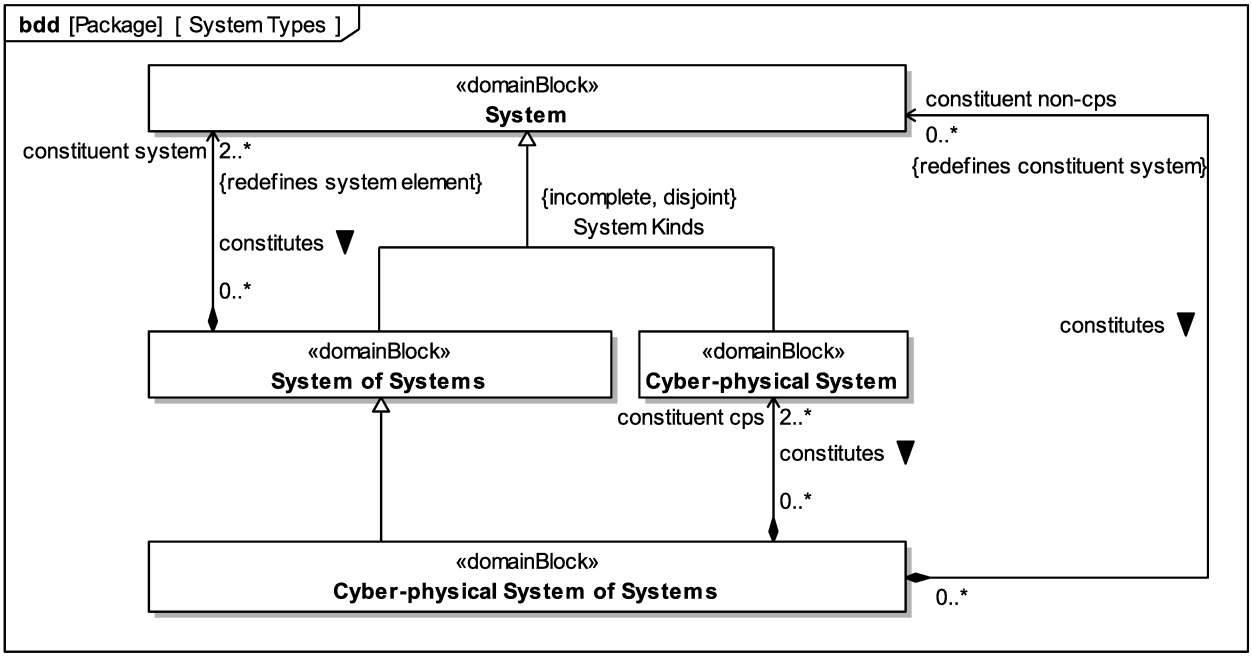


Figure 2.4: SysML block definition diagram with a concept for the relationships between SoS, CPS, and CPSoS types [53]

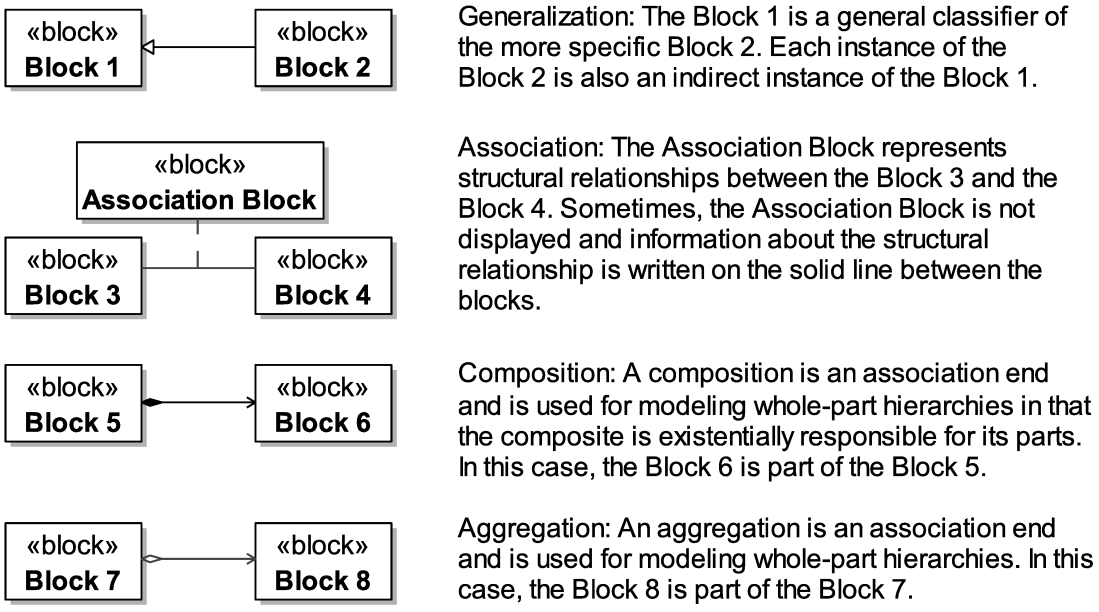


Figure 2.5: Common elements in SysML block definition diagrams [12]

All system types are specializations of the System block at the top of the diagram, as indicated by the arrow that points to the System block and is further described by a SysML

generalization set with the name *System Kinds*. The generalization set is incomplete as there are more system types, *e.g.* mechatronic systems that are not part of the diagram. SoS and CPSs are individual system types with disjoint system properties and definition so that the generalization set is disjoint. Associations are drawn with a black diamond and describe the relation between the system types. The association between the SoS and system type and related multiplicities indicate that at least two systems constitute an SoS. Systems can be part of none, one, or more SoS so that the multiplicity of SoS is zero or more. If a system is part of an SoS, it takes the role *constituent system*. The CPSoS type is a specialization of the SoS type that is constituted by CPSs and, optionally, by additional systems. For this reason, the multiplicity at the association between CPSs and CPSoS indicates that at least two CPSs are contained in zero or more CPSoS and that zero ore more systems are part of zero or more CPSoS.

Cooperation of CPSs in a CPSoS allows the provision of functions that the CPSs could not provide if they operated in isolation from each other. These new functions can be used for offering new services. If a provider of conventional tangible products connects its products with services, a *Product-Service System* (PSS) is created. Baines et al. define a PSS as a "market proposition that extends the traditional functionality of a product by incorporating additional services." [35] Based on this definition, Tukker proposes the PSS categories *product-oriented*, *use-oriented*, and *result-oriented*. The categorization depends on the ratio between the tangible product content and the intangible service content in a PSS, as shown in the center of Figure 2.6.

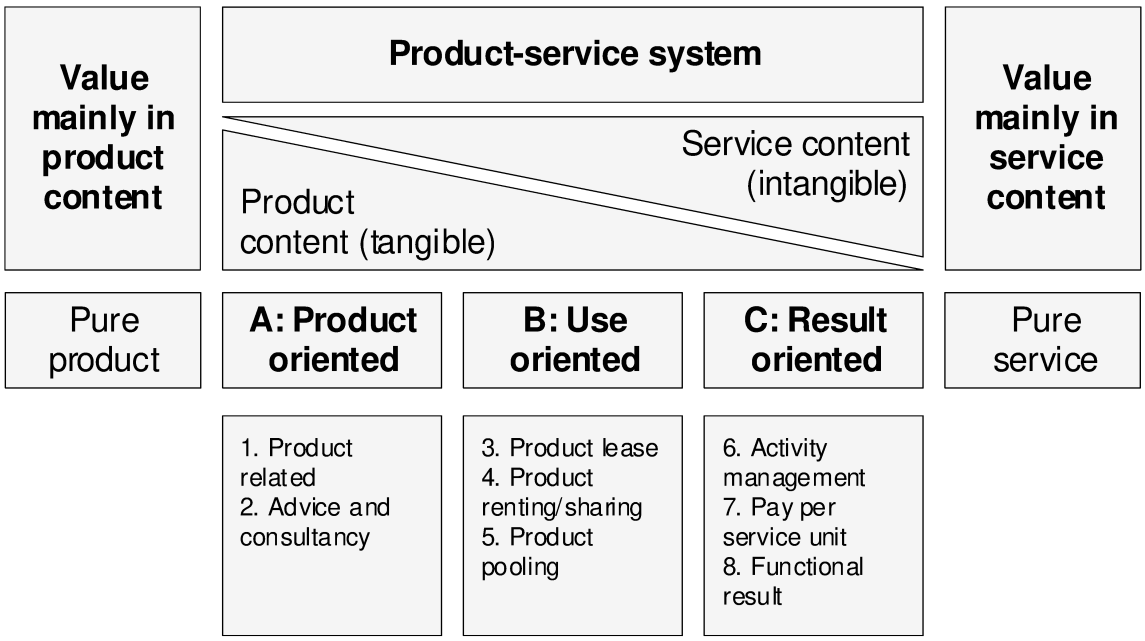


Figure 2.6: PSS categories according to Tukker [107]

In product-oriented PSS, the value of product content is greater than the value of service content. Product-oriented PSS are products that are offered in combination with a mainte-

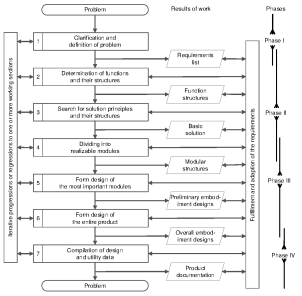
nance contract, a financing scheme, or advice and consultancy regarding the product. The product and service content are balanced in use-oriented PSS. The product is owned by the provider and users can either lease, rent, share, or use the product in product pooling concepts. In all use-oriented PSS, providers are responsible for maintenance and repair of the product. In leasing concepts, users pay for unlimited access to the product. The product is used sequentially by different users in renting and sharing concepts. Product pooling describes the simultaneous use of a product by multiple users. Result-oriented PSS are characterized by a predominant service content and include activity management, pay per service unit, and functional result concepts. A certain company activity is delegated to another company in activity management concepts, *e.g.* the provision of catering. In pay per service unit concepts, customers buy the output of a product. As an example, providers of office printers offer a copy function instead of an office printer and are responsible for paper and toner supply and maintenance. In turn, customers pay per printed page. In functional result concepts, providers and customers agree on a specific result. As an example, providers of gas or cooling equipment offer a certain climate in offices using their products [107].

2.2 History of Process Models in System Development

Organizing the development of the previously introduced system types is a challenging task that has become increasingly difficult as the system types have evolved. Multiple disciplines and teams must be coordinated for the development of extensive and complicated systems. Thus it is not surprising that the first process models emerged in the 1970s with Boehm [42] in the field of software development. This trend can also be observed in other engineering domains, *e.g.* in mechanical engineering the Pahl/Beitz [66] is the basis for design theory and methodical design and proposes methods for the design of mechanical products. Across domains, it can be stated that the increase in complexity of systems has led to the development of process models so that systems and products are designed in a structured manner. A process model organizes the system development process into structured phases, which in turn are assigned corresponding methods and techniques of the organization [102]. Different process models have been developed for each system type. This section presents the VDI standard 2221 *Systematic engineering design of technical systems and products* [10] for the development of mechanical systems, the V model [16, 42], waterfall model [101] and spiral model [43] for the development of software, mechatronic systems, and CPSs. The Unified Architecture Framework (UAF) [8] for the design and specification of SoS is already introduced in Section 1.2 on page 9. Figure 2.7 illustrates considered system types and related process models.

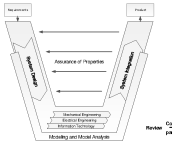
The VDI standard 2221 with the title *Systematic engineering design of technical systems and products* [2] was published in 1985 by the Association of German Engineers and provides an approach for the development of predominantly mechanical systems and products with

Mechanical Systems

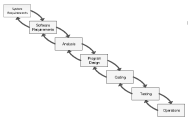


VDI 2221: Systematic engineering design of technical systems and products

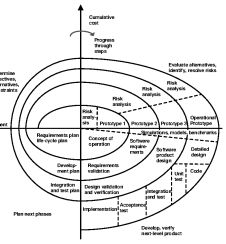
Mechatronic Systems and CPSs



V model

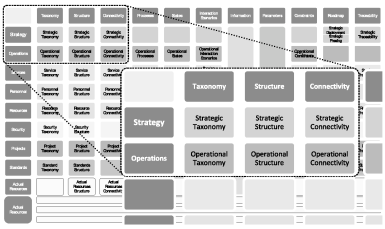


Waterfall model



Spiral model

SoS



Unified Architecture Framework

Figure 2.7: Analyzed system types and related process models that are illustrated schematically. The VDI 2221 is presented in more detail in Figure 2.8 on page 29, the V model in Figure 1.4 on page 11 and in more detail on page 31, the waterfall model in Figure 2.9 on page 30, the spiral model in Figure 2.12 on page 34, and the Unified Architecture Framework in Figure 1.3 on page 10.

a small portion of electronics and software. The standard structures the development process in the four phases, *i.e.* plan (I), concept (II), design (III), and elaboration (IV), shown on the right-hand side of Figure 2.8. In the planning (I) phase, the requirements are collected in cooperation with all relevant enterprise departments. The result of this phase is a requirements list. A function structure is developed in the following concept (II) phase based on flow and transformation of information, material, and energy. Subsequently, operating principles are selected for the defined functions and the system is structured into system elements. In the following design (III) phase, the system elements are structured into modules, designed in detail, and integrated into the product design. Afterwards, the product documentation is prepared in the elaboration (IV) phase. The product documentation includes production drawings, bills of materials, and software documentation [10]. The current version of the standard has the title *Design of technical products and systems – Model of product design* and is divided into two parts. The first part describes a general model of product design that comprises the same activities and results of work as the previous version of the standard (*cf.* Figure 2.8). The second part of the new standard version modifies the arrangement of the phases. While the first standard version proposes a sequential order of the phases with some overlap between the phases, the new version suggests a tailoring of phases to the individual project. This tailoring allows the iterative execution of phases and the switching between activities so that interim results can be created and reviewed sooner for earlier product maturity and shorter time to market [10].

The waterfall model is one of the first process models in software engineering. Its roots go back to a proposal by Benington [38], who suggested an approach similar to the waterfall model, but in a different graphical representation. The first presentation of the waterfall

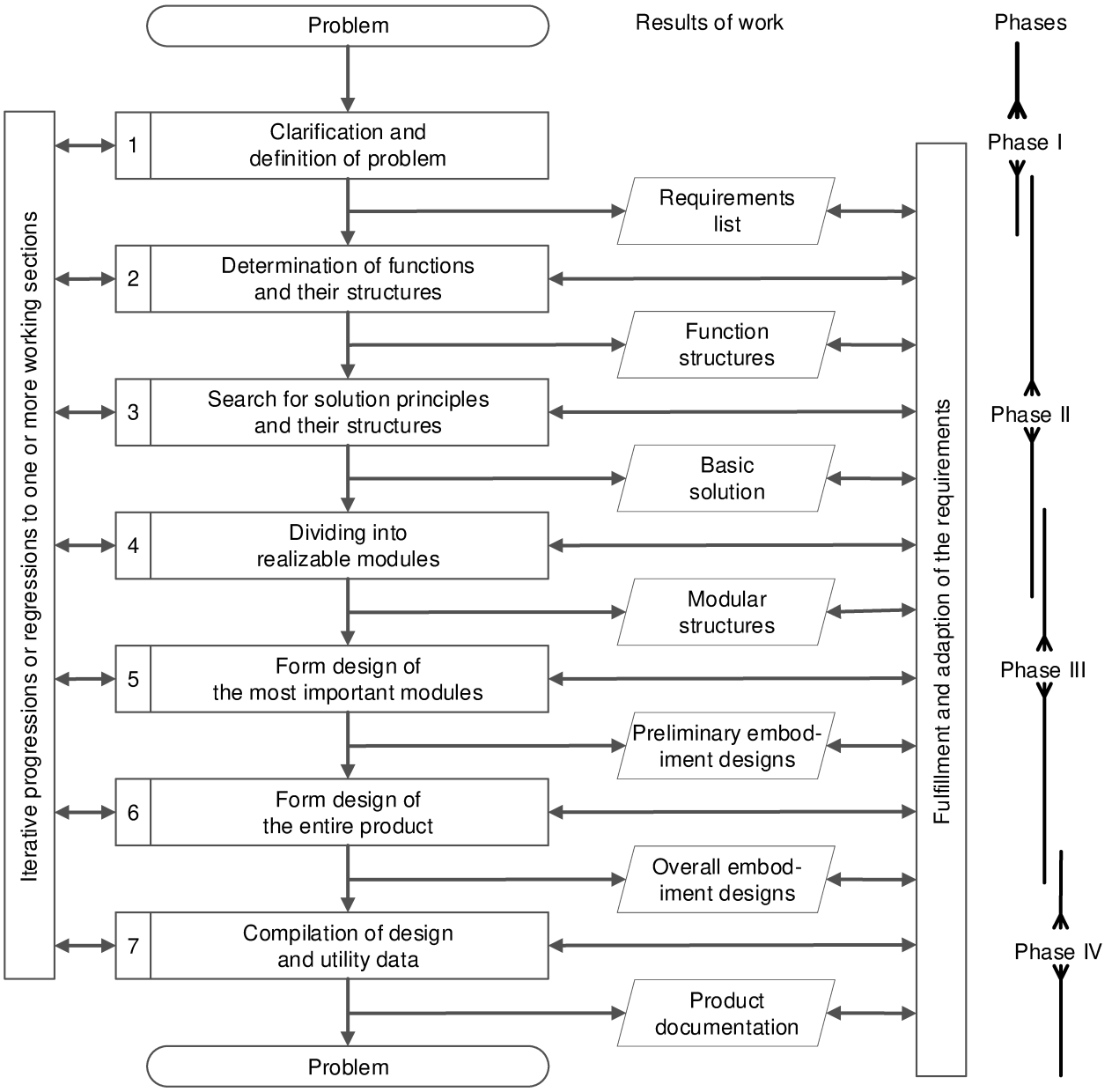


Figure 2.8: Systematic engineering design of technical systems and products according to VDI 2221 [10]

model in its characteristic form, as shown in Figure 2.9, is published by Royce [101]. The main feature of this process model is the division of the process into manageable steps. One advantage of this procedure is the possibility of documenting partial results before the transition to the subsequent phase. In the event of difficulties, there is thus a secure fallback position to the previous phase from which a restart is possible. Interestingly, Royce is today often considered the originator of the waterfall model. Actually, the waterfall model is only part of the publication [101] to describe how interaction between different phases of software development should be confined to a limited number of other steps and that

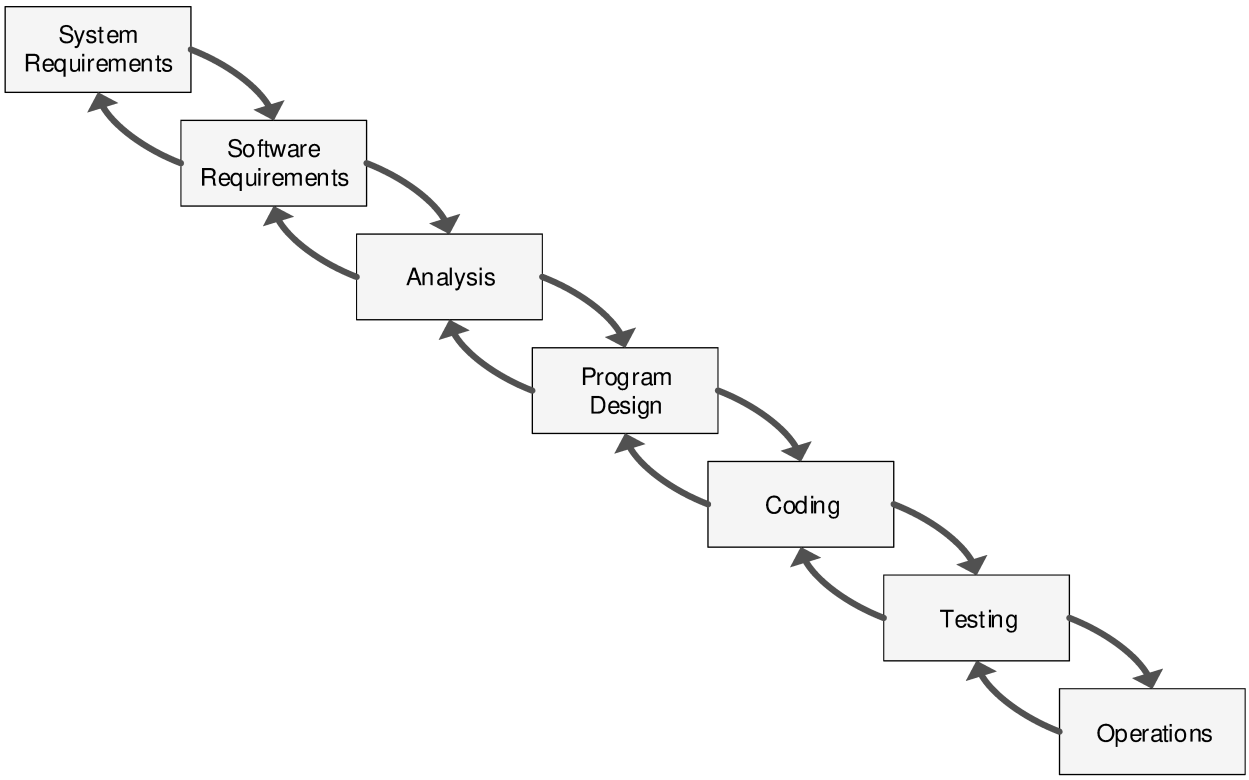


Figure 2.9: Waterfall model according to Royce [101]

there should be an effective fallback position of development results. The proposed process model is more detailed than the waterfall model and includes a path from testing via program design to software requirements to enable results from testing to be incorporated into software requirements and program design. In addition, multiple iterations of the process and associated documentation measures are proposed so that a reduction of Royce's contribution to the waterfall model is not accurate.

In 1979, Boehm proposes the V model for verification and validation of software requirements and specifications [42]. The V model was developed to reduce the number and thus the cost of specification and design errors discovered late by means of earlier identification and correction. Boehm defines verification as the establishment of "the truth of the correspondence between a software product and its specification," thus answering the informal question "Am I building the product right?" Validation is defined as "to establish the fitness or worth of a software product for its operational mission," corresponding to the informal question "Am I building the right product?" Figure 2.10 shows the associated V model for verification and validation through the software life cycle. The V model includes the phases of the software life cycle, starting with the Exploratory Conceptual phase and the subsequent design steps in the left part of the V, coding and unit testing at the bottom of the V as well as further testing activities in the right part of the V. The V model is divided into validation and verification activities by the horizontal requirements baseline that is developed in the Plans & Requirements

phase. Verification activities ensure that refinements are consistent with the requirements baseline. Validation activities above the requirements baseline include concept and requirements validation, design validation, validation of tests, etc. Problems identified in validation are resolved by changes in requirements specification.

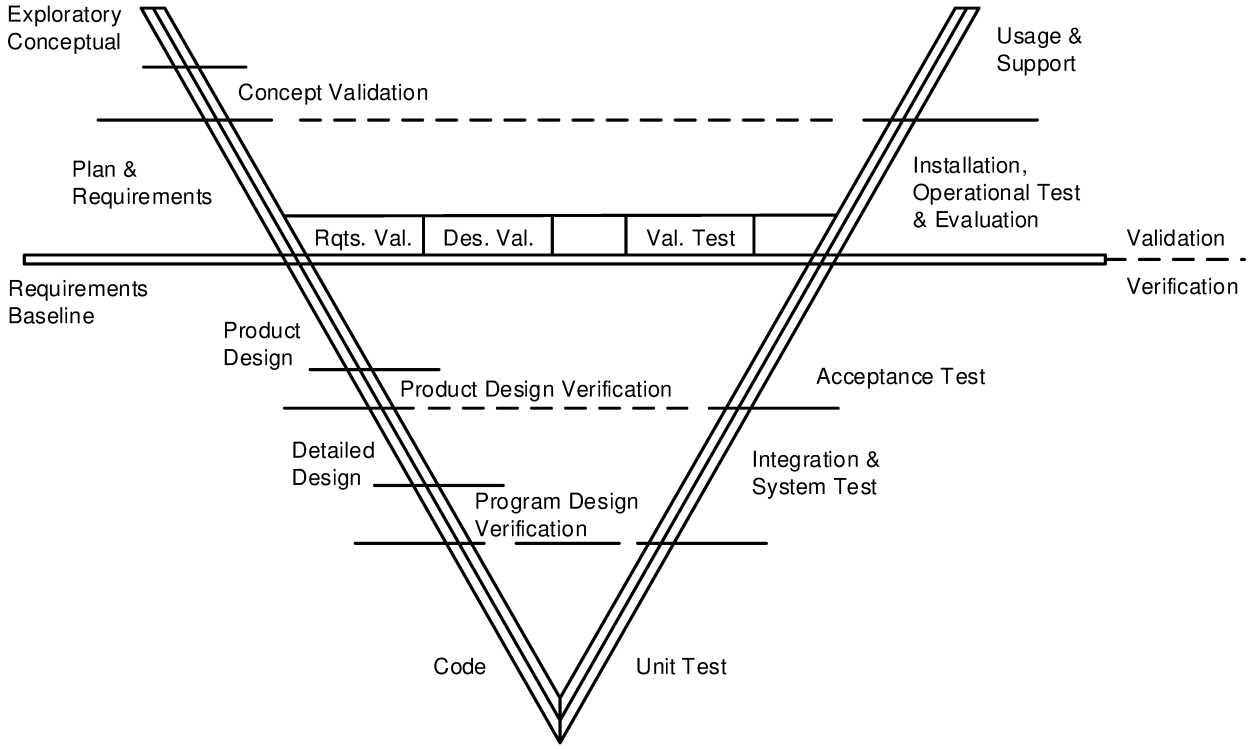


Figure 2.10: V model for verification and validation through the software life cycle according to Boehm [42]

The V model by Boehm is the basis for the development of mechatronic systems according to the VDI/VDE standard 2206 with the title *Design methodology for mechatronic systems* [3], published in 2004. Figure 1.4 on page 11 presents the proposed V model that is structured into the phases System Design in the left part of the V, domain-specific implementation at the bottom of the V, and System Integration in the right part of the V. The starting point of the V model is requirements collection in the upper left corner of Figure 1.4. Requirements describe the development task and are used for assessment of the final product. After requirements collection, an interdisciplinary solution concept is developed in the subsequent System Design phase. This solution concept includes fundamental physical and logical operating characteristics. For this purpose, the overall system function is decomposed into sub-functions resulting in a functional architecture. Functional principles and solution elements are then assigned to the sub-functions in a so-called logical architecture and the provision of the overall system function is verified. The design of functional and logical CPSoS architectures is not considered in the standard and remains a topic for possible research. As there are various options for logical architectures, the selection of the most suitable logical CPSoS architecture should also be included in the investigations. Based on the functional

architecture, system design is further detailed by the development disciplines involved in the Implementation phase at the bottom of the V, *e.g.* mechanical engineering, electrical engineering, and information technology. These development disciplines are involved in the implementation of self-contained systems. CPSoS demand the implementation of a communication architecture to connect the CS to each other and to the environment of the CPSoS. Hence, the CPSoS communication architecture implementation is a further interesting topic for research. The development results of the Implementation phase of the V model are integrated step by step into the final product in the subsequent System Integration phase. The properties of integration results are verified and validated against the design results in the left part of the V model. All phases of the V model are supported by modeling and model analysis [3].

In 2021, a newer version of the standard VDI/VDE 2206 was published that also takes into account the development of CPSs, as can be seen from the new title “Development of mechatronic and cyber-physical systems.” [16] For CPS development, three strands were added to the V model (*cf.* Figure 2.11). The outer strand represents modeling and analysis parallel to core tasks of system development in the middle strand. Requirements management is illustrated by the inner strand of the modified V model. Each strand includes sub-strands for software, electrics/electronics, mechanics, and other development disciplines. These development disciplines are considered in all main strands and illustrate the intensive networking between the disciplines that is required for the development of CPSs. The previous version of the standard from the year 2004 of the V model for the development of mechatronic systems [3] considers development disciplines exclusively in the implementation phase, that are represented as isolated arrows (*cf.* Figure 1.4 on page 11).

Another change concerns the assurance of properties. Hence, the standard version from 2004 [3] considers the assurance of properties between the System Design and System Integration phases and provides a definition for validation and verification but does not propose a specific position of validation and verification activities. The new version of the V model is more concrete and locates validation and verification. Validation is represented by a gray arrow that points to a point over the requirements elicitation that represents the validation in cooperation with relevant stakeholders. Results of system integration in the right part of the V are verified against results from the system architecture phase in the left part.

In addition, checkpoints are introduced in the transition between phases. These checkpoints should support the reflection of design and integration results and are not meant as milestones. The standard provides questions for each checkpoint to support the reflection process. As an example, the question “Are relevant interfaces for global networking of the system (with the Internet of Things and Services) known and described?” should support the reflection of design results from the architecture phase [16].

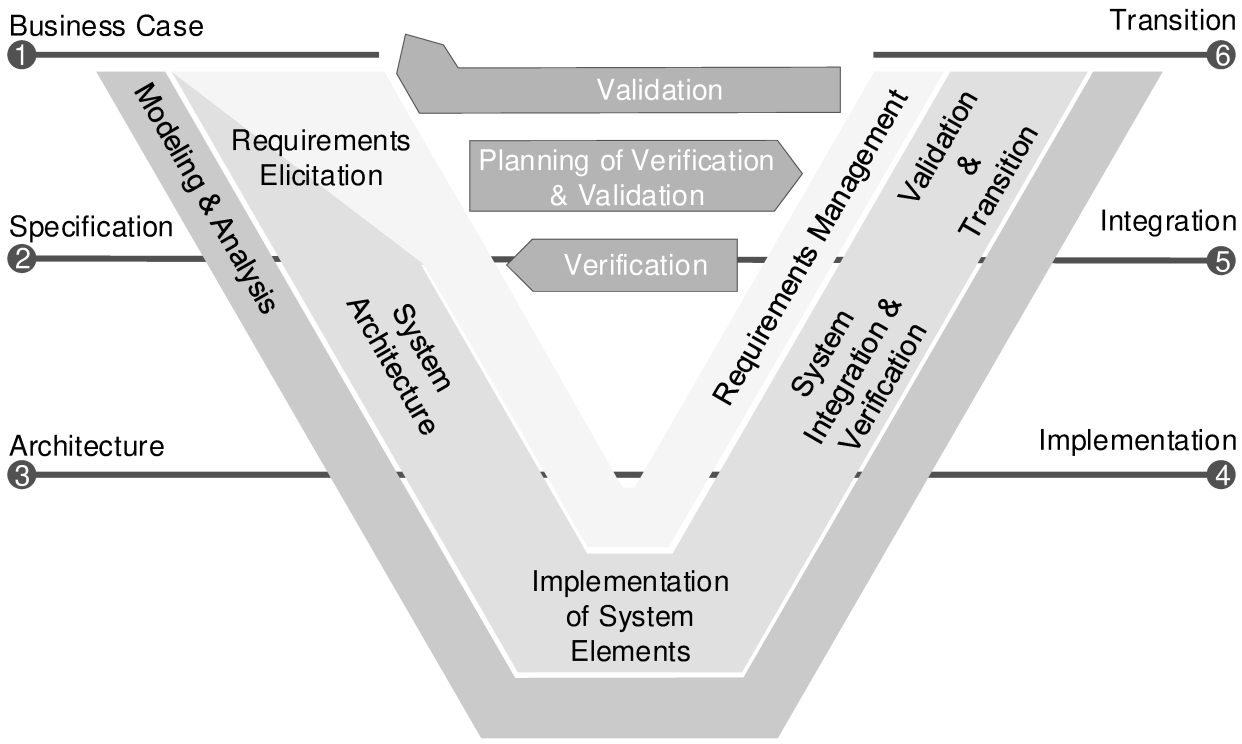


Figure 2.11: V model for the development of mechatronic systems and CPSs according to the newest version of the VDI/VDE standard 2206 from the year 2021 [16]

It was also Boehm who, based on his experiences with the waterfall model and other process models, developed the spiral model [43]. The spiral model was developed in 1988, even before the V model, and is still the basis for agile process models today. Goals of the approach are to achieve higher efficiency and better risk management than other process models. Figure 2.12 shows the spiral model. The radial dimension of the spiral model represents cumulative costs on the vertical axis and growing commitment on the horizontal axis. Progress is described by the angular dimension, and each cycle of the spiral represents a round through recurring processes that become more detailed with each cycle. As an example, a first cycle could start with a feasibility study, followed by a concept of operations in the second cycle, and a top-level requirements specification in the third cycle. Each cycle begins in the top left quadrant with determination of objectives, alternatives, and constraints for the upcoming development cycle. The objectives include, for example, a description of the performance and functions of the prototype or product developed at the end of the cycle. Alternatives consider alternative designs as well as reuse and buy decisions. Costs, schedules, and interfaces are typical examples of constraints. In the top right quadrant, areas of uncertainty and risks are identified that can be reduced by means of prototypes, simulations, models, and benchmarking. An operationally useful prototype is the basis for evolutionary development in the third quadrant located at the bottom right of the spiral model. At the end of each spiral model cycle there is a review for assessment of the prototypes and products developed in the previous cycle. Based on the assessment results, the next cycle is

planned in the bottom left quadrant, *e.g.* in the form of a requirements plan, development plan, or integration and test plan. The cyclical performance of reviews makes it possible to identify and correct undesirable developments at an early stage.

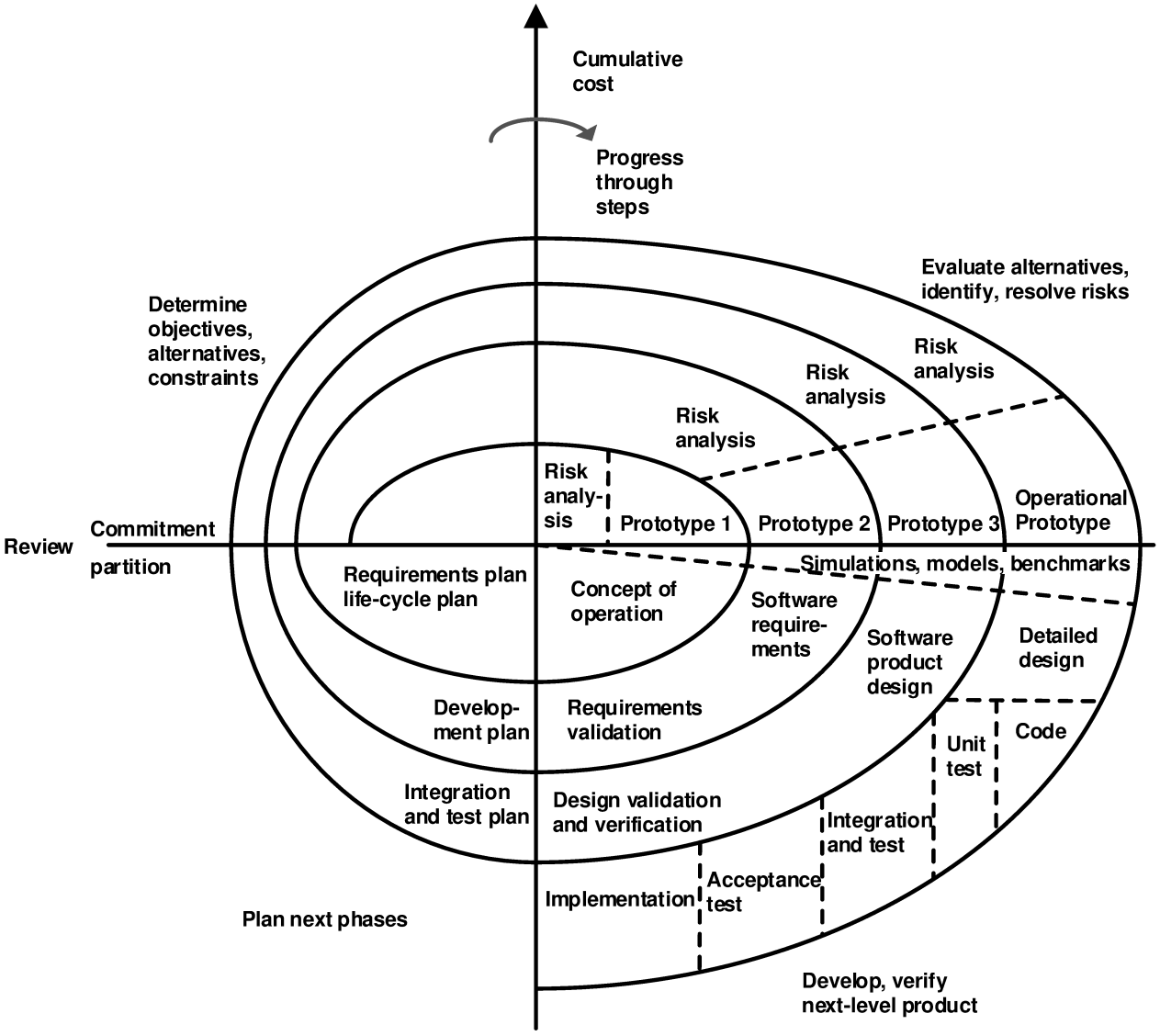


Figure 2.12: Spiral model [43]

2.3 The Rise and Spread of Cyber-Physical Systems of Systems in Aviation

With the advent of smartphones, the spread of CPSoS in aviation has also seen a breakthrough. Smartphones enable the digitally supported passenger journey, illustrated in Figure 1.1 on page 7. However, this is only one aspect of CPSoS in aviation. As shown in Figure 1.2 on page 8, it is not only passengers who benefit from the rise of CPSoS, but other stakeholders as well, e.g. in catering, logistics, and maintenance processes. Drivers

for the rise of CPSoS are advances in microelectronics [63, 64] and in-flight connectivity [51]. Since these prerequisites have been created, global aviation is predestined for the worldwide spread of CPSs and CPSoS.

The importance of CPSs and the resulting demand for an adaption of existing process models is also indicated by the extension of the VDI standard 2206 [16]. The former version from 2004 [3] was suitable for the development of mechatronic systems and is revised for the development of CPSs. However, CPSoS are not considered in the new version of the standard. The V model is also used for system specification and design in aviation and is complemented by aviation-specific development processes, as aviation has special requirements with respect to safety and security. An important component of the development processes is presented in the Aerospace Recommended Practice (ARP) 4754A with the title *Guidelines for the Development of Civil Aircraft and Systems* [6]. The ARP 4754A describes a development life cycle as shown in Figure 2.13.

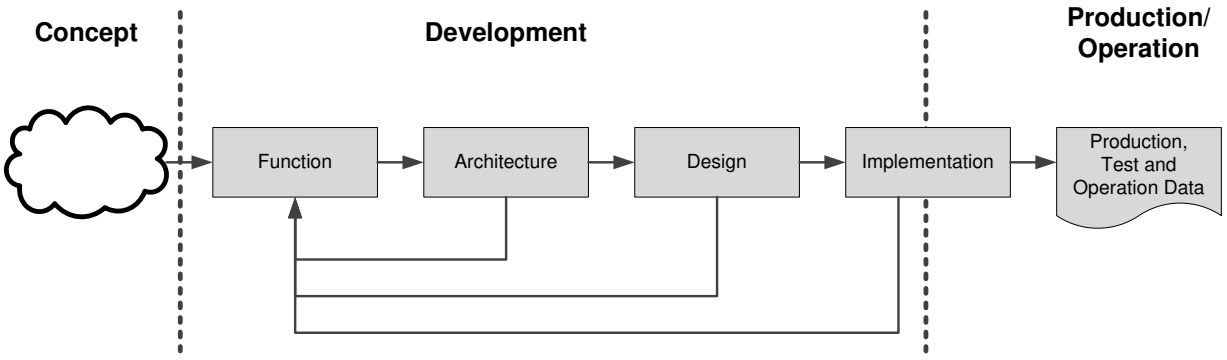


Figure 2.13: Development Life Cycle [6]

The development life cycle starts with the concept phase. This phase determines rough aircraft properties, *i.e.* aircraft performance, payload, range, size, number and location of engines, airfoil, and usage of technologies in design and manufacturing. The following development phase specifies the results of the concept phase in detail leading to a design ready for implementation in production and operation. For this purpose, build and test information must be provided to manufacturing facilities, regulatory and internal compliance data be complete and approved, and limitations, maintenance, and other operational information be made available to aircraft operators.

The development life cycle is an iterative process. Results of design phases can be used in upstream phases, thus resulting in a more detailed design with each iteration loop. The ARP 4754A focuses on the development phase, that is divided into the phases function, architecture, and design. Aircraft functions for new aircraft or additional functions for existing aircraft are developed in the function phase. Typical functions include flight control, ground steering, passenger comfort, etc. These functions are the basis for safety assessments.

Safety assessments use failure condition classification that considers the probability of occurrence and related effects, *i.e.* catastrophic, hazardous, major, minor, or no safety effects. After their identification, functions are grouped and allocated to aircraft systems. Grouping criteria depend on the functions and can be, *e.g.* implementation constraints, failure effects, performed processes, operational and support aspects as well as common input objects in the form of material, energy, or data. Additional requirements are developed during function and requirements allocation to achieve the required safety objectives. These requirements may have an impact on the aircraft-level function requirements, so a new cycle of the process may be required. The output of the function phase should be a set of requirements for each aircraft system and its interfaces.

The function development is the prerequisite for the following system architecture definition. The system architecture determines the structure and boundaries for the item design in subsequent process phases. Often, more than one so-called candidate system architecture is developed. In this case, candidate architectures are compared and evaluated by using criteria, *e.g.* technology readiness, implementation schedules, production consideration, contractual obligations, economics, and prior experience. In addition, suitable candidate architectures are evaluated based on functional and performance analyses and with respect to safety considerations according to the rules of Preliminary Aircraft Safety Assessment (PASA), Preliminary System Safety Assessment (PSSA), and Common Cause Analysis (CCA). Further information concerning safety assessment methods and analyses is provided in [6]. Additional requirements can be derived and allocated to systems and their items in each cycle of system architecture development. The impact of new requirements on previously defined requirements must be assessed. A functional architecture in that every requirement is allocated to systems and their items should be the result of the architecture development phase. Analogous to the bottom of the V model, system and item requirements are used in the development phase for domain and system specific design. Afterwards, design results are implemented. The allocation between requirements and systems as well as their items ensures that systems and items can be verified in the implementation phase.

As the name already implies, the standard is intended for the development of aircraft and aircraft systems. As services development in CPSoS becomes more important, the consideration of aircraft and system levels should be extended by a CPSoS level. One could counter that the air transport system is already an SoS and that SoS development has happened. Exemplary SoS characteristics are found in air traffic management and airport security that are centrally organized and involve the interaction of multiple independent systems. These SoS have grown historically and independently of a process model and can be characterized as directed SoS. Contemporary CPSoS in aviation are more service-oriented with less influence by a central management so that they are characterized as acknowledged and collaborative CPSoS. These types of CPSoS lack a central management and are more based on cooperation of individual systems. As acknowledged and collaborative CPSoS

are new to aviation, established process models need to be examined with respect to their suitability and adapted as needed. The following research questions are oriented on the development life cycle phases of the ARP 4754A as shown in Figure 2.14. The development of virtual CPSoS is characterized by a high degree of emergent behavior and the absence of central management or voluntary agreements (*cf.* Section 2.1 on page 15). These characteristics are in conflict with aviation-specific requirements for safety and security, so the development of virtual CPSoS should not be covered by the methodology proposed in this thesis.

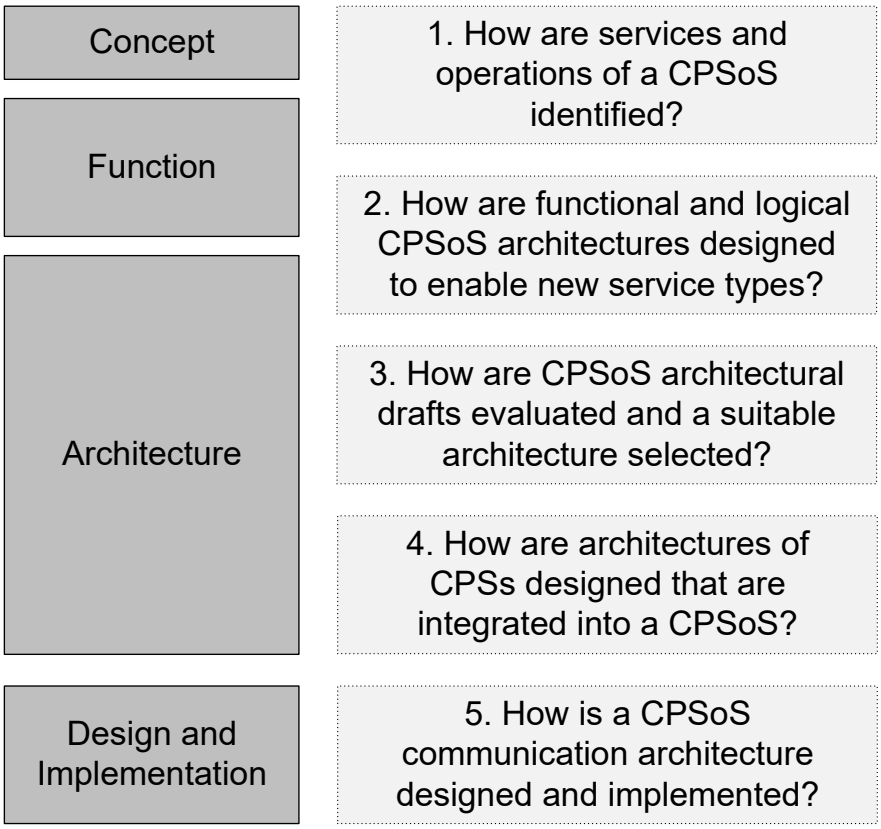


Figure 2.14: CPSoS Research Needs Identification

The first research question *How are services and operations of a CPSoS identified?* is related to the concept and function phases. Tools and methods for CPSoS development should support identification of desired services and operations of the CPSoS. They should enable stakeholder needs elicitation, the documentation of the status quo of the CPSoS, and the description of desired system behavior.

The following research question *How are functional and logical CPSoS architectures designed to enable new service types?* is allocated to the transition from function to architecture phases. An architecture represents elements of a system and their relationships. A functional architecture is based on functional blocks whose inputs and outputs are connected to each other by means of functions. Functions and sub-functions of a functional architecture

define the transformation of input flows into output flows performed by the system [18]. A functional architecture is a solution-independent system structure with required functionality. In subsequent development steps towards a physical product, functional elements of the functional architecture are allocated to logical elements. Logical elements are abstractions of physical system components and constitute the logical system architecture [69, 110]. Since there are multiple options for the allocation of functional elements to logical elements, various alternative logical architecture drafts can be developed for a single functional architecture. The set of alternative logical architecture drafts demands a selection of the most suitable architecture draft during subsequent development steps. Logical architecture drafts should be evaluated and the most suitable draft be identifiable as expressed by the third research question *How are CPSoS architectural drafts evaluated and a suitable architecture selected?*

The functions allocated to CPSs in the selected logical CPSoS architecture have to be taken into account in the design of CPS architectures besides origin CPS functions. The fourth research question *How are architectures of CPSs designed that are integrated into a CPSoS?* describes the need to consider origin CPS functions and additional CPSoS functions for the design of CPS architectures.

After the allocation of functions to logical elements has been determined by selecting a suitable logical CPSoS architecture, the data and information exchange is designed and implemented in the form of a communication architecture. A communication architecture represents the communication partners and the paths for the exchange of data and information in a CPSoS. The demand for the design and implementation of a CPSoS communication architecture is reflected in the last research question *How is a CPSoS communication architecture designed and implemented?*

3 Development Life Cycle of Cyber-physical Systems of Systems

Large-scale distribution and complexity of CPSoS results in a novel and challenging development task. It is expected that these novel challenges in CPSoS development can be solved through adapted and appropriately enhanced systems engineering. Systems Engineering is an interdisciplinary approach for the development of complex technical products in large-scale projects. The application of good systems engineering practices should fundamentally enable the identification of stakeholder needs, the design of a system and validation of the system with respect to the needs of the stakeholders [108].

Due to software tools not being available at the time, conventional Systems Engineering approaches present themselves as document-centric. They are based on a variety of documents that contain all relevant development artifacts, *e.g.* requirements, functions, test scenarios, etc. Massive and complex systems require extensive specification and design documents. The extent of documents grows with increasing system size and complexity so that clarity and consistency of development results may decrease. Requirements identification and analysis can serve as an example of this key challenge for specification and design of systems that are increasing in size and complexity. Bahns et al. [34] present a project for the identification of synergies through the joint development of systems in the entrance area of passenger aircraft. The project included five systems with individual specifications, comprising 450 pages and 100,000 words in total. Figure 3.1 shows a photo of document-based requirements analysis in that similar and contradicting requirements were identified. The size of the specifications made the requirements analysis and the design of a synergistic cabin system a very challenging task [34].

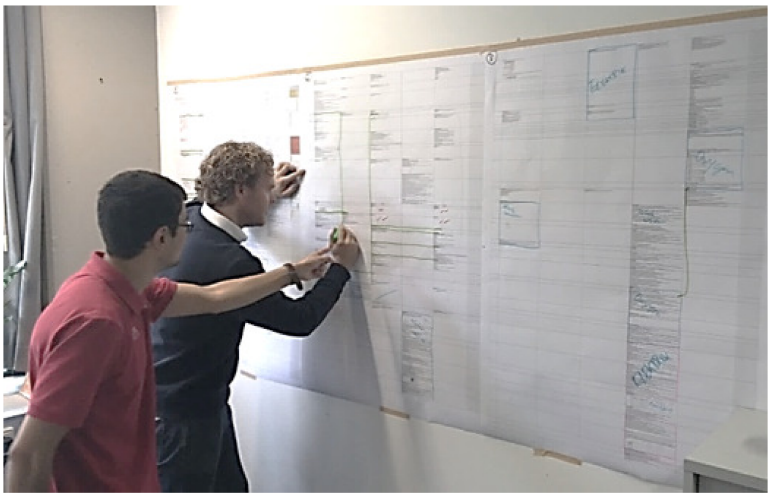


Figure 3.1: Document-based requirements analysis [34]

The error-prone document-based approach can be substituted by an approach using methods and tools of Model-based Systems Engineering (MBSE) supported by suitable software tools. Modeling and simulation of system functions are well known and established development methods [104, 108] that support all life cycle phases of the system under development. MBSE methods [67, 69, 109], currently much discussed in aviation, today pursue the goal of describing the development process itself, beginning with requirements elicitation up to system integration.

In general, a model is defined as an abstract description of reality [30]. Therefore, it can also be used to to support system requirements elicitation, design, analysis, verification, and validation activities. All relevant specification and design information is stored in a central model repository. Specific views on this information focus on certain aspects such as traceability between stakeholder and system requirements [83]. The utilization of views supports the development of complex systems with multiple dependencies between development artifacts and allows the tailoring of views on development artifacts to the needs of certain development disciplines. The central management of information in the model repository ensures consistency between the views [108].

A variety of SE and MBSE approaches for the development of systems can be found in the literature on the individual life cycle phases. Some approaches are already intended for application to CPSoS, others for self-contained systems. In aviation, system development follows the life cycle phases concept, function, architecture, design, implementation, and further phases defined in the ARP4754A standard [6]. Therefore, the following overview has been structured according to these phases. The research questions addressed in this thesis emphasize the identification of services and functions as well as the design and selection of CPSoS architectures. Since the research questions presented in Chapter 2 and visualized in Figure 2.14 on page 37 relate in particular to the life cycle phases of concept, function and architecture, a special focus is put on these phases in the Sections 3.1 and 3.2 on page 56. These start with an introduction to the fundamentals of each life cycle phase and a presentation of criteria for evaluating existing approaches with respect to their applicability to CPSoS development. For identifying suitable aspects of the approaches for CPSoS specification and design, subsequently, the approaches are presented and evaluated based on these evaluation criteria. Approaches for further detailing and implementation of the architectures in the subsequent design and implementation phases are presented in Sections 3.3 on page 70 and 3.4 on page 78.

3.1 Concept Phase

The concept phase is also denoted as the research and preliminary development phase and determines the desired properties of the system to be developed [6]. For the concept phase, various approaches for self-contained systems and SoS exist. These approaches

are often not suitable for service development in CPSoS. Often, the approaches focus on the design of a self-contained system with a limited number of interfaces to its environment. Modern services by CPSoS require intensive and wide-spread communication in CPSoS that is not taken into account in most approaches. Furthermore, context identification and description of the system under development is part of many approaches. However, the consideration of both contexts, the context of the CPSoS and the contained CPSs is not part of most approaches. The following boxes summarize important aspects of existing approaches and describe the room for improvement in their application to the development of CPSoS. Subsequently, the approaches are presented and evaluated in terms of their suitability for CPSoS development.

Summary - Concept Phase

- The identification of all relevant stakeholders should be one of the first steps in the concept phase.
- Stakeholder needs are captured through business requirements.
- The status quo is documented in system and basic architectures.
- A concept of operations describes desired system behavior.
- Non-functional requirements describe stakeholder-defined solutions and constraints.
- Measures of Effectiveness (MOEs) are determined for subsequent evaluation of architecture drafts.

Room for improvement in CPSoS development

- Most approaches are intended for the development of self-contained systems with a limited number of interfaces to other systems. Consequently, CPSoS-characteristic intensive communication with multiple interfaces for the provision of new services is not supported.
- Approaches focus on the development of systems or SoS but often do not take into account the design of new services that are based on the cooperation of CPSs in a CPSoS.
- Often, methods provide document-based methods and processes and do not provide guidance for model-based development. The increased number of interfaces in CPSoS and the associated complexity call for model-based approaches.
- Only one of the evaluated approaches considers the contexts of SoS and the systems contained.

3.1.1 Approaches for the Concept Phase

A process for the concept phase is described in the ISO/IEC/IEEE standard 15288:2015 “Systems and software engineering – System life cycle processes” [21] as part of system life cycle processes. More information about application of ISO/IEC/IEEE 15288:2015 to SoS is found in the standards ISO/IEC/IEEE 21839:2019, 21840:2019, and 21841:2019. The area of application of these standards is shown in Figure 3.2. Systems are represented by boxes

with letters “A,” “B,” and “C.” The box fill color indicates if the system is self-contained (white fill color) or contained in an SoS as CS (grey fill color). ISO/IEC/IEEE 15288:2015 forms the basis and describes system life cycle processes for self-contained systems. These processes consider, among other things, the definition of stakeholder needs and requirements as well as system requirements. Information about SoS considerations in life cycle stages of CS are found in ISO/IEC/IEEE 21839:2019. ISO/IEC/IEEE 21840:2019 contains advice for application of ISO/IEC/IEEE 15288:2015 to SoS development. SoS classification is supported by taxonomies presented in 21841:2019 [23].

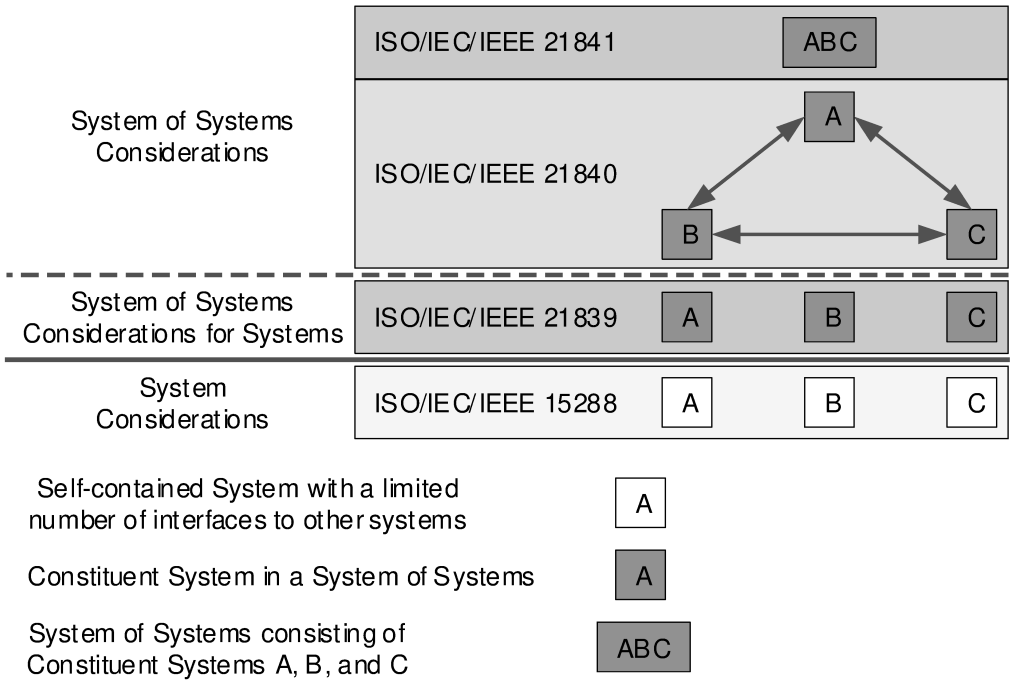


Figure 3.2: Relationships between standards for system life cycle processes and SoS considerations [23]

The following sections introduce the aforementioned standards and additional approaches for the concept phase of self-contained systems and SoS and describe the room for improvement in the application to CPSoS.

ISO/IEC/IEEE 15288:2015: Systems and Software Engineering – System Life Cycle Processes

The ISO/IEC/IEEE standard 15288:2015 describes processes for stakeholder needs and requirements definition and for system requirements definition and is suitable for the development of self-contained systems with a limited number of interfaces to other systems. The standard does not provide guidance for a model-based application, it solely suggests application of requirements management tools. It is not intended for service development in CPSoS based on the cooperation of multiple CPSs. Its Annex G.3 contains

considerations for the application to SoS development and focuses on the distinction between SoS and CS requirements. Thereby, the SoS acts as a stakeholder for CS requirements.

Suitable considerations of the ISO/IEC/IEEE standard 15288:2015 for the development of CPSoS are used for elaborating the method CPSoS Mission and Operational Scenarios Definition (*cf.* Section 4.1 on page 86). Aspects adopted in the method are the identification of all relevant stakeholders at the beginning of the development and the utilization of a concept of operations. The concept of operations describes the desired system behavior in its life cycle phases by means of activity sequences. Furthermore, the identification of stakeholder-defined solutions, implementation-relevant constraints, legacy system requirements, and affordability constraints according to the standard are part of the definition of non-functional requirements.

ISO/IEC/IEEE 21839:2019: Systems and Software Engineering – System of Systems (SoS) Considerations in Life Cycle Stages of a System

The ISO/IEC/IEEE standard 21839:2019 provides SoS considerations in the life cycle stages concept, development, production, utilization and support, and retirement of a CS. The CS under development is denoted as the System of Interest (SoI). SoS considerations to the life cycle stages concept and development are most important to the research questions addressed in this thesis and are provided by the standard in the form of questions and additional explanations. Important aspects are identification and consideration of external stakeholders and systems as well as interfaces that accompany SoI implementation into the SoS [22].

ISO/IEC/IEEE 21840:2019: Systems and Software Engineering – Guidelines for the Utilization of ISO/IEC/IEEE 15288 in the Context of System of Systems (SoS)

Guidelines for the utilization of ISO/IEC/IEEE 15288 in SoS contexts are provided in ISO/IEC/IEEE 21840:2019. However, considerations for the development of CPSoS are not given so that service development is not part of the standard and the contexts of CPSs and CPSoS are only considered in the form of CS and SoS. The standard offers a process but does not suggest a corresponding model-based approach.

Instead, focus is put on guidance concerning the application of processes described in ISO/IEC/IEEE 15288 in SoS contexts, *e.g.* stakeholder needs and requirements definition process and system requirements definition process. Not all stakeholders may be identifiable in stakeholder needs and requirements definition process in SoS contexts because of managerial and operational independence of CS. However, CS and SoS stakeholders must be considered and it must be possible to add new stakeholders and their needs. Moreover, each CS has its own life cycle and changes in CS life cycles affect SoS capabilities.

Therefore, CS life cycle stages that can lead to different and evolving SoS capabilities must be considered. In addition, constraint definition must take into account existing CS that may constrain the SoS. Finally, agreement on stakeholder requirements may be more difficult in SoS contexts, since SoS and CS stakeholder needs may be contradictory and conflicts must be resolved [23].

ISO/IEC/IEEE 21841:2019: Systems and Software Engineering – Taxonomy of Systems of Systems

ISO/IEC/IEEE 21841:2019 provides a taxonomy for SoS to support communication between stakeholders in SoSE but does not offer a process or a method for CPSoS development. A taxonomy is defined by the standard as a "scheme that partitions a body of knowledge and defines the relationships among the pieces" [24]. The standard incorporates directed, collaborative, and virtual SoS classifications defined by Maier [89] and acknowledged by Dahmann and Baldwin [49] that are already explained in Section 2.1 and in Figure 2.3 and can be considered as a supplement to the ISO/IEC/IEEE standard 21840:2019 [23].

In conclusion, the introduced standards describe processes for the elicitation of stakeholder needs and their transformation into stakeholder and system requirements. Usually, various stakeholders are affected by CPSoS development and numerous stakeholder needs and requirements must be taken into account. Application of document-based approaches in the context of complex systems is limited in terms of traceability and consistency, as already introduced in Section 2.3 on page 34. The introduced standards suggest obtainment and acquisition of so-called enabling systems that support stakeholder needs and requirements definition [21] and leave room for advice with respect to method and tool support that is elaborated in this thesis.

Systems Modeling Toolbox (SYSMOD)

Weilkiens has developed the Systems Modeling Toolbox (SYSMOD) that provides a collection of methods and processes for MBSE. As already indicated by the name, the SYSMOD approach strongly supports MBSE. The focus lies on the development of a self-contained system instead of a CPSoS or services that are provided by a CPSoS. However, the environment of the system under development is taken into account by identifying the system context that describes interfaces and interactions of the system with actors and external systems. The basic architecture concept can be used to describe the current context of a system that should be adapted for integration into a CPSoS. A context for CPSoS development is not considered. Part of SYSMOD is the SYSMOD analysis process, shown in a SysML activity diagram in Figure 3.3. Figure 3.4 provides an overview of common elements in SysML activity diagrams.

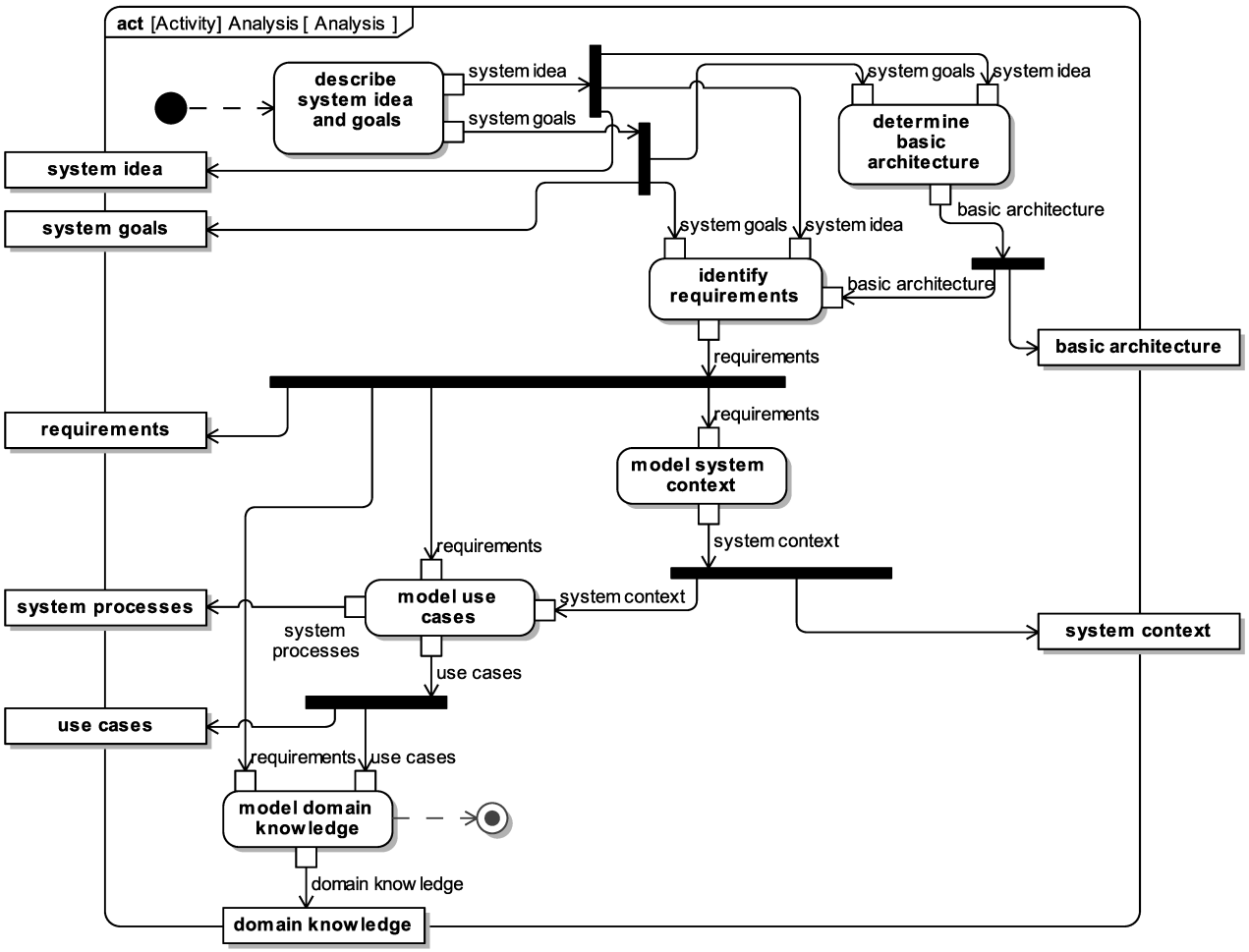


Figure 3.3: SYSMOD analysis process by Weilkiens [109], modeled in a SysML activity diagram

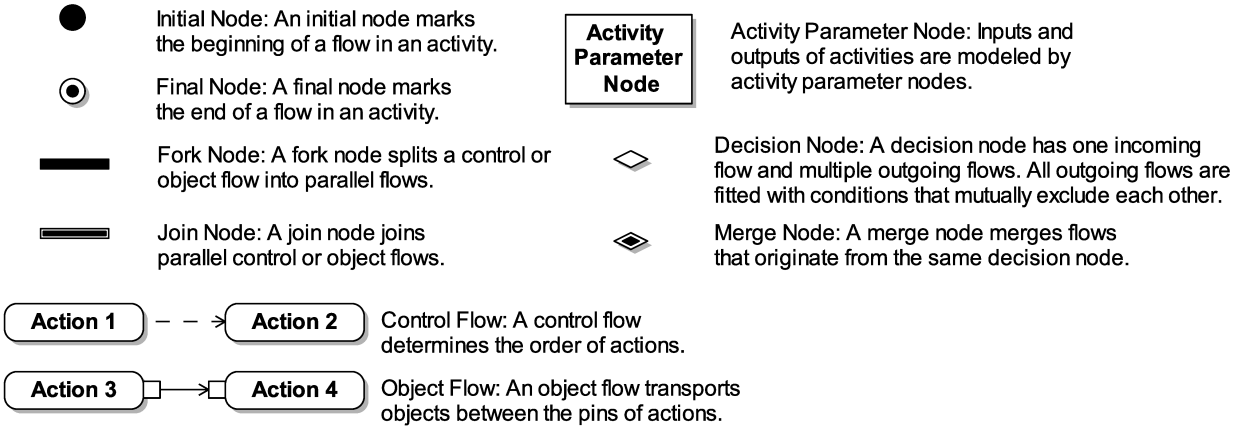


Figure 3.4: Common elements in SysML activity diagrams [12]

The SYSMOD analysis process begins with a description of the system idea and system

goals to establish a common understanding between stakeholders and the development team. Often, the development task is based on an existing system or has to integrate technologies that are predefined, *e.g.* by customers. These boundary conditions must be respected during the system analysis process and are documented in the basic architecture. Together with the system idea and goals from the first process step, the basic architecture is input for the requirements definition step. Relevant stakeholders are identified and their requirements are documented. In the following, interfaces of the system to stakeholders and other systems are identified and modeled in the system context in order to consider the environment and its effects on the system. Therefore, stakeholders are modeled as actors and other systems are modeled as so-called external systems. They are connected to the system under development by means of object flows. Similar to the Object-Oriented Systems Engineering Method (OOSEM), that is introduced below, the previously determined requirements are refined by use cases. Based on requirements and the system context, use cases of the system are identified systematically. The use cases are further refined by activities and described in activity diagrams in order to define system processes and their connections. The final step of the SYSMOD analysis process is modeling domain knowledge. Domain knowledge describes objects that are transferred in the form of object flows in system processes. Domain knowledge is modeled to obtain a common understanding between stakeholders [109].

The concept of documenting the status quo in a basic architecture is integrated into the method CPSoS Mission and Operational Scenarios Definition by defining an information network structure (*cf.* Section 4.1.2 on page 88). The information network structure emphasizes existing communication interfaces and exchange of data and information. Refining use cases with activities is also part of OOSEM, presented below, and in the Functional Architectures for Systems (FAS) method, introduced in Section 3.2.1 on page 57. Use case refinement is adapted to CPSoS needs by considering interfaces between multiple CPSs in the method CPSoS Function Identification and Architecture Development, as described in detail in Section 4.2.1 on page 97.

Object-Oriented Systems Engineering Method (OOSEM)

Friedenthal et al. present the Object-Oriented Systems Engineering Method (OOSEM). The focus of OOSEM lies in the improvement of existing enterprise systems. The service development aspect is considered to a lesser extent and may be integrated into enterprise use case definition. Different contexts of CPSs and the CPSoS are not considered in OOSEM. However, calculating Measures of Effectiveness (MOEs) is presented in a model-based approach using SysML parametric diagrams and can be utilized for the CPSoS domain.

OOSEM contains processes for system management, specification, and design as well as processes for systems integration and verification. It is intended for the application

to two hierarchies, *e.g.* SoS and system level or system and system element level respectively. The OOSEM process for specification and design contains a subprocess for stakeholder needs analysis that is shown in Figure 3.5 and described in the following. The stakeholder needs analysis process supports understanding stakeholder problems, specifying their requirements, and setting the context for the system or systems under development.

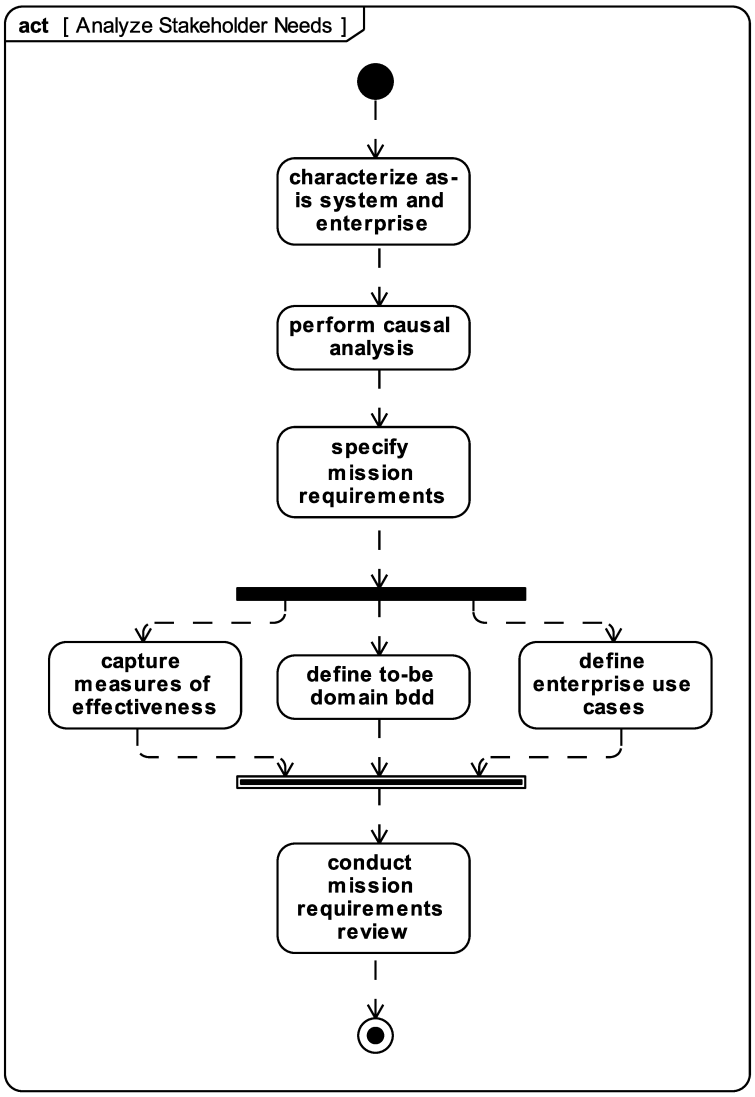


Figure 3.5: Process for stakeholder needs analysis by Friedenthal et al. [69], modeled in a SysML activity diagram

The process begins with a characterization of the system and enterprise. The causal analysis in the second process step identifies potential improvements by taking into account all relevant stakeholder perspectives. Potential improvements are used for specifying mission requirements that are refined by enterprise use cases. Enterprise use cases can be further detailed by a textual description or an activity and related SysML activity diagrams. A to-be domain block definition diagram and measures of effectiveness are determined parallel to

the definition of enterprise use cases. The to-be domain block definition diagram describes the system and its integration into the enterprise as well as its environment with stakeholders and other systems, similar to a stakeholder and system context according to SYSMOD. An example of a system under development and its enterprise integration is provided in the form of a so-called enhanced security system that supports the provision of emergency responses and is integrated into a security enterprise. The to-be block definition diagram includes the enhanced security system, the security enterprise, the site that is protected as well as its occupants and possible intruders. Measures of effectiveness are considered as “mission-level performance requirements” [69] and enable a quantitative evaluation with respect to stakeholder expectations during the development process. A final mission requirements review is intended to validate the results of stakeholder needs analysis with stakeholders [69].

MOEs are used for a quantitative evaluation of architecture drafts developed according to OOSEM. The related process for MOE identification and calculation is intended for the evaluation of self-contained systems but can be adopted for the CPSoS domain. Section 4.1.4 on page 95 describes how MOEs are determined for a CPSoS by application of the method CPSoS Mission and Operational Scenarios Definition. These MOEs are used for CPSoS architecture draft evaluation according to the method CPSoS Architecture Evaluation, presented in Section 4.3 on page 103.

System of Systems Approach to Context-based Requirements Engineering (SoSACRE)

Holt et al. provide the Approach to Context-based Requirements Engineering (ACRE) for MBSE and extend it to SoSACRE for application to MBSE for the development of SoS. The provided method and processes primarily focus on the development of SoS and CS independent of the service concept and cyber-physical characteristics of contemporary CPSoS. SoSACRE combines contexts of multiple CS in a single diagram. Depending on the size of the SoS, this approach can result in confusingly large diagrams and can be modeled differently, *e.g.* by using multiple diagrams or dependency matrices.

The context is a fundamental part of the ACRE ontology and is comparable to a point of view. There are different points of view for requirements definition represented by different contexts. Figure 3.6 provides an overview of the ACRE ontology and indicates that the context is incomplete by definition. Typical contexts are the system context and the stakeholder context.

The stakeholder context represents the view of relevant stakeholders of the system. A stakeholder is defined as a person or thing having a role and an interest in the system under development. Exemplary stakeholders are users, operators, sponsors, management, and engineering as well as standards and regulations. Different system hierarchy levels are

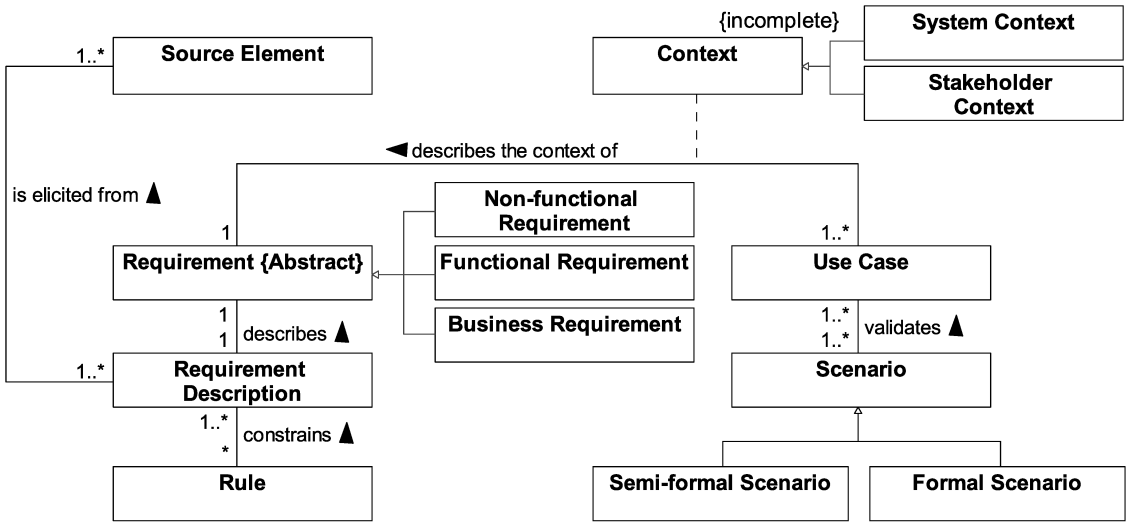


Figure 3.6: *Approach to Context-based Requirements Engineering* (ACRE) ontology [81]

considered in system contexts. System contexts are an important aspect if a system can be broken down into subsystems, assemblies, and components with different development teams and production facilities. Often, these system breakdowns already exist before requirements definition. Related viewpoints are captured in the system context and need to be taken into account during requirements definition. Other contexts are, *e.g.* business, project, program, and System of Systems contexts [81].

Requirements can be interpreted differently depending on alternative viewpoints. Therefore, the use case concept is introduced. Use cases describe the context of requirements and put requirements into a context to understand their meaning. As a generalization for requirements, an abstract requirement is part of the ACRE ontology. Business, functional, and non-functional requirements are specializations of this abstract requirement. Business needs and required capabilities from business perspective are expressed in business requirements, including business drivers and describing needs and capabilities from a high level. Business requirements are the basis for functional requirements that describe an observable result to interacting stakeholders and other systems. Functional requirements should describe required system functions as solution-neutrally as possible to maintain a wide range of possible solutions and not exclude possible solutions that have not been evaluated in detail. Non-functional requirements describe constraints that must be taken into account when realizing functional requirements and arise from law, standards, system performance specifications and preselection of technology.

All requirement specializations have a requirements description in common. As indicated by their name, requirements descriptions describe requirements in a textual manner based on source elements. Source elements can be anything that is used for requirements elicitation, *e.g.* workshop and discussion results, standards, existing systems, and information sources

in the form of books, articles, etc. Rules constrain requirements descriptions in order to reduce ambiguities and the risk of misconceptions. An exemplary rule may constrain requirement formulation using modal verbs such as “shall” for mandatory requirements, “may” for expression of permissible courses of action, “can” for possibilities, etc. Scenarios are introduced for validating and demonstrating that use cases and requirements are met. These scenarios should cover different situations that may occur in connection with the use case. Holt, Perry, and Brownsword differentiate between semi-formal and formal scenarios. Semi-formal scenarios are documented in SysML sequence diagrams showing the interaction between stakeholders and the system at stakeholder-level scenarios and interaction between system elements at system-level scenarios. Formal scenarios utilize SysML parametric constraints. Connection of parametric constraints in a SysML parametric diagram enables calculation of simulation results and trade-offs [81].

Holt et al. modify and enhance the ACRE ontology in the course of the COMPASS project. Part of the COMPASS project was the integration of engineering notations, methods, and tools for the support of SoS modeling and analysis. Holt et al. address the consideration of SoS and CS perspectives as well as present and future SoS needs according to Lewis et al. [88] in the COMPASS ontology [80], as presented in Figure 3.7.

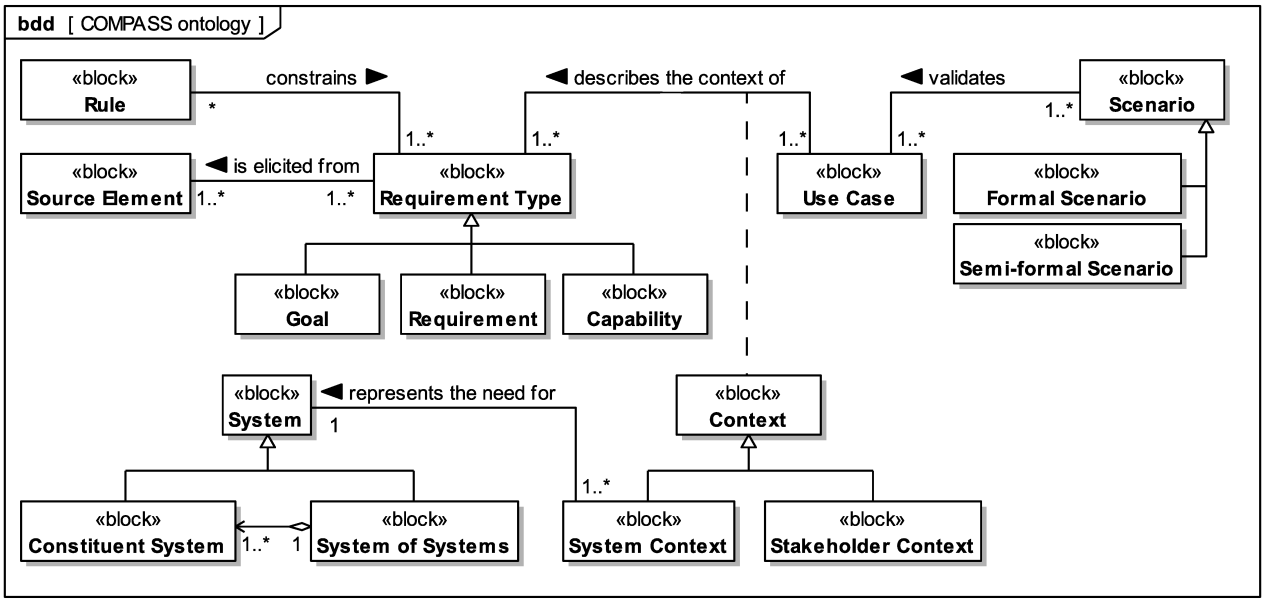


Figure 3.7: COMPASS ontology [80], modeled in a SysML block definition diagram

The COMPASS ontology also contains a stakeholder context and a system context. This system context represents the need for a system (bottom left in Figure 3.7) that can be either a Constituent System or a System of Systems. Details about Systems, Constituent Systems, and Systems of Systems are provided in a subset of the MBSE ontology, depicted in Figure 3.8.

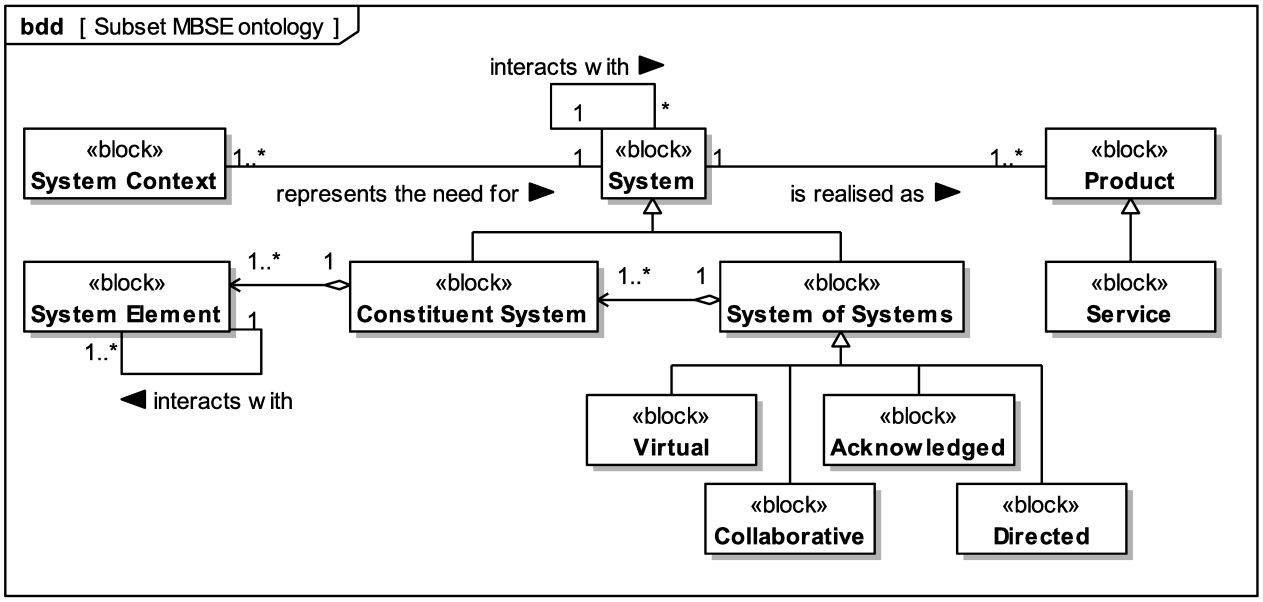


Figure 3.8: Subset of the MBSE ontology by [79], modeled in a SysML block definition diagram

This subset adds the SoS types virtual, collaborative, acknowledged, and directed SoS according to Maier [89] and Dahmann and Baldwin [49] as specializations of the System of Systems. Furthermore, services are added to the ontology and are considered as a specialization of a product (top right in Figure 3.8).

In addition to modeling concepts in the form of the MBSE ontology, Holt and Perry suggest processes for stakeholder needs elicitation and requirements definition based on the ISO/IEC standard 15288 from the year 2008 [20]. The processes are described by means of SysML activity diagrams and are structured into the requirements definition process, the context process, and the requirements elicitation process. The requirements definition process begins with identifying source elements, SoS stakeholder and SoS constituent system contexts. Then, the SoS requirements elicitation process is invoked. Needs are identified, elicited, and reviewed during the requirements elicitation process. Afterwards, the requirements definition process is followed by the SoS context process for both the SoS and relevant CS. Finally, interactions between the SoS and CS are identified, modeled, and reviewed [79].

Although the SoSACRE does not fully address the challenges in CPSoS engineering, the concepts of business and non-functional requirements as well as system and stakeholder contexts are integrated into the method CPSoS Mission and Operational Scenarios Definition. In addition, parts of the ontology for the relationships between Systems, Constituent Systems, and Systems of Systems are adopted for the concept for the relationships between SoS, CPSs, and CPSoS in Figure 2.4 on page 25.

Requirements Engineering for Systems of Systems

Lewis et al. [88] assess Requirements Engineering techniques for self-contained systems as inadequate for SoS. Insights from these techniques are used in their approach for Requirements Engineering in SoS. It emphasizes interfaces of the SoS and between CS in context identification and interaction understanding. Considered data-centric and process-centric interactions between CS meet the communication-intensive characteristic interaction between CPSs in a CPSoS. It is remarkable that the suggested approach does not include services provided by the SoS. Instead, SoS and CS goals are used. Moreover, the approach does not offer any advice with respect to a model-based application.

Lewis et al. identify two aspects that differentiate Requirements Engineering for SoS from Requirements Engineering for self-contained systems. Furthermore, SoS and CS perspective must be taken into account in Requirements Engineering for SoS. Therefore, required SoS capabilities must be identified and resulting requirements for CS must be derived. Second, independent needs of CS must be respected, especially if the CS was originally designed as a self-contained system with individual requirements and use cases. The second aspect corresponds to the evolutionary character of SoS by demanding the consideration of present and future SoS needs for defining required CS capabilities. Consequently, CS must be designed flexibly to respond to changes in SoS environment [88]. To meet these aspects for Requirements Engineering for SoS, Lewis et al. propose a five-step approach, which is presented in Figure 3.9.

Step 1) starts with SoS context identification in that the SoS type, *i.e.* directed, collaborative, acknowledged, and virtual, according to Maier [89] and Dahmann and Baldwin [49], as well as the SoS and system environment are identified. Requirements definition should take into account the degree of operational and managerial independence indicated by the SoS type. Analysis of SoS and system environment considers entities that are interacting with the SoS and its CS as well as influences and evolution of elements outside the SoS. Goals of the SoS and CS are identified in step 2). The goals are compared to each other, common and conflicting goals are identified by using commonality analyses and goal trees, and conflicts are resolved. Afterwards, interactions between CS are collected and analyzed in step 3). The basis for interactions identification can be the consideration of use cases. After their identification, interactions can be analyzed in a process, data, and resource-centric manner. Afterwards, individual system capabilities and constraints are identified in step 4). Finally, the gap between SoS goals and existing capabilities is analyzed in step 5). For this purpose, goals are mapped to existing capabilities, and dependencies between capabilities are identified. As a result, existing CS capabilities may be modified and missing capabilities developed to meet SoS goals [88].

The identification of interaction between CPSs is incorporated into the method CPSoS Mission and Operational Scenarios Definition by means of operational scenarios that describe

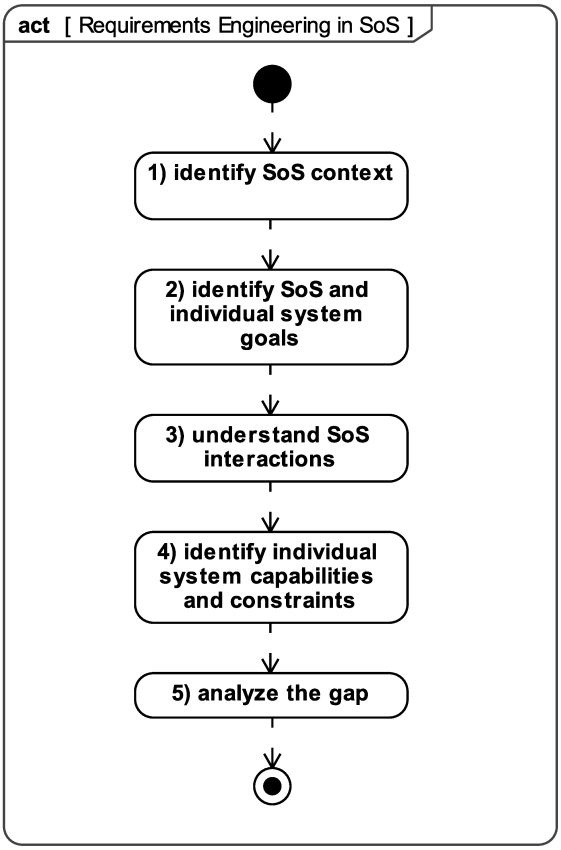


Figure 3.9: Process for Requirements Engineering for SoS by Lewis et al. [88], modeled in a SysML activity diagram

the desired behavior between CPSs. Another aspect integrated into the method is the identification of constraints and their modeling as non-functional requirements according to SoSACRE [79].

Unified Architecture Framework (UAF)

Military organizations with multiple weapon and defense systems often show SoS characteristics, *e.g.* a certain degree of managerial and operational independence of CS and wide distribution of the contained systems. Frameworks such as the Department of Defense Architecture Framework (DoDAF) and the Ministry of Defence Architecture Framework (MODAF) are developed for modeling architectures of military SoS. The Unified Profile for Department of Defense Architecture Framework and Ministry of Defence Architecture Framework (UPDM) consolidates concepts from DoDAF and MODAF to improve interoperability between the frameworks. UPDM is used by many organizations that are not part of the military domain so that there is a demand for a more general non-military framework in the form of the Unified Architecture Framework (UAF). UAF is an industry compatible evolution of UPDM. All elements and diagrams in UAF are specializations of SysML elements and diagrams so that the UAF is classified as a SysML stereotype profile [14].

Individual CPS and SoS contexts can be well modeled using the UAF and the SysML. UAF is a SysML stereotype profile so that SysML models can be integrated and connected to UAF models. The extent of compatibility between UAF and SysML models depends on the modeling tool and the tool vendor-specific implementation of UAF and SysML. Interfaces and communication between CPSs can be designed and modeled in strategic and operational domains in combination with the connectivity model kind. The whole framework is intended for model-based application. However, the framework lacks a process for CPSoS development and the extensive UAF specification requires much effort for familiarization and restricts the freedom to model CPSoS, *e.g.* the simulation of system behavior is not possible.

The UAF view specifications are organized in a grid, shown as an extract in Figure 3.10, that represents domains in rows and model kinds in columns.

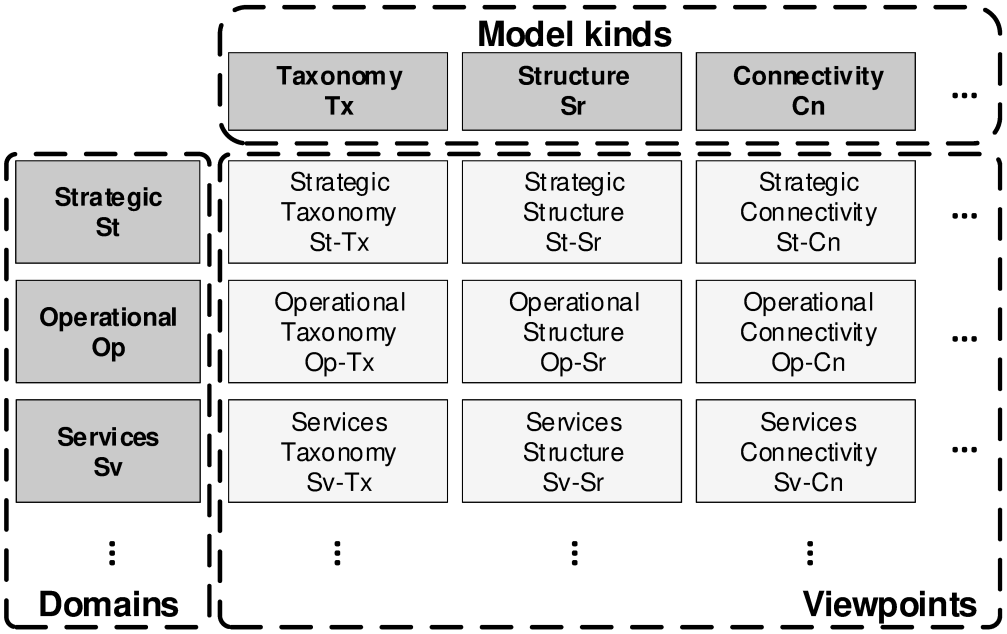


Figure 3.10: UAF grid [19]

Domains categorize the information in the model repository. Exemplary domains are meta-data, strategic, operational, services, personnel, resources, security, projects, standards, actual resources, dictionary, summary and overview as well as requirements. UAF model kinds in grid columns enable different ways of representing the model information. UAF model kinds encompass taxonomy, structure, connectivity, processes, states, interaction scenarios, information, parameters, constraints, roadmap, and traceability. The model kinds concept can be illustrated by means of an example: Elements in the structure model kind can be represented in diagrams that are based on the SysML block definition diagram. Connections between these structural elements are illustrated in the connectivity model kind using diagrams that are based on the SysML internal block diagram. Matrix cells are

intersections of domains and model kinds. Each cell is a viewpoint specification, *e.g.* the intersection of the strategic row and the structure column is the strategic structure viewpoint specification [19].

3.1.2 Evaluation of Approaches for the Concept Phase

The methods presented in the previous section are evaluated with respect to evaluation criteria that are derived from SoS characteristics (*cf.* Section 2.1 on page 15). Table 3.1 summarizes the evaluation results. As the ISO/IEC/IEEE standards 21839:2019, 21840:2019, and 21841:2019 are a set about SoS considerations for the ISO/IEC/IEEE standard 15288:2015, they are evaluated together.

Approaches for the Concept Phase	**Evaluation Criteria**				
	Service development	Consideration of individual CPS and CPSoS contexts	Interfaces and communication between multiple CPSs	Model-based	Provision of a process
ISO/IEC/IEEE 15288:2015: Systems and Software Engineering - System Life Cycle Processes	○	◑	○	○	●
ISO/IEC/IEEE 21839:2019: Systems and Software Engineering - System of Systems (SoS) Considerations in Life Cycle Stages of a System ISO/IEC/IEEE 21840:2019: Systems and Software Engineering - Guidelines for the Utilization of ISO/IEC/IEEE 15288 in the Context of System of Systems (SoS) ISO/IEC/IEEE 21841:2019: Systems and Software Engineering - Taxonomy of Systems of Systems	○	◑	◕	○	●
Systems Modeling Toolbox (SYSMOD)	◑	○	◕	●	●
Object-Oriented Systems Engineering Method (OOSEM)	◑	○	◔	●	●
System of Systems Approach to Context-based Requirements Engineering (SoSACRE)	◔	●	◑	●	●
Requirements Engineering for Systems of Systems by Lewis et al.	◔	◕	◑	○	●
Unified Architecture Framework (UAF)	●	◑	●	●	○

● fulfilled ◕ mostly fulfilled ◑ partially fulfilled ◔ fulfilled to a lesser extent ○ not fulfilled

Table 3.1: Evaluation of methods with respect to suitability for the concept phase in CPSoS development

Service development: As the offer of innovative services gains in importance, the service orientation of the approaches is evaluated. Only the UAF considers services completely while other approaches neglect services for the most part.

Consideration of individual CPS and CPSoS contexts: Existing CPSs have their own purpose and context before their integration into a CPSoS. This context is taken into account by the SoSACRE and partially considered by the standards and other approaches.

Interfaces and communication between multiple CPSs: CPSoS-characteristic communication and exchange of data and information requires the consideration of multiple interfaces between CPSs. Only the UAF completely takes into account these interfaces while other approaches are only partially suitable for the design and modeling of interfaces between CPSs.

Model-based: The extent and the complexity of CPSoS call for a model-based approach. Many methods provide a model-based approach. However, they do not meet all other evaluation criteria so that they cannot be adopted for the design of CPSoS without modification.

Provision of a process: All methods provide a process for their application, with the exception of UAF.

The summary shows that none of the methods meets all desired criteria for the concept phase of CPSoS to address the first research question “How are services and operations of a CPSoS identified?” (*cf.* Section 2.3 on page 34). For this reason, none of the methods can be completely adopted for the design of CPSoS. Nevertheless, partial aspects of the methods can be adapted and combined, *e.g.* consideration of CS and SoS contexts in SoSACRE [79] and thoughts about service development in UAF [14].

3.2 Function and Architecture Phases

Corresponding to the concept phase in Section 3.1, this section describes fundamentals of functional analysis and architecture development in SoS. Methods and approaches for functional analysis and architecture development are presented and evaluated with respect to evaluation criteria to identify gaps in CPSoS development.

Summary - Function and Architecture Phases

- Functions are relationships between inputs and outputs of a system. Examples for inputs and outputs are signals, energy, and matter [66].
- Functions can be identified by refining use cases with activities and grouping of similar activities.
- The topmost system function can be broken down into subfunctions resulting in a hierarchical decomposition.
- An architecture determines system elements and their relationship to each other [7].
- A functional architecture is based on functional blocks whose inputs and outputs are linked by functions.

Room for improvement in CPSoS development

- Existing approaches are not suitable for the functional analysis and architecture design of CPSoS, as they do not differentiate between CPSoS and CPS functions.
- Most approaches do not consider interfaces in detail so that the CPSoS-characteristic interfaces for the exchange of data and information cannot be sufficiently specified.
- Often, approaches do not support the identification of redundant functions so that a large set of functions emerges, resulting in unnecessarily extensive architectures.
- Many methods do not provide a systematic approach for the derivation of architectures from functions.
- Only two methods support the selection of the most suitable architecture draft with an approach for architecture evaluation.

3.2.1 Approaches for the Function and Architecture Phases

As described in the box above, it is remarkable that most approaches consider only one level for functional analysis and architecture design. Figure 3.11 illustrates the scope of established approaches and the required scope. Established approaches can be either applied to the CPSoS level (top) without connection to the CPS level (bottom) or to the CPS level without consideration of the CPSoS level. Consequently, not all functional elements are identified and form the functional architecture. As an example, the functional element FE1.1 represents a function that is required at the CPSoS level. If an established approach is applied to the CPS level, the FE1.1 is not identified and considered in the functional architecture. Similarly, the FE1.2 is required by stakeholders of the CPS 1 and not considered if an established approach is applied to the CPSoS level. The required scope of an approach for functional analysis and architecture development for CPSoS and CPSs is shown in the center of Figure 3.11. This approach considers functions that are required either at CPSoS, CPS, or both levels.

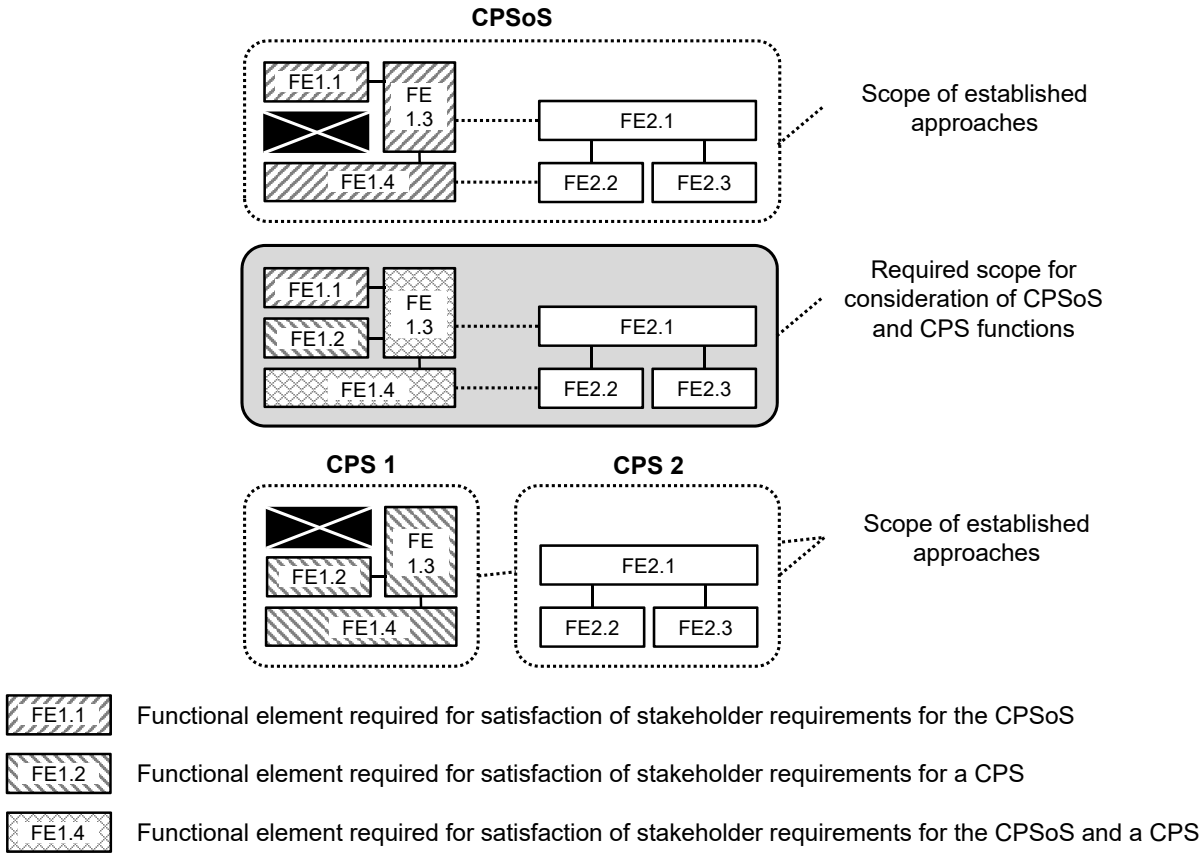


Figure 3.11: Scope of approaches for functional analysis and architecture design

The following sections introduce approaches for the function and architecture phases of self-contained systems and SoS and describe the need for solutions for the application to CPSoS.

ISO/IEC/IEEE 15288:2015: Systems and Software Engineering – System Life Cycle Processes

The ISO/IEC/IEEE standard 15288:2015 suggests a generic approach for architecture development. Although a process for architecture development is described, functional considerations and interfaces between functions have only a low priority. For this reason, the identification of redundant functions and the systematic derivation of an architecture from the functions are not supported. Instead, functions are only part of the suggested architecture design process and are considered parallel to concepts, properties, characteristics, behaviors, and constraints of system elements so that the designed architecture is not primarily driven by functions. Furthermore, the standard lacks a suggestion for a model-based approach.

The architecture development process according to the ISO/IEC/IEEE standard 15288:2015 should be conducted after requirements definition and is used for the development of

architecture drafts, the selection of suitable architecture drafts, and to describe these drafts in appropriate views. The activity diagram in Figure 3.12 depicts the associated main process steps.

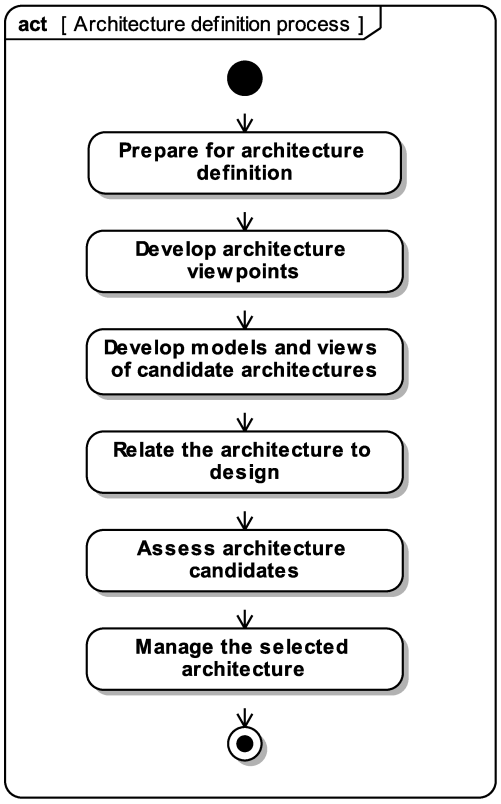


Figure 3.12: ISO/IEC/IEEE 15288:2015 process for architecture definition [21], modeled in a SysML activity diagram

In preparation for architecture definition, information for architecture development, *e.g.* market studies, organizational studies, missions, concept of operations, etc. are reviewed and key drivers for the architecture are identified. Stakeholders and their concerns are collected and evaluation criteria are defined based on stakeholder concerns and requirements. In the next process step, architecture viewpoints are developed based on existing architecture frameworks. These architecture frameworks and elaborated viewpoints are used for modeling candidate architectures in the subsequent process step. Modeling activities include the consideration of interfaces of the system to its environment that are modeled in a system context. Furthermore, architectural entities and their relationships to other architectural entities and system boundaries are identified and modeled. Then, concepts, properties, characteristics, behaviors, functions, and constraints are allocated to architectural entities.

In the subsequent fourth process step, physical system elements are identified and allocated to the architectural entities. Interfaces and interactions between system elements

are described and requirements are traced to system elements. The resulting architecture drafts are evaluated based on constraints, requirements, and evaluation criteria from the first process step. One or more of the evaluated architectures are selected and documented as a baseline by means of models, views, and other descriptions. The last process step is dedicated to architecture management. For this purpose, an architecture governance approach is elaborated and roles for different architecture aspects, *e.g.* design, quality, security, and safety, are determined. Finally, the architecture is presented to stakeholders for obtaining their acceptance and the selected architecture is maintained with respect to technological, implementation, and operational experience [21].

The annex G.3 of ISO/IEC/IEEE standard 15288:2015 provides remarks for application of the standard to SoS. According to the annex, SoS architecture definition begins with a de facto SoS architecture. The de facto architecture describes the status quo of the SoS and its CS. Furthermore, the annex comments on requirements at the SoS and CS levels. The SoS architecture must meet requirements and constraints at the SoS and the CS levels. SoS architecture alternatives are developed and evaluated based on SoS and CS requirements and constraints [21].

The identification of constraints, requirements, and evaluation criteria for the subsequent evaluation of architecture drafts is incorporated into the method CPSoS Mission and Operational Scenarios Definition. Together with the considerations of the SoSACRE from the concept phase (*cf.* Section 3.1.1 on page 41), the de facto architecture is adopted so that a system context and the information network structure describe the status quo of the CPSoS.

ISO/IEC/IEEE 21840:2019: Systems and Software Engineering – Guidelines for the Utilization of ISO/IEC/IEEE 15288 in the Context of System of Systems (SoS)

The ISO/IEC/IEEE standard 21840:2019 adds SoS considerations to the ISO/IEC/IEEE standard 15288:2015 and distinguishes between SoS and CS architectures. The architectural considerations do not encompass functional architectures and emphasize general architectural elements. Accordingly, interfaces are not considered between functions but between general architectural elements. Consequently, the standard does not provide advice for the identification of redundant functions and for systematic architecture derivation from functions. Furthermore, the standard lacks modeling support.

ISO/IEC/IEEE standard 21840:2019 adds SoS considerations to all processes described in ISO/IEC/IEEE standard 15288:2015. The following considerations are provided for the SoS architecture definition process according to ISO/IEC/IEEE standard 15288:2015. SoS stakeholder and CS stakeholder needs should be respected in parallel. It is favorable to design the architecture in a way that changes in stakeholder needs at the SoS or the CS level influence the other development level as little as possible. Therefore, SoS and CS

architecture viewpoints should be added. The standard emphasizes that context, boundaries, and interfaces of an SoS evolve because CS may change, enter, or leave the SoS. Consequently, it is suggested that abstract classes of CS be created that can be allocated to different CS to be able to integrate different CS generations and to make the SoS robust towards CS changes. This evolution results in frequent architecture changes so that the evaluation of SoS architecture alternatives is an important aspect that should also address impacts on CS architectures by the SoS architecture [23].

As described in the presentation of approaches for the concept phase in Section 3.1.1 on page 41, the ISO/IEC/IEEE standard 21840:2019 is part of a set with the standards 21839:2019 [22] and 21841:2019 [24]. These standards address the development of CS and provide a taxonomy for SoS respectively. Neither standard addresses functional SoS architectures and they are therefore not part of this section about approaches for the function and architecture phases.

Functional Architectures for Systems (FAS)

Guidance for functional analysis and functional architecture definition is provided in the method *Functional Architectures for Systems* (FAS) [87]. The FAS method is intended for the development of self-contained systems and requires an adaption for the design of CPSoS architectures, as CPSoS considerations are not part of the conventional FAS method. Interfaces between functions are taken into account, especially in the model-based part of the FAS method by using SysML domain blocks for the specification of interfaces in functional architectures. Redundant functions are identified during functional grouping so that a set of functions without much overlap is obtained. The FAS method provides a detailed process for derivation of a functional architecture from use case activities and functional groups. A plugin for modeling tools supports this process with automation of selected process steps. As the scope of the FAS method is the development of functional architectures, further development of logical architectures is not part of the method. The FAS method is independent of a modeling language and tool. However, advice for utilization of SysML in combination with the FAS method is provided.

The FAS method transfers use cases into a functional architecture and is independent of any modeling language. However, a model-based application of the method is supported and utilizes the SysML. The result of FAS application is a functional architecture that is independent of technical solutions and therefore robust with respect to technological changes and evolution. Developing functional architectures for self-contained systems according to the FAS method is already established and presented in the following. Figure 3.13 shows the associated process for developing and modeling a functional architecture in SysML using the FAS method.

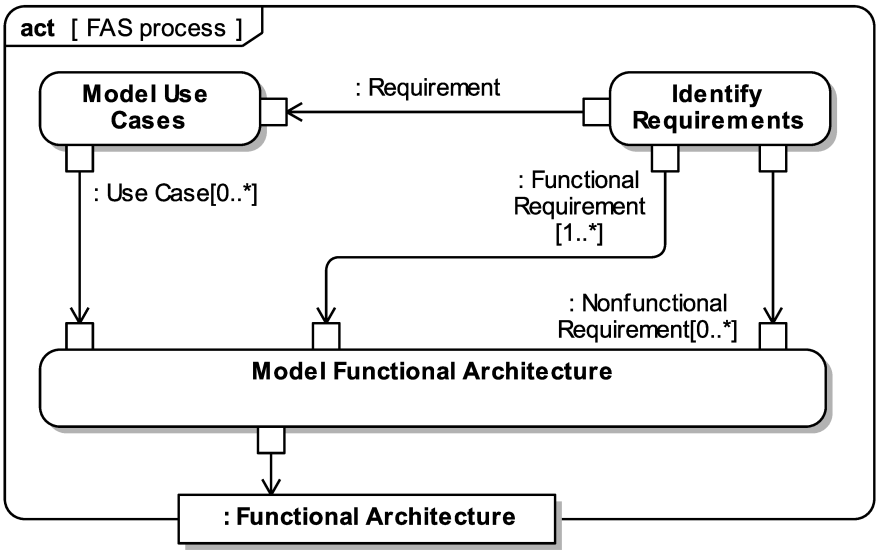


Figure 3.13: FAS process [86], modeled in a SysML activity diagram

The FAS method begins with a description of the system context and identification of system boundaries. System use cases are identified and refined by means of use case activities. These activities are grouped into functional groups that are the basis for subsequent development of functional elements. Functional elements are translated into functional blocks that constitute the functional architecture of the system. The FAS method process steps are described in detail in the following.

The system context according to the FAS method describes the system boundary and interfaces to actors and external systems. It is comparable to the stakeholder context according to Holt, Perry, and Brownsword in [81]. After system context elaboration, use cases of stakeholders are identified. For the FAS method, use cases describe the prerequisite, transition, and result of system functions. According to SysML specification version 1.6 [12], use cases describe “behavior in terms of the high-level functionality and uses of a system, that are further specified in the other behavioral diagrams.” The FAS method utilizes activities and activity diagrams for a detailed description of use cases in the next step. It is important that use case and activity definition remains abstract enough so that the possible solution space for physical implementation is not unnecessarily constrained. Object flows in activity diagrams represent the flow of information, signals, materials, force, and energy between the inputs and outputs of actions and indicate the chronological order of actions.

The previous steps, *i.e.* identification of system context, use cases, and use case activity modeling can be summarized as system analysis. This analysis is followed by system architecture development. The system architecture development starts with grouping of use case activities into functional groups using heuristics. A well-defined grouping ensures that functions are not provided by multiple functional elements and thus avoids unnecessary

development of system elements with similar or identical functionality. Heuristics for the development of sensible functional elements are provided by the FAS method. One of the heuristics takes into account stakeholders. Functions that are allocated to the same stakeholder are probably in the same functional group. Another aspect for similar functions are function calls. A network of functions that call each other or share data is a considerable candidate for a functional group. In addition, grouping criteria of existing groups may be a hint for functional grouping if they are conceptual. Technical criteria for grouping should be avoided to reduce the risk for early solution space constraints.

Functional groups are the basis for creating functional blocks and functional interfaces that represent the functional architecture of a system. Per functional group one functional block with the same name is created. Interfaces, modeled by pins and object flows in use case activities, are translated into ports of functional blocks and connectors between ports. To model the functional architecture, part properties are created that are typified by the functional blocks. These part properties are added to an internal block diagram and the connectors between part properties are displayed to illustrate the interfaces in the functional architecture.

For the automation of some steps of the FAS method, a plugin for the modeling tools Cameo Systems Modeler and MagicDraw supports modeling functional architectures by automated initialization of functional groups as well as automatic creation of functional blocks and interfaces after manual functional grouping.

The FAS method is the basis for the methods CPSoS Function Identification and Architecture Development and CPS Function Identification and Functional Architecture Development. The concept of refining use cases, functional grouping, and deriving a functional architecture from the functional groups is adopted for the needs of CPSoS development and integrated into the methods.

Object-Oriented Systems Engineering Method (OOSEM)

OOSEM provides an approach for logical architecture development without design of a functional architecture. The OOSEM approach does not consider SoS and CS functions. Instead, it is intended for the application in self-contained system development. Functions are modeled as operations of the system block. They are not connected to each other so that a specification of function interfaces is not supported. Nevertheless, connectors in internal block diagrams connect ports of logical components in the logical architecture. The OOSEM approach does not provide specific advice for identifying redundant functions. However, redundant functions can be identified when defining interactions between logical components for each function as indicated by the iteration loop in the process overview in Figure 3.14. The approach suggests manual logical architecture derivation after identifying logical components by sequential consideration of system functions and lacks a systematic process for

architecture derivation. The whole approach is intended for model-based application and provides processes in the form of activity diagrams.

OOSEM starts with modeling a system block. Desired system behavior is described in scenarios by means of SysML actions and modeled as operations of this system block. As a result, the system block contains a set of operations that represent the desired system functions. After identification of all actions, relations between actions are modeled in a SysML state machine diagram. The state machine diagram describes the possible order of system operations and permitted transitions between system operations. Afterwards, a logical architecture is developed for decomposing the system into logical components according to the process depicted in Figure 3.14.

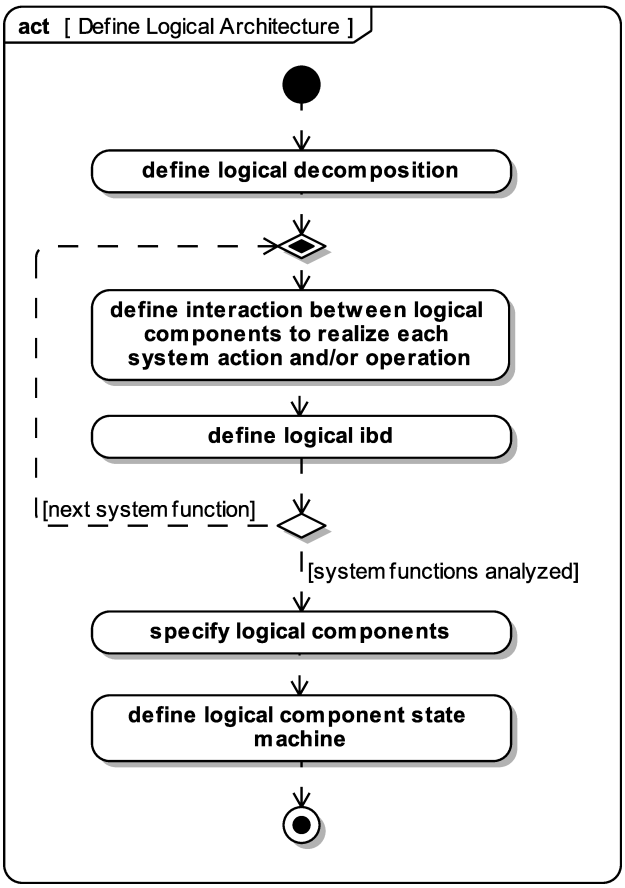


Figure 3.14: Logical architecture definition according to OOSEM [69], modeled in a SysML activity diagram

Logical components are abstractions of physical components and describe what functionality the physical components of the system should provide. The OOSEM approach skips a functional architecture as it is presented in the FAS method. Instead, a logical architecture is developed immediately after identification of system functions. While the FAS method groups similar use case activities into functional groups, OOSEM identifies system functions, defines logical components, and allocates actions to logical components. This allocation is

modeled in SysML activity diagrams by means of activity partitions for each system function as indicated by the loop in Figure 3.14. Object flows between actions from different activity partitions represent interactions between logical components.

Interfaces between logical components are modeled in an internal block diagram based on a manual analysis of the modeled system operations. The diagram shows part properties with the type of logical components that are connected to each other by means of ports and connectors. As soon as all system functions are analyzed, the logical components are specified in more detail by allocating actions to logical components. In conclusion, the OOSEM approach works with two levels for modeling system functions and logical architectures, *i.e.* system level and logical component level. Functions are modeled as operations at the system level. An activity diagram describes each function in more detail. The actions in these activity diagrams are modeled as operations of the logical components [69].

The OOSEM is not adopted and integrated into the methods proposed in Chapter 4. However, the concept of using part properties, ports, and connectors in SysML internal block diagrams for modeling a system architecture is also part of the FAS method and confirms the approach chosen in the methods CPSoS Function Identification and Architecture Development and CPS Function Identification and Functional Architecture Development.

Integrated Systems Engineering and Pipelines of Processes in Object-Oriented Architectures methodology (ISE&PPOOA)

Fernandez and Hernandez present the model-based Integrated Systems Engineering and Pipelines of Processes in Object-Oriented Architectures (ISE&PPOOA) approach for Systems Engineering and Software Architecture Development in [67]. It focuses on system functions, does not consider SoS functions and recommends analysis of functional interfaces in N2 matrices. Unfortunately, SysML does not provide a suitable diagram for these matrices. Consequently, an approach-specific adaption of modeling tools is required if the interface information from the N2 matrices should be traceable to other model elements. Furthermore, the approach provides no guidance for the identification of redundant or missing functions so that a complete analysis of CPSoS functions is not possible. The design of extensive functional CPSoS architectures requires a systematic approach that organizes the derivation of a functional architecture from identified functions. In the ISE&PPOOA approach, the functional architecture must be derived manually by modeling functional blocks in the functional decomposition and functional actions in activity diagrams for modeling the functional flow. Moreover, functional blocks and functional activities are not traceable and a logical layer is missing so that functions are directly allocated to physical components.

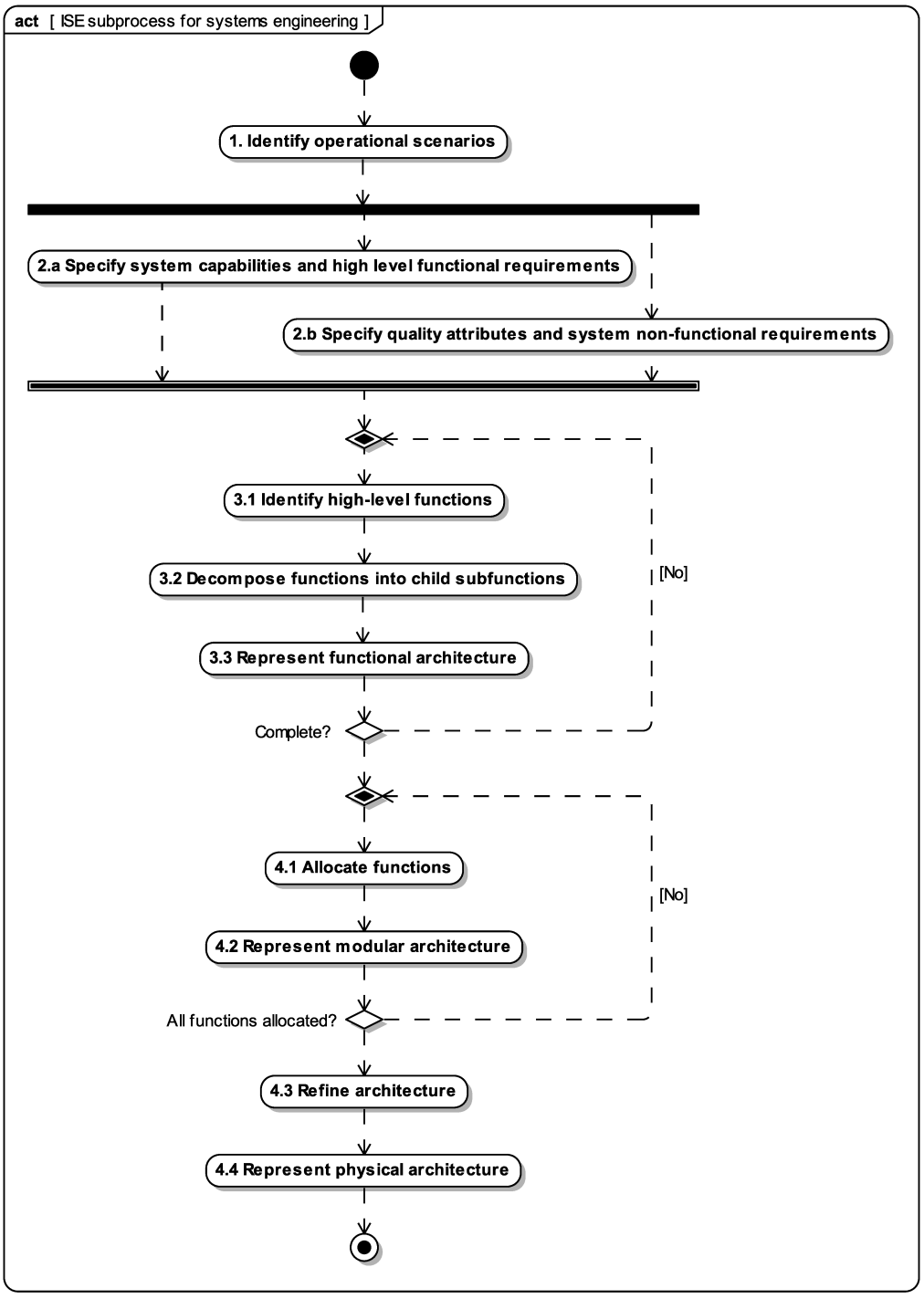


Figure 3.15: ISE&PPOOA process for architecture development [67], modeled in a SysML activity diagram

Operational scenarios describe desired system behavior and are identified in the first process step. They are the basis for the subsequent specification of system capabilities, high-level functional requirements, quality attributes, and non-functional requirements. System capabilities are expressed as hierarchical decomposition in a block definition diagram.

High-level functional requirements describe system requirements in natural language in more detail. In parallel, non-functional requirements are specified. They describe quality attributes concerning, *e.g.* security or reliability. In the next step 3.1 of the ISE&PPOOA methodology, high-level system functions are identified and modeled in a block definition diagram. Fernandez and Hernandez suggest the analysis of similar systems as well as inputs and outputs of the system for the identification of functions. Each high-level function is then decomposed into subfunctions in step 3.2. The corresponding function granularity depends on the desired granularity of the physical system components. In step 3.3, a functional architecture is designed. A functional architecture according to the ISE&PPOOA approach is a functional decomposition presented in a block definition diagram. It is complemented by an activity diagram for functional flow description, textual function description, and N2 charts for illustrating functional interfaces.

As soon as the set of identified system functions is complete, functions are allocated to physical components in step 4.1. The ISE&PPOOA approach aims at physical elements that are as independent as possible for a modular architecture. For this purpose, physical components and their allocated functions should have strong inner coupling and only few interfaces to other physical components and functions respectively. The physical components and their interfaces are represented in block definition diagrams for hierarchical representation and internal block diagrams for representing connections between physical components. Quality attributes and non-functional requirements from step 2.b are considered for refining the physical architecture in step 4.3. Trade studies can be performed for selecting the most suitable physical architecture draft. The hierarchy of the final physical architecture hierarchy is modeled in a block definition diagram while the interfaces between subsystems and physical components are presented in an internal block diagram. The related system behavior is described in activity and state machine diagrams.

Most parts of the ISE&PPOOA approach are not adopted for the function and architecture phases of CPSoS. However, the utilization of operational scenarios for the description of desired system behavior and the identification of quality attributes and non-functional requirements are integrated into the method CPSoS Mission and Operational Scenarios Definition.

Unified Architecture Framework (UAF)

As already introduced in Section 3.1.1 on page 53, UAF is intended for modeling SoS architectures, especially in the context of enterprises. UAF is well suited for the consideration of CS and SoS functions. Functions can be described at the SoS level and are then allocated to CS. SysML as basis for UAF allows function description at the CS level using interfaces between UAF and SysML. In UAF, interfaces between functions are described by means of *DataElements* that represent the type of *FunctionAction* pins. However,

UAF lacks a viewpoint for the identification of redundant functions. Nevertheless, functional decomposition in block definition diagrams of the *Resources Processes* viewpoint can be used for analysis with respect to redundant functions. Viewpoints for systematic architecture derivation from the function set and architecture evaluation are also not part of UAF.

Functions are considered in the *Resources Processes* viewpoint of UAF by using diagrams that are based on SysML block definition diagrams and activity diagrams. Block definition diagrams describe functional hierarchies of activities with the stereotype *Function* that are connected by aggregation relationships. In addition, *CapabilityConfigurations* and *Posts* are connected to functions by allocation relationships with the stereotype *IsCapableToPerform*. *CapabilityConfigurations* are composite structures of physical and human resources that provide a capability. *Posts* describe job titles or positions by persons, such as lawyer, solution architect, and so on. The interfaces between functions and the flow of information is modeled in activity diagrams. Functions are not directly integrated into the activity diagrams. Instead, *FunctionActions* are modeled in the activity diagrams and call *Functions*, analogous to the call behavior action and activity mechanism in SysML. *FunctionActions* are connected to each other by means of object flows between pins. So-called *DataElements* are used for specification of pin types. *DataElements* represent “data that is managed or exchanged between systems.” [19] *FunctionActions* are allocated to partitions that represent *CapabilityConfigurations* and *Posts* [19].

3.2.2 Evaluation of Approaches for the Function and Architecture Phases

Corresponding to the evaluation of methods for the concept phase in Section 3.1.2 on page 55, this section evaluates approaches for the function and architecture phases on the basis of evaluation criteria that are derived from SoS characteristics (*cf.* Section 2.1 on page 15). Table 3.2 summarizes the evaluation results.

Consideration of CPSoS and CPS functions: Often, CPSs exist before their integration into a CPSoS and they have individual CPS functions for their initial purpose. When these existing CPSs are integrated into a CPSoS, some of the CPS functions can be used for the CPSoS mission and some functions are added. An approach for functional analysis and architecture design in CPSoS should consider both function types, initial CPS functions and additional CPSoS functions. Only the UAF allows the complete consideration of both function types.

Consideration of interfaces between functions: Interfaces between functions allow the CPSoS-characteristic intensive exchange of data and information. Consequently, approaches for the function and architecture phases should consider interfaces between

Approaches for the Function and Architecture Phases	Evaluation Criteria					
	Consideration of CPSoS and CPS functions	Consideration of interfaces between functions	Identification of redundant functions	Architecture derivation	Model-based	Provision of a process
ISO/IEC/IEEE 15288:2015 Systems and Software Engineering - System Life Cycle Processes	◔	◔	○	○	○	●
ISO/IEC/IEEE 21840:2019 Systems and Software Engineering - Guidelines for the Utilization of ISO/IEC/IEEE 15288 in the Context of System of Systems (SoS)	◔	◔	○	○	○	◑
Functional Architectures for Systems (FAS)	○	●	●	●	●	●
Object-Oriented Systems Engineering Method (OOSEM)	○	◑	◔	◑	●	●
Integrated Systems Engineering and Pipelines of Processes in Object-Oriented Architectures methodology (ISE&PPOOA)	○	◑	◔	◔	●	●
Unified Architecture Framework (UAF)	●	●	◑	○	●	○

● fulfilled ◕ mostly fulfilled ◑ partially fulfilled ◔ fulfilled to a lesser extent ○ not fulfilled

Table 3.2: Evaluation of methods with respect to architecture development in CPSoS

functions. Only the FAS method and the UAF fully support the design and modeling of functional interfaces.

Identification of redundant functions: The extent and complexity of CPSoS results in a large amount of functions. Each function causes effort in the architecture design and implementation so that function duplicates should be avoided. Only the FAS method enables sufficient identification of redundant functions.

Architecture derivation: After identification of required functions, the approaches should support the systematic derivation of an architecture. This architecture should enable the allocation of functions to CPS that are part of the CPSoS, which is only completely supported by the FAS method.

Model-based: As in the concept phase, the extent and complexity of CPSoS call for a model-based approach for consistency and traceability. All approaches apart from the standards offer a model-based approach.

Provision of a process: Corresponding to the concept phase, all approaches except the UAF offer a process.

None of the methods fulfills all desired criteria. However, method aspects can be combined into a solution meeting the third research question “How are functional and logical CPSoS architectures designed to enable new service types?” (*cf.* Section 2.3 on page 34). The

functional grouping and deriving functional blocks concept of the FAS method is an example of suitable method aspects.

3.3 Design Phase

The architecture life cycle phase is followed by the design phase [6]. Since the literature does not offer any approaches for the design phase of SoS or CPSoS, this section presents approaches for the design phase of self-contained systems and evaluates them with respect to their suitability for application to CPSoS.

Summary - Design Phase

- A logical architecture comprises logical elements that are abstractions of physical system elements. In the CPSoS domain, logical elements represent CPSs [25].
- Functions are allocated to logical elements.
- Since there are multiple variants for function allocation to logical elements, there is a need for assistance in selecting the most suitable architecture draft.
- Model-based evaluation of logical architecture drafts can support the identification of the most suitable draft.

Room for improvement in CPSoS development

- None of the analyzed approaches considers the allocation of functions to CPSs in a CPSoS.
- Existing approaches are limited with respect to interface specifications in a logical architecture.

3.3.1 Approaches for the Design Phase

Figure 3.16 and Figure 3.17 illustrate the allocation of a functional element to different CPSs in alternative logical CPSoS architecture drafts. In the first architecture draft in Figure 3.16, the functional element X (gray fill color) is allocated to the CPS 1 while in the second draft in Figure 3.17, it is allocated to the CPS 2. Approaches for the design phase of CPSoS should allow the systematic design of alternative logical CPSoS architectures and the selection of the most suitable proposal.

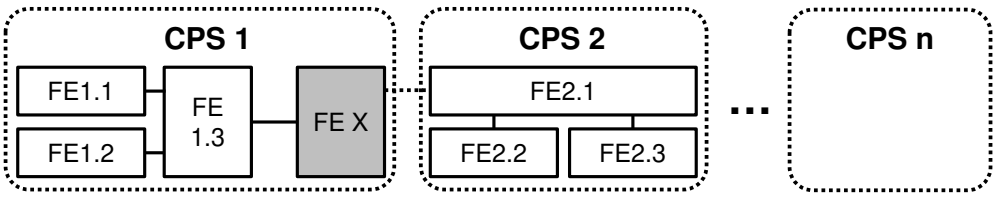


Figure 3.16: Allocation of functional elements to CPSs in a logical CPSoS architecture draft

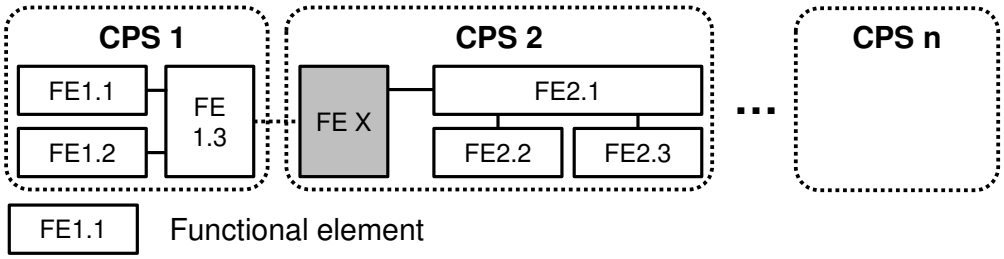


Figure 3.17: Allocation of a functional element to CPSs in an alternative logical CPSoS architecture draft

The following sections present approaches that can be adapted to the design phase of CPSoS. Since all approaches are intended for the design of self-contained systems, the room for improvement with respect to CPSoS design is elaborated.

Systems Modeling Toolbox (SYSMOD)

Designing and modeling a logical architecture for self-contained systems is also part of SYSMOD. However, the method is not intended for the allocation of CPSoS functions to CPSs so that the exchange of data and information in a CPSoS is not completely specifiable. In addition, the method does not provide any advice for the evaluation of logical architecture drafts. Nevertheless, some parts of the methodology were considered for the development of the methodology proposed in this thesis. The associated SYSMOD process for logical architecture design is presented in Figure 3.18.

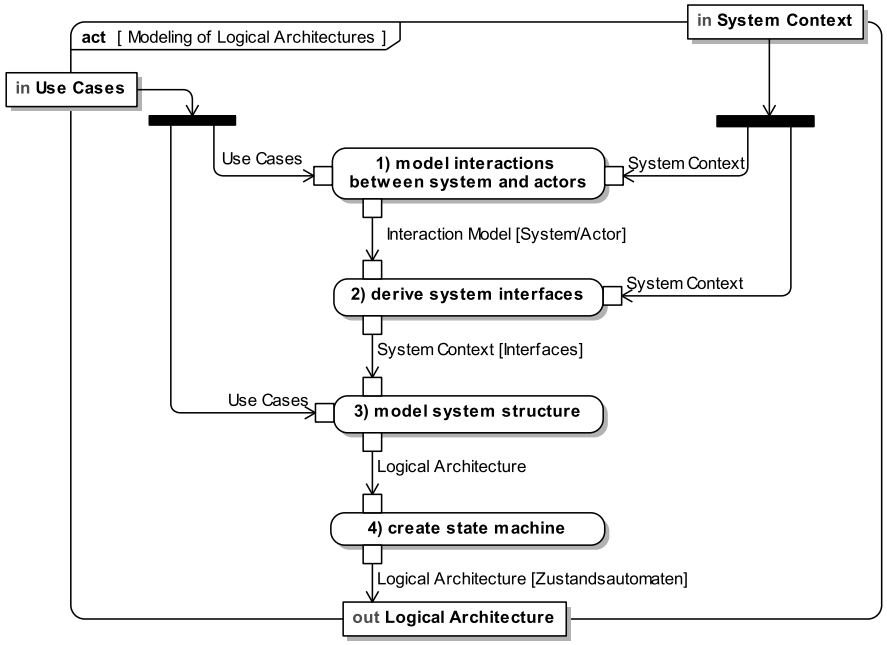


Figure 3.18: Modeling logical architectures according to SYMOD [110], modeled in a SysML activity diagram

First, expected interactions between the system and actors are modeled in a SysML sequence diagram for each use case of the system. In the second step, system interfaces are derived for these interactions and modeled as proxy ports of the system. Afterwards, the system structure is modeled as logical architecture in the third step. SysML block definition diagrams are used for modeling the system hierarchy, and internal block diagrams represent the internal system structure. Finally, the behavior of the system parts is described in the fourth step. For this purpose, actions are allocated to the logical elements in a SysML activity diagram, as shown in Figure 3.19. This activity diagram illustrates the allocation of actions to logical system elements and the exchange of data and information between the actions. Vertical partitions structure the activity diagram and represent the logical components of the system. Within the partitions, actions are entered that are executed by the respective logical component.

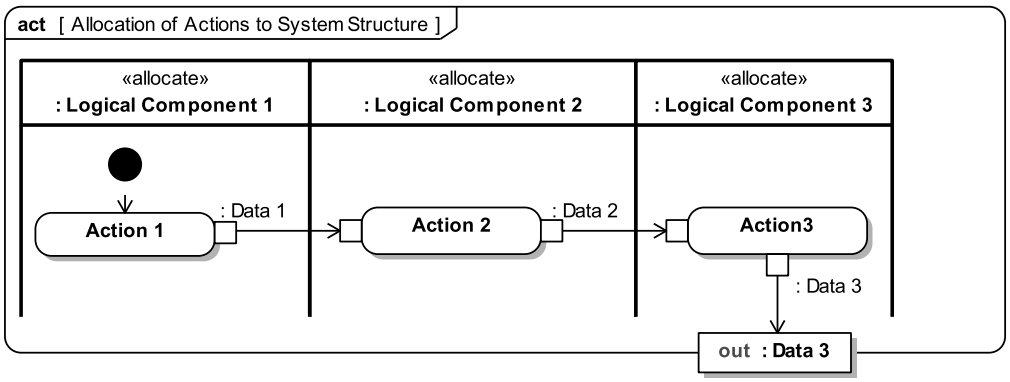


Figure 3.19: Allocation of actions to logical elements according to SYSMOD [110], modeled in a SysML activity diagram

Although SYSMOD does not consider CPSs and CPSoS, the definition for logical architectures could be adopted in the method CPSoS Function Identification and Architecture Development. Furthermore, the ideas of modeling a logical architecture by means of blocks in block definition diagrams and internal block diagrams as well as allocating functions to logical elements is integrated in the same method. While SYSMOD proposes to model functions as actions that are allocated to logical blocks, the method CPSoS Function Identification and Architecture Development uses functional blocks for modeling a functional architecture and allocating functional to logical blocks.

Object-Oriented Systems Engineering Method (OOSEM)

Part of the Object-Oriented Systems Engineering Method (OOSEM) by Friedenthal et al. [69] is introduced in Section 3.2.1 on page 63. This section presents and evaluates an additional part of OOSEM for physical architecture design and architecture evaluation. OOSEM has not been elaborated for the development of CPSoS. However, it can be adapted well for use with CPSoS. It does not allow the allocation of functions to CPSs, but the allocation of actions

to physical components of the system under development. It is notable that OOSEM uses individual architecture definitions. While a functional architecture is not considered, OOSEM distinguishes between logical components and logical nodes. Logical components are solution-neutral abstractions of physical components and constitute the logical architecture, as introduced in Section 3.2.1. Logical nodes are an aggregation of logical components at a specific location and thus allow the distribution of logical components and their functions in distributed systems. Figure 3.20 shows a segment of OOSEM for the development of a logical node architecture and physical architecture alternatives.

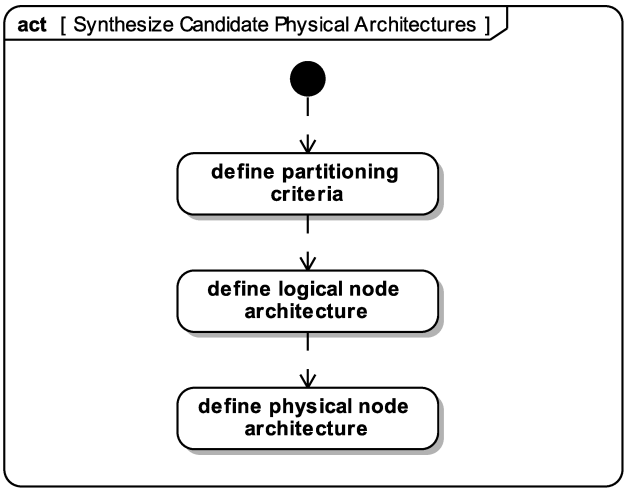


Figure 3.20: Segment of OOSEM for the development of physical architecture alternatives [69], modeled in a SysML activity diagram

The OOSEM process for the development of physical architecture alternatives begins with the definition of partitioning criteria. These partitioning criteria are based on technical and life cycle considerations. Technical considerations encompass, *e.g.* the allocation of shared functions to shared components while partitioning of components based on maintainability and update rates is an example of application of life cycle considerations. In the next step, logical components of the logical architecture (*cf.* Section 3.2.1 on page 63) are grouped and allocated to logical nodes. Both, logical components and logical nodes, are modeled as SysML blocks and are connected to each other using a composition. For transition to a physical architecture, logical components at each logical node are mapped to physical components at each node. OOSEM distinguishes between physical hardware and physical software components that are allocated to logical components in dependency matrices [69].

In addition to the architecture definition process, OOSEM provides a process for optimization and evaluation of alternatives that can be applied at various process steps for analyses and trade studies. The process is structured into four activities, as shown in Figure 3.21, and can be applied to the evaluation and selection of architecture drafts. This process is widely adopted in the method CPSoS Architecture Evaluation so that a more

detailed explanation of its application in the CPSoS context is provided in Section 4.3 on page 103.

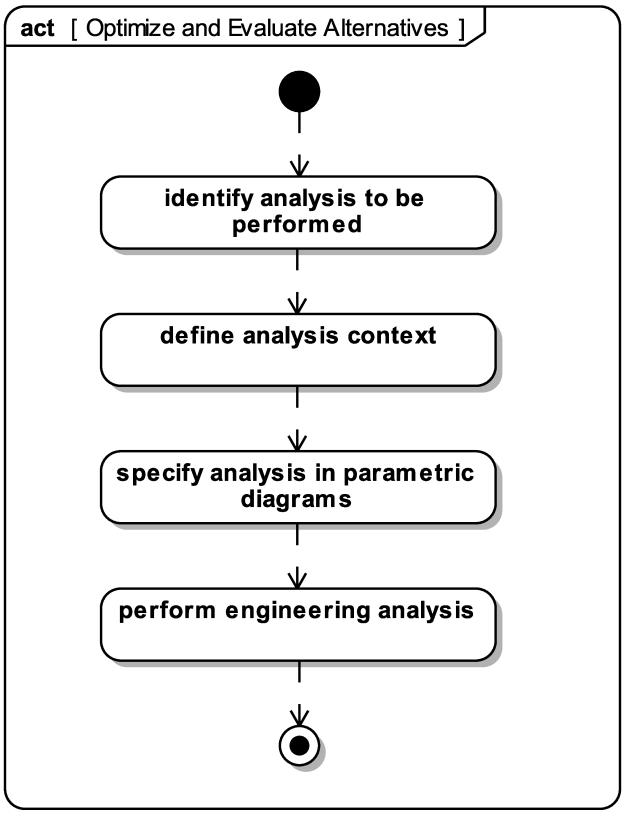


Figure 3.21: Evaluation of architecture alternatives [69], modeled in a SysML activity diagram

First, the analysis to be performed is identified. Analysis objectives are determined and include the estimation of certain system aspects, *e.g.* cost, performance, reliability, or mass. Then, the identified analysis objectives are modeled in an analysis context. Each analysis to be performed is modeled as a block that is connected to an analysis context block via a composition. Next, objective functions are defined and modeled as constraints in the analyses blocks. The constraints contain an equation and associated parameters. The analysis specification is connected to system block properties in parametric diagrams in the third process step. The results of execution of these parametric diagrams during the fourth process step can be incorporated into the system model and can be compared with constraints or evaluation results of alternative solutions [69]. The OOSEM approach for the evaluation of architecture alternatives is adopted in the method CPSoS Architecture Evaluation.

Engineering Design: A Systematic Approach

Feldhusen et al. [66] present an approach for the development of mechanical systems. Their approach emphasizes a functional description of the product to recognize significant

problems and functions of the system, to identify a possible structure of the system, and to discuss components of a modular product structure with sales representatives and the product management. Therefore, two viewpoints are introduced, *i.e.* the input-output-viewpoint and the hierarchical viewpoint. The input-output-viewpoint is a chain of functions and shows either a process sequence or a flow of material, energy, or information. The hierarchical viewpoint presents functions and their dependencies by showing independent functions that are divided into subfunctions. Functions, their subfunctions, and possible subfunctions of the subfunctions form a tree representation of the system functions that shows dependencies between the functions and their subfunctions instead of a sequence of events. The corresponding development process is shown in Figure 3.22.

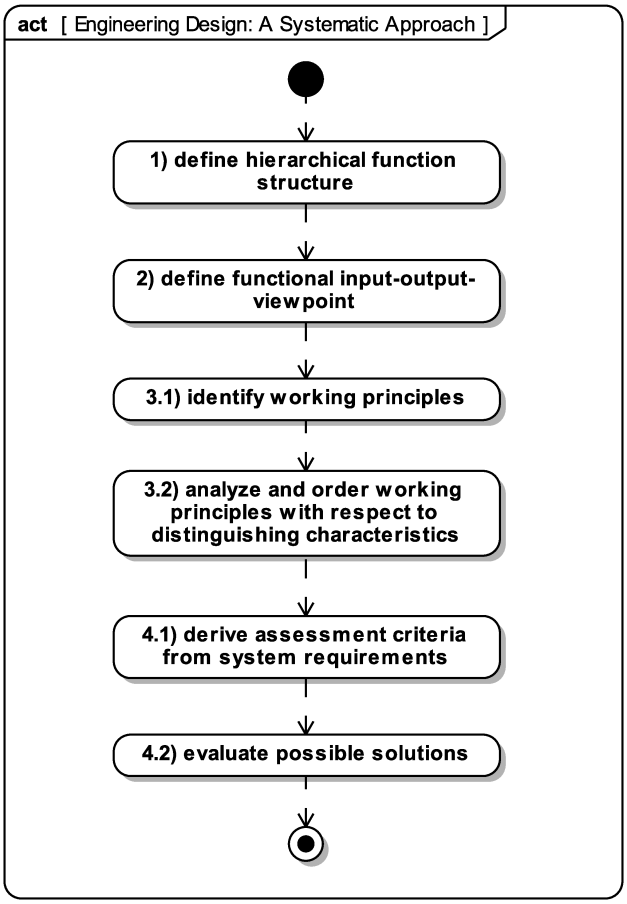


Figure 3.22: Derivation and evaluation of architecture alternatives [66], modeled in a SysML activity diagram

Feldhusen et al. suggest developing the hierarchical function structure in the first step in order to gather required functions, their subfunctions, and dependencies between them. In step 2, connections between the subfunctions can be determined for identifying logical and physical dependencies between the subfunctions while keeping a solution-neutral abstraction level [66].

Step 3.1 includes the development of working principles that provide the previously defined functions. Feldhusen et al. distinguish between four method groups for identification of working principles, *i.e.* analogy review, intuitively emphasized methods, theory of inventive problem solving, and discursive emphasized methods. Identified working principles are analyzed and systematically ordered by means of order schemes in step 3.2. Matrices support the ordering and contain distinguishing characteristics, *e.g.* energy type, geometry, movement type, etc. of the working principles in the columns. The collected working principles are entered into the rows below the ordering column headings. This approach supports the identification of possible conflicts between working principles for the functions and encourages the search for further working principles. Further working principles can be identified by variation of distinguishing characteristics. An example is the movement type. If a working principle is based on translational movement, consideration of rotational movement may lead to another working principle [66].

The combination of working principles results in several possible solutions for the system design. An evaluation of the possible solutions can support the selection of the most suitable combination of working principles. For the evaluation, evaluation criteria are derived from system requirements in step 4.1. Subsequently, these criteria are converted into measurable quantities and each possible solution is evaluated on the basis of the measurable quantities. Usually, the measurable quantities have different importance for the overall evaluation of a possible solution. The importance of measurable quantities is respected by means of a weighting of measurable quantities. Therefore, measurable quantities are multiplied by a weighting factor thus resulting in a so-called rating number. The overall assessment result of a particular solution is obtained by addition of the rating numbers in the evaluation of possible solutions in step 4.2 [66].

Although the Engineering Design approach is used for mechanical systems and does not consider CPSs or CPSoS, certain aspects can be used for their design. First, the derivation of evaluation criteria from system requirements is also part of OOSEM and confirms this approach. Second, Feldhusen et al. introduce weighting factors that complement the approach suggested in OOSEM. These weighting factors are integrated into the method CPSoS Architecture Evaluation.

3.3.2 Evaluation of Approaches for the Design Phase

Consistent with the evaluation for the concept phase in Section 3.1.1 and the function and architecture phases in Section 3.2.2, this section evaluates approaches for the design phase of CPSoS in Table 3.3.

Allocation of functions to CPSs: For the design of CPSoS, it should be possible to allocate functions to the CPSs contained. SYSMOD allows the allocation of actions to the structure

Approaches for the Design Phase	Evaluation Criteria				
	Allocation of functions to CPSs	Specification of interfaces between CPSs	Architecture evaluation	Model-based	Provision of a process
Systems Modeling Toolbox (SYSMOD)	◑	◕	○	●	●
Object-Oriented Systems Engineering Method (OOSEM)	◕	◑	●	●	●
Engineering Design: A Systematic Approach by Pahl, Beitz et al.	○	○	◕	○	●

● fulfilled ◕ mostly fulfilled ◑ partially fulfilled ◔ fulfilled to a lesser extent ○ not fulfilled

Table 3.3: Evaluation of methods with respect to the design phase of CPSoS

of a self-contained system but does not consider CPSoS. As an alternative, OOSEM groups logical components to logical nodes at the system level.

Specification of interfaces between CPSs: The intensive communication between CPSs in a CPSoS calls for the specification of interfaces between CPSs in the design phase of CPSoS. The consideration of interfaces by SYSMOD and OOSEM can de adapted for the CPSoS level.

Architecture evaluation: There are many alternatives in CPSoS to distribute functions across CPSs, resulting in several different architecture drafts. An approach for the design phase should support the evaluation and selection of the most suitable architecture draft. OOSEM suggests an evaluation approach that can be adopted for CPSoS.

Model-based: In order to link to the results from the concept, function, and architecture phases contained in the model and to promote clarity in the complex CPSoS context, an approach for the design phase should also be model-based. SYSMOD and OOSEM are completely model-based but not intended for the application to the CPSoS level.

Provision of a process: Corresponding to the previous phases, all approaches offer a process.

As in the evaluation for the previous life cycle phases of CPSoS, none of the approaches fulfills all desired criteria. However, parts of the approaches can be modified and combined, *e.g.* the evaluation of architecture alternatives proposed by OOSEM.

3.4 Implementation Phase

Figure 1.2 in Section 1.1 illustrates that the air transport system is evolving into a CPSoS. The key features of CPSoS are the intensive and long-range communication links, as shown in Figure 3.23, that constitute the CPSoS communication architecture. These communication links allow the exchange of data and information between CPSs that are owned by different operators. Usually, these operators share only a subset of their valuable information with other operators. Consequently, the extent of exchanged information is agreed in contracts so that a so-called contract-based communication between operated CPSs emerges [93].

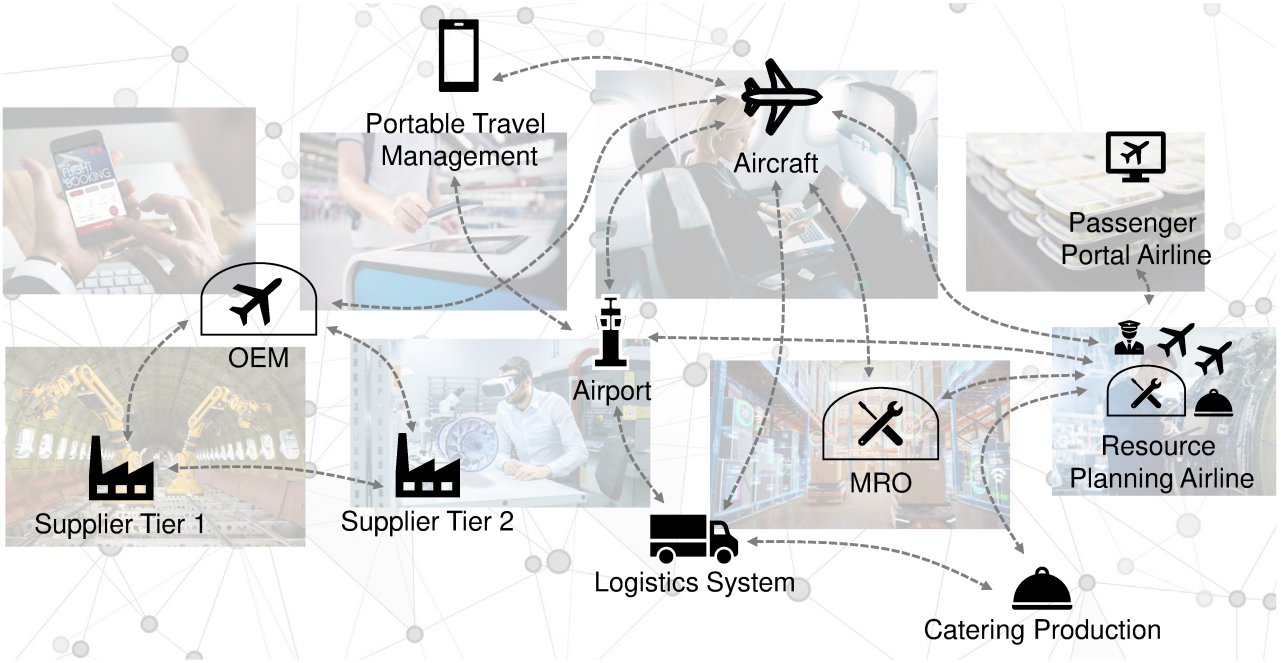


Figure 3.23: Communication paths in a CPSoS

There are several options for implementing a CPSoS communication architecture. Two popular options are the communication via a Representational State Transfer Application Programming Interface (REST API) and the utilization of Message Brokers. Message brokers tend to be more suitable, as their extensive configuration options support the establishment of contract-based communication. REST APIs offer fewer configuration options and only provide synchronous communication between servers and clients so that temporary interruptions in communication are problematic. Both options are presented in the following.

REST determines and standardizes architecture principles so that all REST-compatible systems and applications can communicate with each other. The generic structure of REST communication is presented in Figure 3.24 and includes a client as well as a server with a REST API and a database.

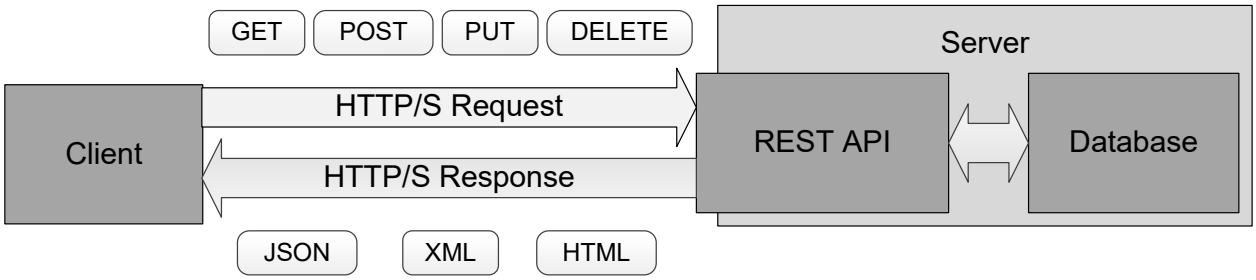


Figure 3.24: REST API structure [97]

The communication between clients and servers is implemented using requests and responses according to the Hypertext Transfer Protocol (HTTP) or Hypertext Transfer Protocol Secure (HTTPS) for security-relevant communication (*cf.* arrows between client and REST API in Figure 3.24). The HTTP/S methods GET, POST, PUT, and DELETE are used for the request by clients.

- GET requests data from a server.
- POST transmits data to a server.
- PUT modifies existing data on the server.
- DELETE deletes existing data on the server.

Exchanged data formats are JavaScript Object Notation (JSON), Extensible Markup Language (XML), and Hypertext Markup Language (HTML) [97]. The REST architecture principles allow the communication between multiple independent systems. However, REST APIs leave room for improvement for the implementation of contract-based communication in CPSoS. Although the access to data could be defined by means of roles and authentication mechanisms, the configuration options of message brokers offer more flexible solutions for the implementation of contract-based communication in CPSoS.

Common protocols for message broker communication are the *Message Queuing Telemetry Transport* (MQTT) protocol [11] for lightweight communication architectures and the *Advanced Message Queuing Protocol* [9] for more complicated communication architectures with multiple configuration options.

MQTT was developed and submitted to the *Organization for the Advancement of Structured Information Standards* (OASIS) by the enterprise IBM and became an ISO standard [82]. MQTT is often used in lightweight applications, *e.g.* in embedded systems applications, because of its low hardware requirements that result from the simple concept as well as short code and message lengths. The MQTT concept considers so-called clients that send and receive messages. Messages are sent to topics on the message broker that store the messages temporarily. These topics are described by a UTF-8 string and can exist in a multi-level structure. As displayed in the example in Sourcecode 3.1, *system1*, *system2*,

status, *humidity*, *temperature*, and *temperatureSensor1* represent different topic levels that are separated by a forward slash.

```
system1/status
system1/status/humidity
system1/status/temperature
system1/status/temperature/temperatureSensor1
system2/status
```

Sourcecode 3.1: MQTT topic description

In contrast to other message broker protocols, topics are created as soon as a client publishes a message to a topic that has not been created before. The broker then distributes the message to all clients subscribed to the corresponding topic. Clients can subscribe to multiple topics using numerous single-level subscriptions or multi-level subscription via topic wildcards. In single-level subscription, clients subscribe to all topics with the same name in different topic branches. If the temperature topic in the example in Code 3.1 is chosen for single-level subscription, the client subscribes to all temperature topics in all branches. In multi-level subscription, clients subscribe to all topic branches that are behind the topic in front of the wildcard. In the example in Code 3.1 setting the multi-level wildcard after the status topic would result in the subscription of all status topics of all branches that contain the status topic. In addition, multiple clients can subscribe to one topic resulting in many-to-many relationships between topics and clients that are characteristic for message protocols using the publish-subscribe pattern [11].

While MQTT represents a lean message protocol with a limited amount of messaging patterns, the *Advanced Message Queuing Protocol* (AMQP) offers more advanced functions for the implementation of contract-based communication in CPSoS. In particular, the broker federation option of the AMQP-based RabbitMQ broker offers extensive configuration options for the communication between CPSs in a CPSoS. Before presentation of the broker federation concept, the basis of AMQP is described in the following. The *AMQ Protocol* specification defines a wire-level protocol that is required for the communication between clients and servers. The *AMQ Model* specifies semantics for server-side components, *i.e.* exchanges, message queues, and bindings that are depicted in Figure 3.25.

As indicated by arrows in Figure 3.25, publishers publish messages to server-side exchanges. These exchanges route messages to queues depending on previously defined bindings. Bindings represent the relationship between exchanges and queues. Queues store the received messages that can be consumed by one (*cf.* Queue 2 in the example in Figure 3.25) or multiple consumers (*cf.* Queue 1). The choice of different exchange types and message queue settings enables the design and implementation of communication architectures between systems that meet individual communication requirements. The AMQ Model

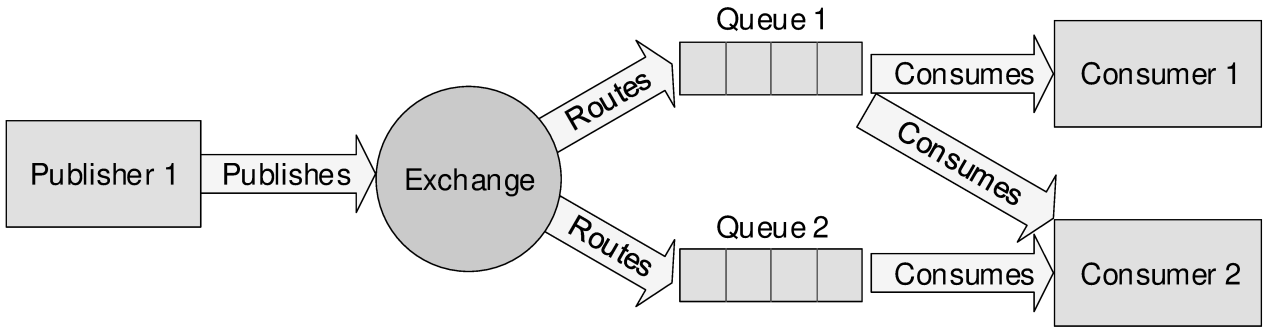


Figure 3.25: AMQ Model components

provides four different exchange types, *i.e. direct*, *fanout*, *topic*, and *headers* exchange. Direct exchange is the default exchange type and enables the routing of messages to predefined queues depending on a routing key, similar to an address in a postal system. Fanout exchange routes messages to all queues that are connected to the exchange. Topic exchange uses wildcards in routing keys to determine the routing of messages to one or multiple queues. Headers exchange is similar to topic exchange and routes messages depending on header values instead of routing keys [9].

As already mentioned above, the broker federation plugin [17] of the AMQP broker RabbitMQ offers more advanced configuration options for the implementation of a contract-based CPSoS communication architecture. Broker federation allows the exchange of a predefined set of information between multiple message brokers. Fundamental parts of broker federation with RabbitMQ are upstreams and policies, as presented in Figure 3.26. Upstreams define how a broker connects to another broker using the broker address and, if required, authentication details. For structuring the upstreams of a broker, upstreams can be grouped in upstream sets. Figure 3.26 illustrates an upstream (light gray) between an upstream broker and a downstream broker.

The flow of messages is unidirectional from the upstream broker to the downstream broker. As soon as an upstream is established, policies can be created. Policies determine a pair of exchanges or queues from different brokers that are connected to each other. Policy 1 in Figure 3.26 connects two exchanges and Policy 2 connects two queues. The function of exchange and queue federation differs. Exchange federation duplicates all messages from the upstream exchange and delivers them to the downstream exchange. In the example in Figure 3.26, all messages that are sent to Exchange A are provided to queues that are connected to Exchange A and to Exchange B. The flow of messages between Queue A and Queue B illustrates queue federation. In queue federation, messages are distributed but not duplicated. Depending on priority settings, messages are sent to consumers that subscribe to the upstream broker queue first. If the upstream broker queue (Queue A in Figure 3.26) does not have any consumers, the messages are sent to federated queues in downstream brokers (Queue B in Figure 3.26) [17].

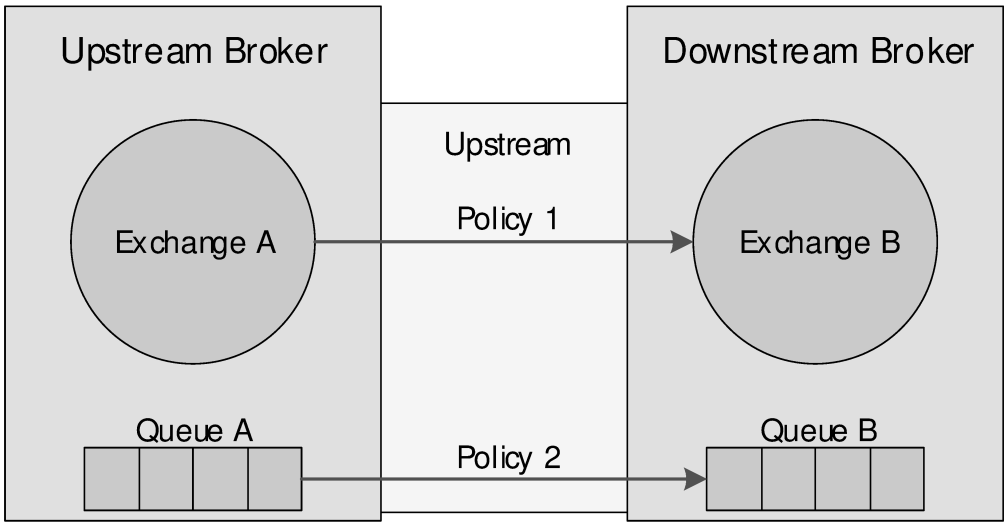


Figure 3.26: Broker Federation

The previous explanations show that different possibilities with individual strengths for the implementation of a CPSoS communication architecture exist. While REST offers a standardization for communication between systems, the configuration options tend to be insufficient for the establishment of contract-based data and information exchange. Message brokers in combination with message broker protocols such as MQTT and AMQP enable a more configurable exchange of data and information and the temporary buffering of messages in case of short-term unavailability of CPSs. The broker federation concept in particular represents a promising approach for the contract-based communication between CPSs that are owned by different operators.

3.5 Further Phases

The system development life cycle not only includes the development phases concept, function, architecture, design, and implementation. After development, the system is produced, operated, maintained, and retired in the life cycle phases production, operation, support, and retirement [6, 21, 31]. More information about life cycle phases subsequent to the group of development life cycle phases and related processes can be found in the ISO standard 15288 [21] and the life cycle phases for commercial aircraft programs by Altfeld [31]. To provide an overview, Figure 3.27 compares the life cycle phases proposed by the ARP 4754A, the ISO/IEC/IEEE standard 15288, and the life cycle phases for commercial aircraft programs by Altfeld.

The ISO/IEC/IEEE standard 15288 also considers the life cycle phases concept and development and additionally includes processes for the operation, maintenance, and disposal of the system [21]. Operation processes encompass the training and qualification of operators, the monitoring of the system, and the provision of support to the customer. Maintenance

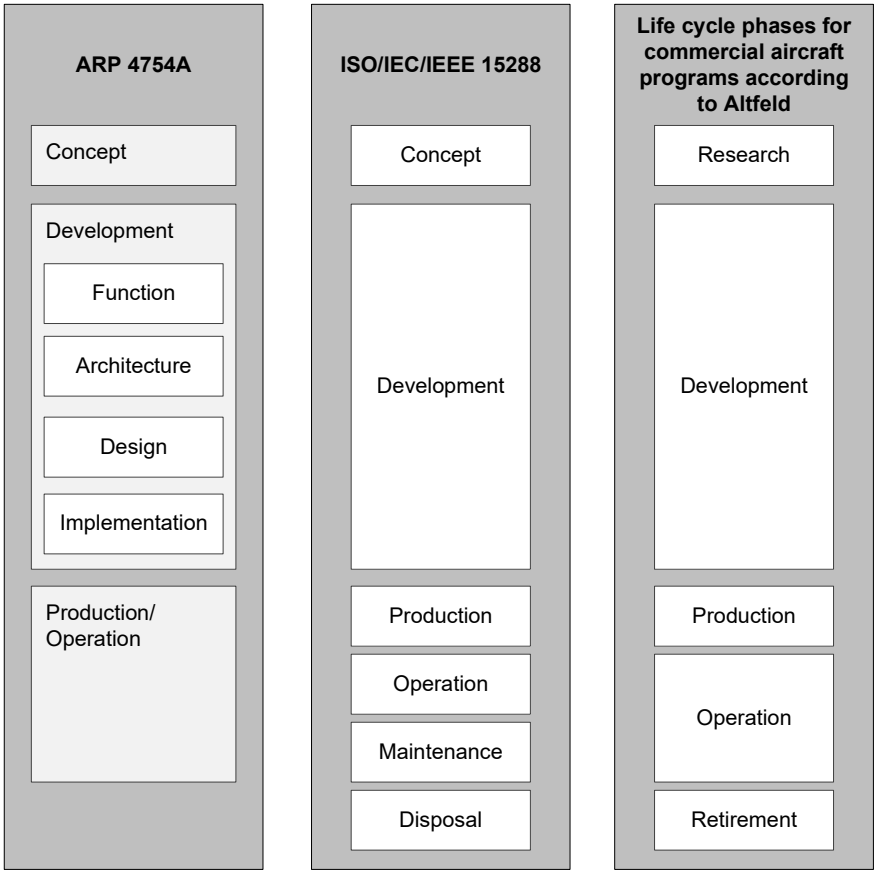


Figure 3.27: Life cycle phases according to the ARP 4754A [6], the ISO/IEC/IEEE standard 15288 [21], and the life cycle phases for commercial aircraft programs by Altfeld [31]

processes ensure that enabling systems and services for maintenance activities are available so that a system can sustain its capabilities. For this purpose, replacements, repairs, or revised system elements are provided and data about faults is collected to improve the system design and avoid future faults. When the life cycle of a system has reached its end, disposal processes terminate the existence of the system in compliance with all applicable requirements, *i.e.* organizational policy as well as environmental, legal, safety, and security aspects. An important part of disposal processes is the deactivation of software so that potential attackers cannot conclude any information about the system from disposed systems [21].

Altfeld proposes the life cycle phases research, development, production, and operation and structures these phases into life cycle sub-phases. Transitions between life cycle sub-phases are initiated by the milestones *Delivery first series aircraft*, *Delivery last series aircraft*, and *Retirement*. After delivery of first series aircraft, product improvements are made for the basic and new versions until delivery of last series aircraft. Subsequent to that, modifications are made until the aircraft is retired. Parallel to the development phase, series and spare parts production is carried out in the production phase. After production, the aircraft is

operated and maintained in the operation phase. Finally, the decommissioning of the aircraft takes place in the retirement phase [31].

The comparison of the concepts shows that there are no significant differences between the life cycle concepts. The ARP primarily focuses on the development phase and thus meets its title “Guidelines for the Development of Civil Aircraft and Systems” while the ISO standard and Altfeld’s concept also describe the production, operation, maintenance, and retirement of the system. As shown in the comparison in Figure 3.27, the ISO and the life cycle phases proposed by Altfeld have much in common. There are only minor differences, *i.e.* the ISO starts with a concept phase that is related to the system under development while Altfeld’s concept considers a research phase, the results of which can be used in various aircraft programs. In addition, Altfeld’s concept proposes an operation phase that includes support and maintenance during the operation of aircraft. The ISO standard, on the other hand, divides the use and support into the operation and maintenance phases.

The analysis of existing approaches for the life phases concept, function, architecture, design and implementation shows that although there are many approaches for the development of self-contained systems, these are only suitable for the development of CPSoS to a limited extent. In particular, the connectivity characteristic of CPSoS, the extensive communication links between CPSs, and the associated complexity require new development methods. The following Chapter 4 presents a further development of the existing approaches into a methodology for the development of CPSoS in aviation.

4 Methods, Processes, and Tools for the Design of Cyber-physical Systems of Systems

The previous chapters describe the evolution ranging from design of self-contained systems with only few physical and almost no external digital interfaces to the service development in CPSoS that requires the consideration of multiple CPSs with various interfaces. Figure 4.1 shows the methods for CPSoS development on the left branch of an enhanced V model in the form of boxes. Research questions (*cf.* Section 2.3 on page 34) related to the methods are shown on the left and allocated to the development life cycle phases of the ARP 4754A [6], *i.e.* concept, function, architecture, design, and implementation.

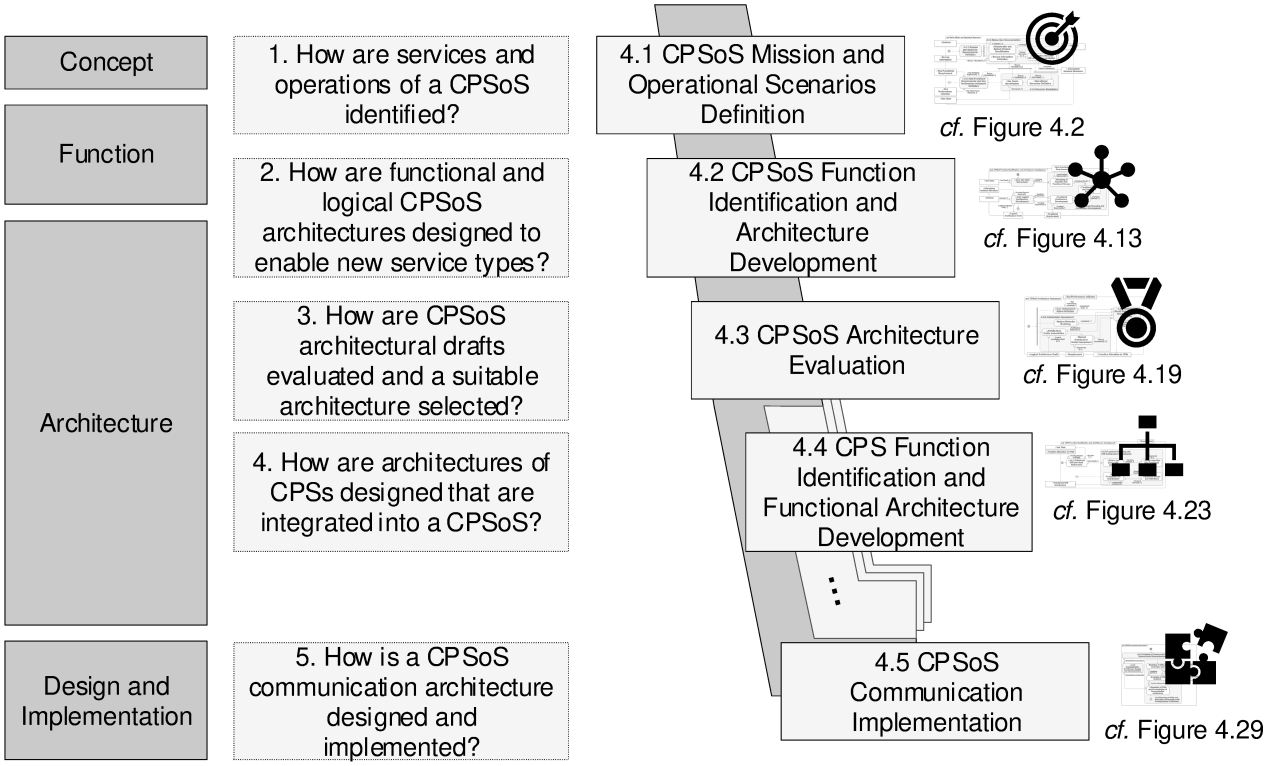


Figure 4.1: Enhanced V model for CPSoS development and transition to CPS development

Four methods support the CPSoS development, *i.e. CPSoS Mission and Operational Scenarios Definition*, CPSoS Function Identification and Architecture Development, CPSoS Architecture Evaluation, and CPSoS Communication Architecture Implementation, and are allocated to the enhanced V model. Part of mission and operational scenarios definition is the identification of CPSoS use cases. CPSoS use cases are the prerequisite for CPSoS

function identification and the development of a functional CPSoS architecture in the CPSoS function identification and architecture development method. CPSoS functions are allocated to CPSs in multiple, different logical architecture drafts. These architecture drafts are evaluated using the CPSoS architecture evaluation method and the most suitable draft is selected for the allocation of CPSoS functions to CPSs.

The allocation of CPSoS functions to CPSs affects the development of CPS. For this reason, CPS development is also considered in the approach developed. The method CPS Function Identification and Functional Architecture Development is allocated to the conventional V model and supports the development of functional CPS architectures that provide functions that are demanded by both the CPSoS and other stakeholders in the conventional context of the CPS. After collection of all required CPS functions and functional CPS architecture development, CPSs can be further designed according to the conventional V model.

In parallel to CPS function identification and functional architecture development, a CPSoS communication architecture is implemented using the CPSoS design and integration method. The CPSoS communication architecture represents the communication paths between CPSs (*cf.* Figure 3.23 on page 78) and can be designed as soon as the distribution of functions to CPSs is finalized by selecting the most suitable logical CPSoS architecture draft. The following sections utilize a so-called intelligent catering CPSoS as an application example for illustration of the developed approach. The intelligent catering CPSoS includes actors and CPSs that are involved in the procurement, production, transport, and service of in-flight catering. Digitization of the intelligent catering CPSoS is expected to result in more accurate catering demand estimations, better passenger experience, and less unused catering.

4.1 CPSoS Mission and Operational Scenarios Definition

In the first method CPSoS Mission and Operational Scenarios Definition, stakeholder needs and expectations towards the CPSoS under development are identified and collected as a basis for the subsequent CPSoS design. The method uses a mission, operational scenarios, use cases, and business requirements to describe stakeholder needs and derives key performance indicators (KPIs) for the evaluation of design results. The associated process is shown in Figure 4.2 and sequentially described in the following sections 4.1.1, 4.1.2, 4.1.3, and 4.1.4.

4.1.1 Mission and Business Requirements Definition

A mission is used in the design of CPSoS to define CPSoS objectives and CPSoS target states [23] in one or few sentences. It is the first step in describing stakeholder needs

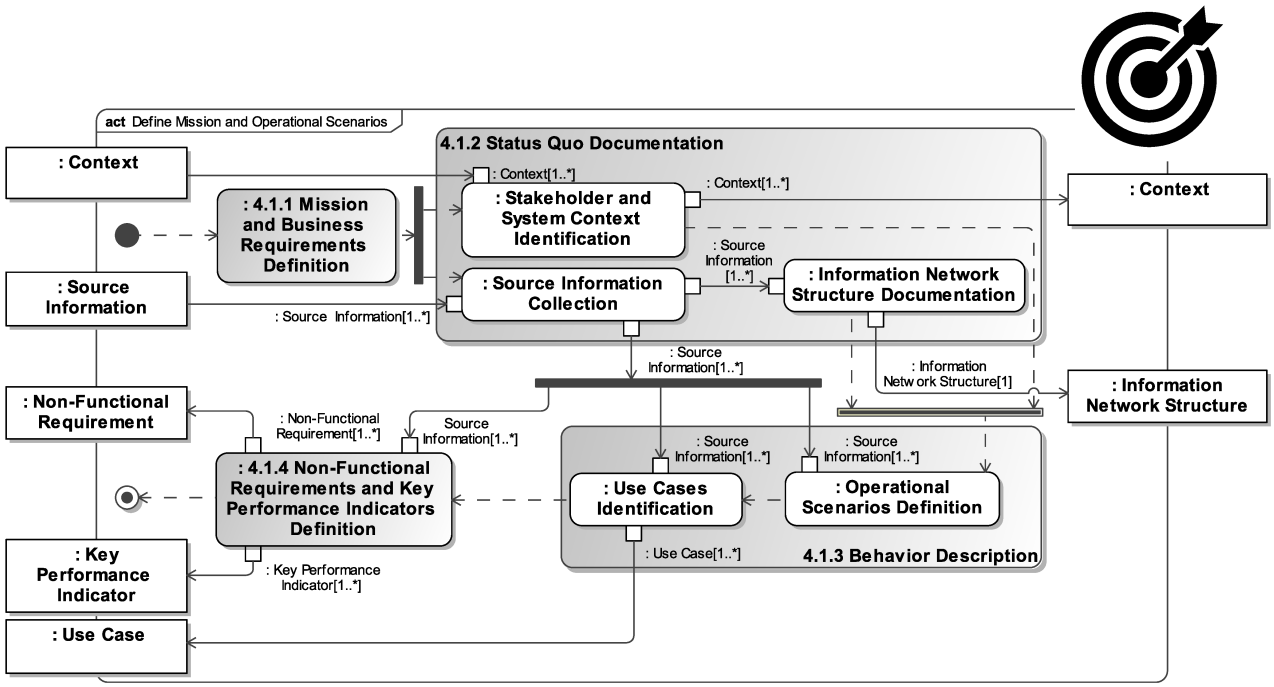


Figure 4.2: SysML activity diagram of the process for CPSoS Mission and Operational Scenarios Definition

and requires discussion by relevant stakeholders. The mission represents a common understanding of the CPSoS to be designed and can be refined when more detailed or corrected expectations of the CPSoS arise. An example of a mission for the intelligent catering CPSoS is shown in Figure 4.3 and is defined as: *The intelligent catering CPSoS should provide increased passenger comfort and satisfaction, optimized cabin operation, and increased airline revenue.* Relevant stakeholders for the definition of this mission are airlines, catering providers, and galley manufacturers. A CPSoS mission is modeled using a *Mission* stereotype that extends the UML element *class*. Missions have ID and text properties, identical to requirement properties.

Discussions about the mission of the CPSoS can reveal stakeholder needs that are more detailed than the mission. For documenting these aspects, detailed stakeholder needs are modeled as business requirements [79]. They are modeled with the *Business Requirement* stereotype that also extends the UML element *class* and has ID and text properties. Business requirements are connected to the mission using a SysML *trace* relationship. An exemplary business requirement is the implementation of an inventory of in-flight catering, as shown in the lower left corner of Figure 4.3, that increases the airlines' overview of catering consumed and is the basis for catering consumption analyses.

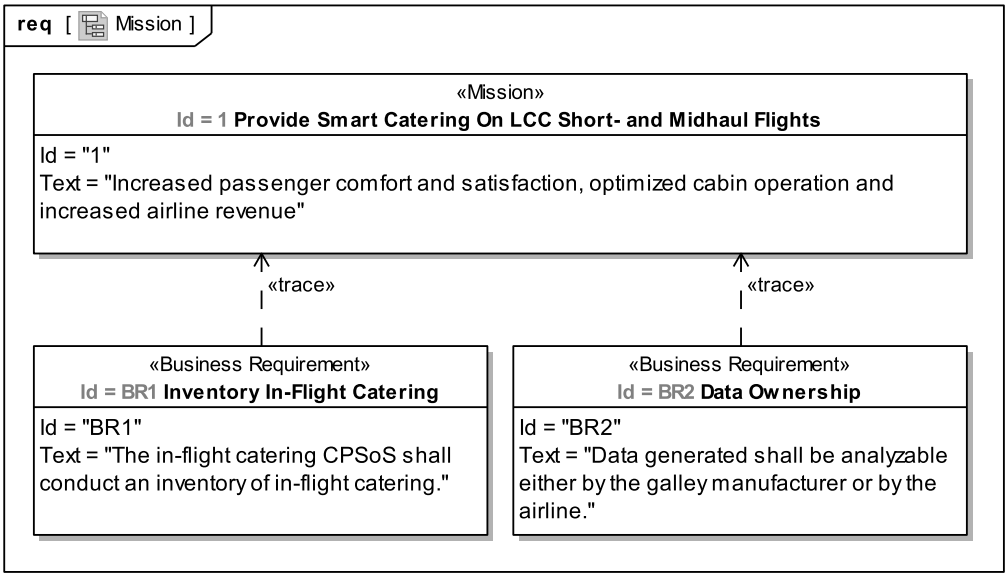


Figure 4.3: Mission and business requirements definition for the intelligent catering CPSoS, modeled in a SysML requirements definition diagram

4.1.2 Status Quo Documentation

Each CPSs that is integrated into a CPSoS already has interfaces to stakeholders and other systems. To get an overview of stakeholders and CPSs of the CPSoS, stakeholder and system contexts are modeled in a SysML block definition diagram, as shown in Figure 4.4 and in Figure 4.5. Stakeholders of the intelligent catering CPSoS are catering providers, passengers, members of the cabin crew, etc. Some stakeholder groups can be decomposed into more granular stakeholders, *e.g.* the Federal Aviation Administration (FAA) and European Union Aviation Safety Agency (EASA) are part of the authorities stakeholder group. Other stakeholder groups can have specializations. As an example, the passenger stakeholder group has the specializations economy class, business class, and first class passenger.

If stakeholders are modeled using SysML actors, properties and behaviors cannot be added. Therefore, stakeholders are modeled as SysML blocks so that stakeholders can be described in more detail by properties and behaviors of the respective blocks. All stakeholder blocks are connected to a CPSoS block via a composition association (recognizable by a black diamond at the end of the CPSoS block, *cf.* top of Figure 4.4) as they are an essential part of the CPSoS. Stakeholder groups such as the authorities in Figure 4.4 are also modeled with SysML composition associations. Specializations of stakeholders are modeled with SysML generalization relationships, as shown for the passenger stakeholder with its passenger classes, so that the behaviors and properties of the general block are inherited by the specialized blocks.

Discussions with stakeholders and analysis of source information provide a first set of

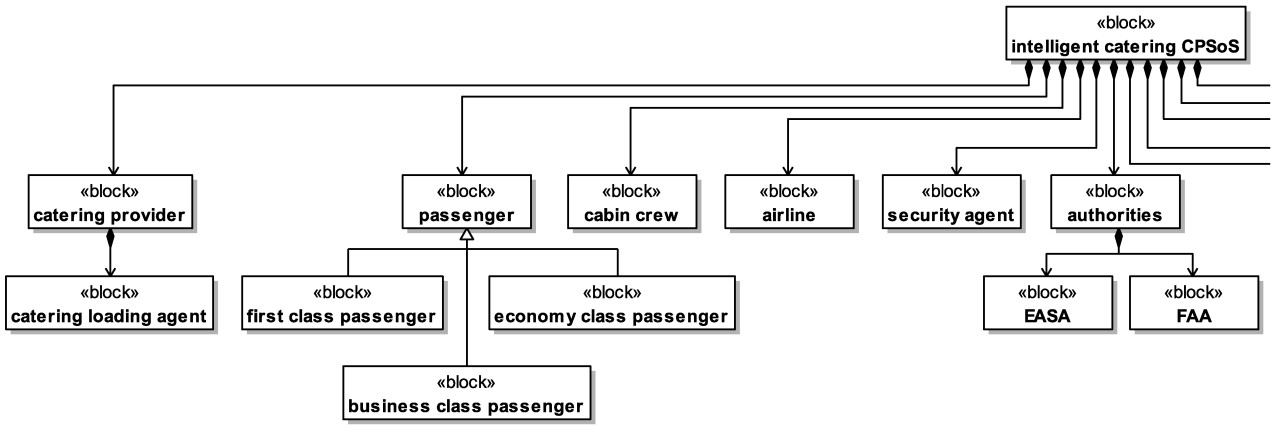


Figure 4.4: Diagram segment of the stakeholder context for the intelligent catering CPSoS, modeled in a SysML block definition diagram. Common elements in block definition diagrams are explained in Figure 2.5

mission-relevant CPSs. Relevant CPSs of the intelligent catering CPSoS are aircraft, airports, catering facilities, turnaround logistics systems, ground stations for data exchange, etc. These CPSs are also modeled as blocks, as shown in the segment of the system context in Figure 4.5.

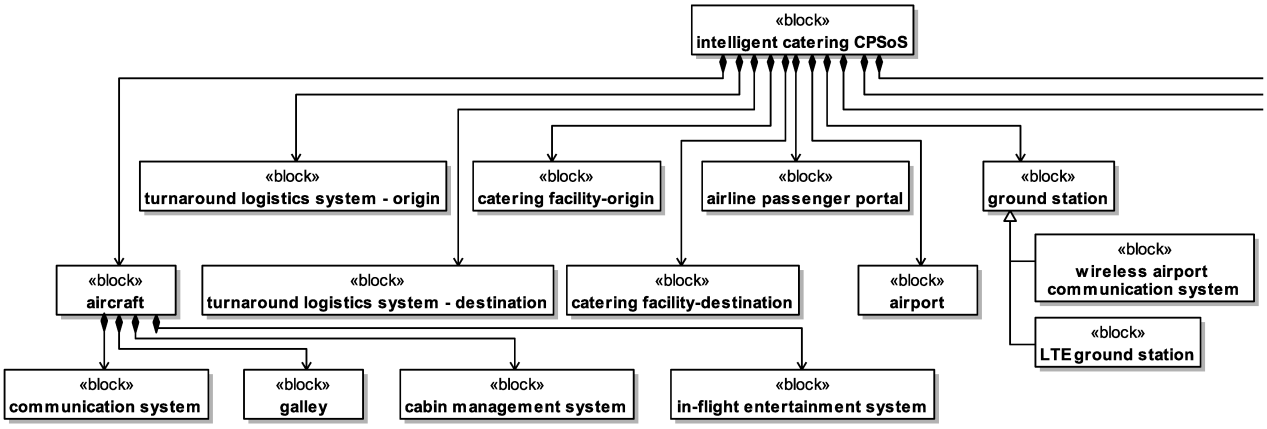


Figure 4.5: Diagram section of the system context for the intelligent catering CPSoS, modeled in a SysML block definition diagram. Common elements in block definition diagrams are explained in Figure 2.5

The system context does not describe specific CPSs such as a specific aircraft with a particular serial or registration number. Instead, the abstraction aircraft is used so that all possible aircraft in the CPSoS are considered. This concept of abstraction is also used in the subsequent design of a logical CPSoS architecture, as described in Section 4.2.3 on page 101. For traceability reasons, the CPS blocks that are modeled the first time in the system context are used in the logical CPSoS architecture.

If there are models that describe CPSs and SoS legacy contexts, these contexts and the model elements contained should be reused in order to avoid modeling stakeholders and systems redundantly. Therefore, it is suggested that contexts be stored in a model for contexts and operational scenarios of the CPSoS. This context and operational scenarios model can be integrated into models for the development of new services in the CPSoS. Then, existing model elements for representing stakeholders and external systems can be reused in multiple models for the development of new CPSoS services. Stakeholders and systems that are not yet part of the contexts and operational scenario model are added to this model. By reusing the contexts and operational scenarios model in multiple models, traceability between all context elements is maintained and duplicates are avoided.

As indicated in the process description for defining missions and operational scenarios in CPSoS in Figure 4.2 on page 87, relevant source information is collected parallel to identification of stakeholders and systems for context definition. Figure 4.6 shows an exemplary block definition diagram for a collection of source elements for the development of an intelligent catering CPSoS. These source elements are grouped into information about status quo, need-relevant information, and regulations. Exemplary source information on the status quo can be obtained through expert interviews, a generic cabin operation breakdown, and a data source and sink overview. Workshops with relevant stakeholders in the form of airline, OEM, and supplier representatives are another source of stakeholder needs. Regulations are often the basis for non-functional requirements that constrain possible solutions. The General Data Protection Regulation contains regulations for dealing with personal data.

There are various forms of source information, *e.g.* conversations, emails, informal documents such as notes and logbooks, formal requirement documents, system specifications, workshop results, standards and so on [81]. All these types of source information are modeled as a block with the stereotype *Source Element* in a SysML block definition diagram. This integration of source information into the model supports traceability between operational scenarios and use cases and their origin so that consequences of changes to the system design can be estimated. In addition, further development results, such as elements in functional and logical architectures, can be traced back to their source elements in the further design process. This traceability supports the estimation of consequences of changes to system elements as well as to source information. If, for example, an interface should be added to an element in the functional architecture, relevant source information can be analyzed so that possible conflicts with regulations or stakeholder requirements can be identified. On the other hand, the effects of changes in source elements can also be examined, for example when standards or regulations change. Suitable tools for the analysis are relation maps and dependency matrices that are not specified in the SysML but are integrated in many modeling tools.

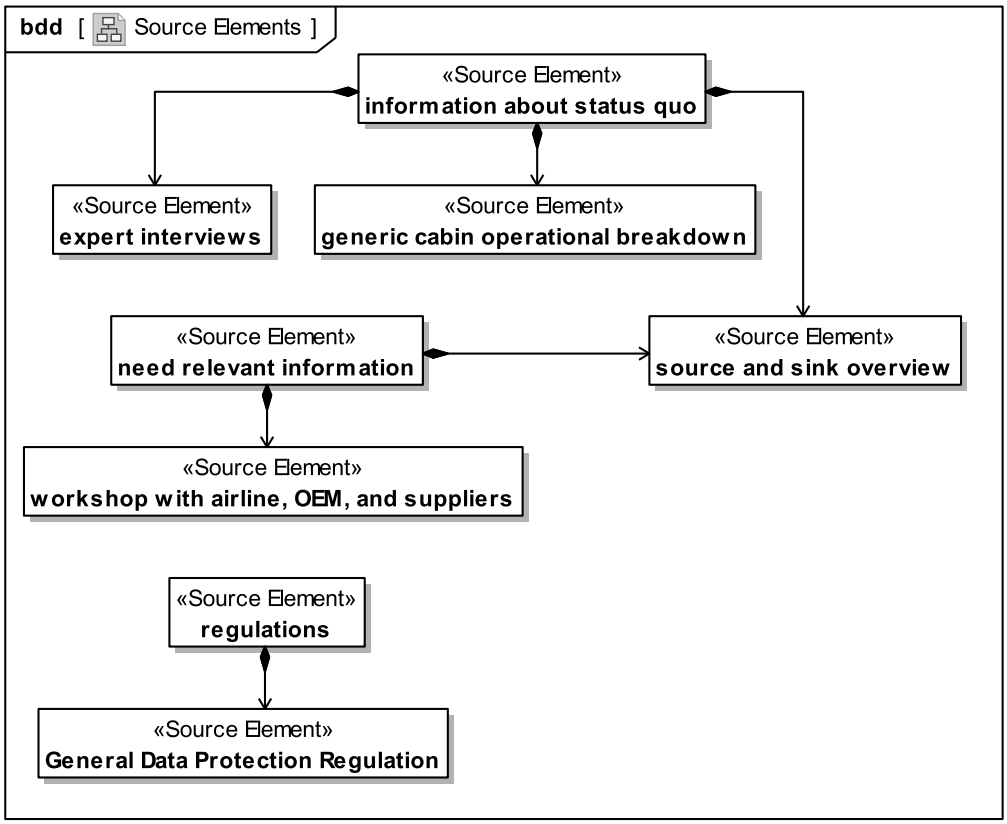


Figure 4.6: Relevant source elements for the intelligent catering CPSoS, modeled in a SysML block definition diagram

Existing communication paths between systems can be used for the implementation of CPSoS functions that require communication between CPSs. Information about existing communication paths can be obtained from source information, in particular from discussions with stakeholders about the system context. The results are modeled in the information network structure, as shown for the intelligent catering CPSoS in the internal block diagram in Figure 4.7. SysML elements in internal block diagrams are described in Figure 4.8. The example presented considers a subset of possible paths for the exchange of data between aircraft and other systems on the ground. These systems are modeled as part properties in an internal block diagram. Systems from the system context (*cf.* Figure 4.5 on page 89) typify the part properties. Interfaces and communication paths between CPSs are modeled by means of proxy ports and connectors. In the example, data is exchanged between the aircraft and ground systems via WiFi when the aircraft is on the ground or via Long Term Evolution (LTE) technology while in flight. Aircraft antennas supporting WiFi and LTE are interfaces between the systems considered [26, 51]. They are modeled as interface blocks and typify proxy ports in the information network structure. The information about existing communication paths in the information network structure is used for the development of logical CPSoS architecture drafts in the CPSoS Function Identification and Architecture Development method, described in Section 4.2 on page 97. The logical CPSoS architecture

drafts use these existing communication paths, and if communication paths are missing, they are added to the CPSoS.

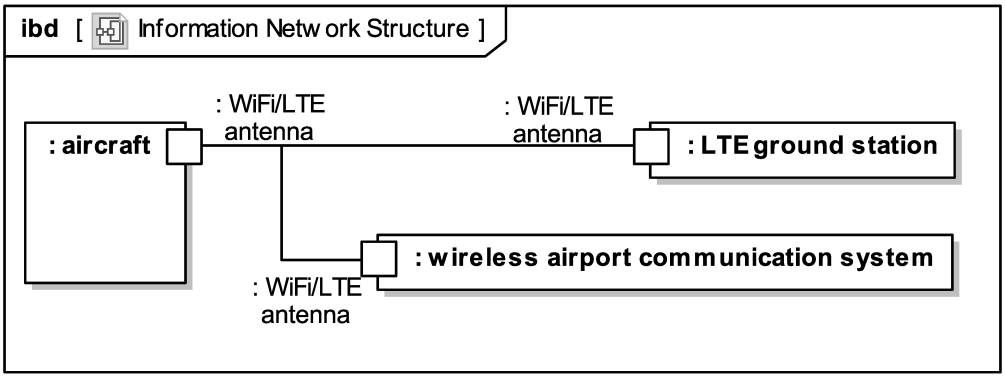


Figure 4.7: Information network structure for the intelligent catering CPSoS, modeled in a SysML internal block diagram

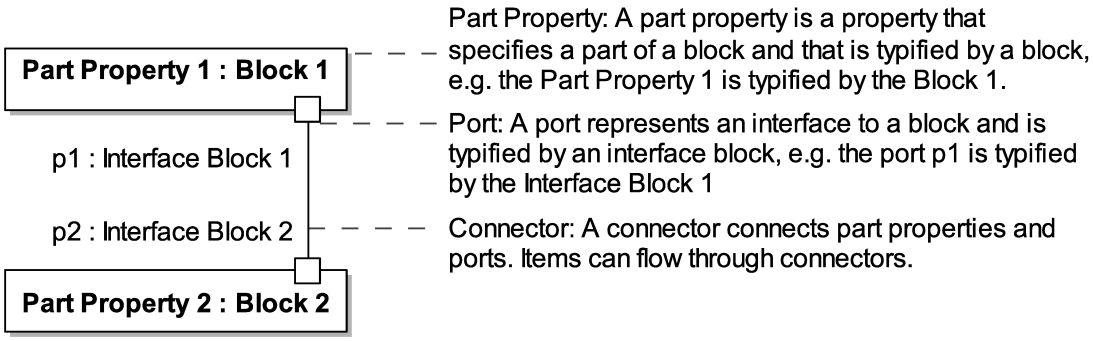


Figure 4.8: Common elements in SysML internal block diagrams [12]

4.1.3 Behavior Description

As shown in Figure 4.2 on page 87, the description of the desired CPSoS behavior is the next step after status quo documentation. Collected source information from status quo documentation leads to the definition of operational scenarios of the CPSoS. Figure 4.9 shows an operational scenario for the operation *passenger provisioning* in the intelligent catering CPSoS. This exemplary operational scenario describes the general process for providing beverages, catering, and other items to passengers. The scenario begins with the question about the catering orders of the passengers by the flight crew of the airline. Based on the passengers' order, catering items are prepared and provided to the passenger, consumed, and non-consumed catering items and waste are subsequently collected by the flight crew.

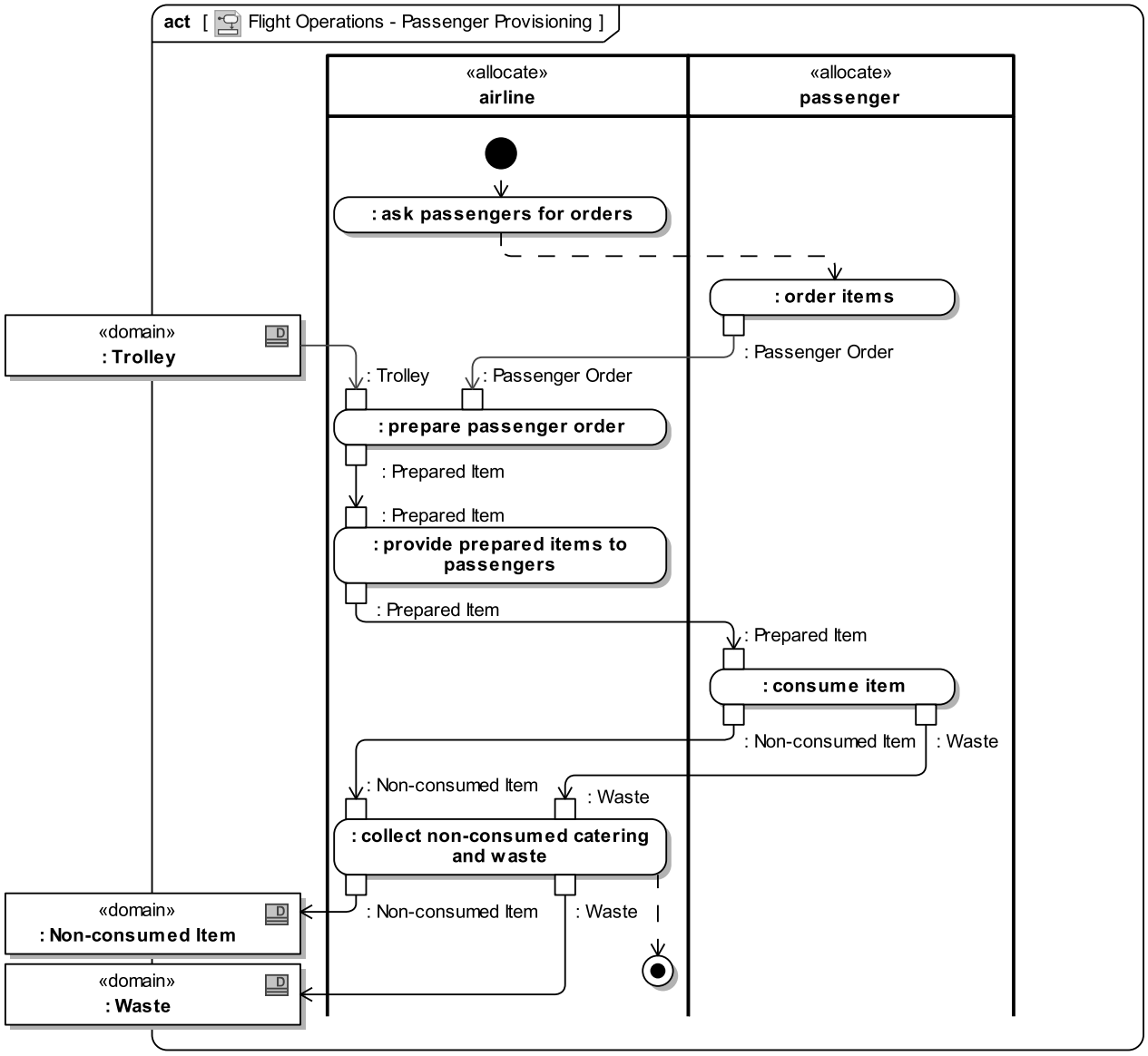


Figure 4.9: Operational scenario for passenger provisioning in the intelligent catering CPSoS, modeled in a SysML activity diagram

In general, operational scenarios describe the desired behavior of the CPSoS and the interaction of stakeholders and CPSs in the form of SysML activity diagrams. Partitions in the activity diagram represent stakeholders and CPSs and contain call behavior actions for describing stakeholder and CPSs behavior. Using call behavior actions instead of actions allows further detailing of operational scenarios in subsequent development phases. Activities describe the behavior of call behavior actions and can be detailed as a sub-process of an operational scenario in another SysML activity diagram. Inputs and outputs of the operational scenario are modeled as activity parameter nodes. Object flows connect these activity parameter nodes with call behavior actions.

The first call behavior action in the exemplary operational scenario in Figure 4.9 illustrates

the solution-independent characteristic of operational scenarios. Asking passengers for orders is independent from a solution. Flight crews have performed this operational activity for decades without any technical support. Combining CPSs in a CPSoS can lead to new use cases, *e.g.* automated meal suggestions to passengers based on previous passenger meal choices. A set of CPSoS use cases for the intelligent catering CPSoS is shown in Figure 4.10.

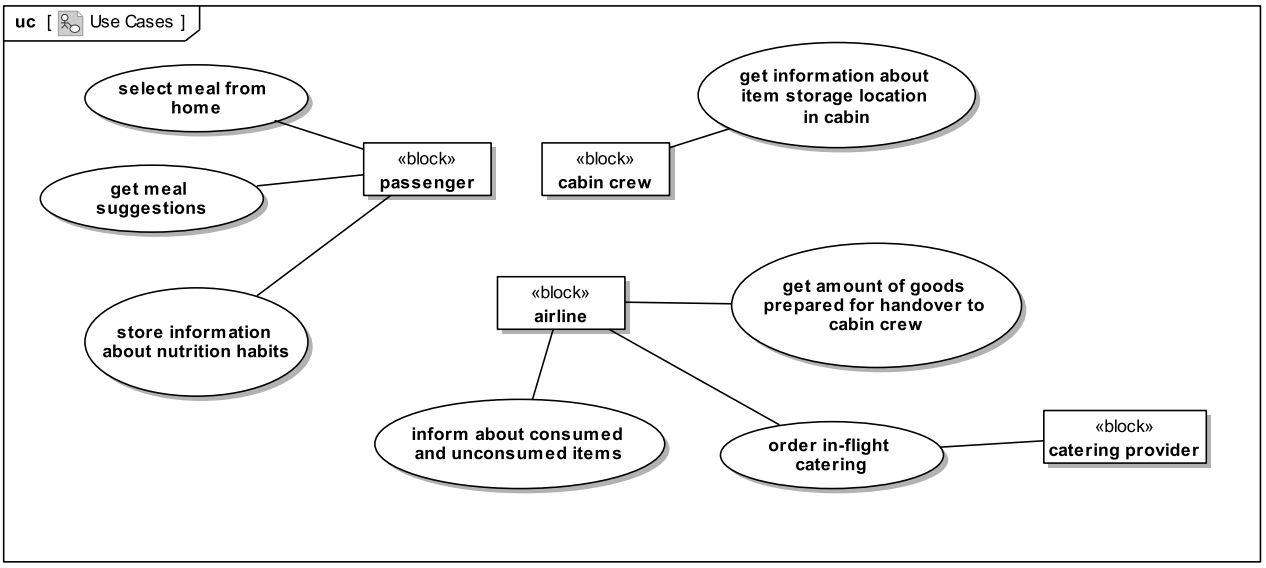


Figure 4.10: Use cases for the intelligent catering CPSoS, modeled in a SysML use case diagram

Discussions about operational scenarios with stakeholders show possible new services provided by the CPSoS. As an example, airlines lack information about consumed and unconsumed items for better planning of future catering orders, resulting in the use case “inform about consumed and unconsumed items.” Cabin crew members often search for stored items in trolleys and galleys. An intelligent catering CPSoS with item identification capabilities could inform about item storage locations, expressed in the use case “get information about item storage location in cabin.”

Use cases are modeled in a SysML use case diagram and are connected to relevant stakeholders by means of an association. Relevant stakeholders are modeled as blocks and are initially modeled in the CPSoS stakeholder context shown in Figure 4.4 on page 89. As already discussed in Section 4.1.2 about status quo documentation, stakeholders are modeled as SysML blocks instead of SysML actors, allowing stakeholder properties and behaviors to be added to the model.

CPSs can be combined in a variety of ways to accomplish different missions. The diagram in Figure 4.11 shows two exemplary missions for the air transport system. The mission “Provide Smart Catering On LCC Short- and Midhaul Flights” is used for demonstrating application of the developed method. The abbreviation LCC stands for “low-cost carrier.”

Another mission is “Increased Reliability of Aircraft Galleys.” Both missions are connected to operational scenarios and these operational scenarios are traced to use cases in turn. The diagram in Figure 4.11 shows only a subset of missions, operational scenarios, and use cases for demonstrating the principle of traceability. In general, traceability is established through trace relationships between missions, operational scenarios, and use cases in SysML requirements and block definition diagrams. Both types of diagrams allow the representation of the required model elements and the trace relationships that connect them.

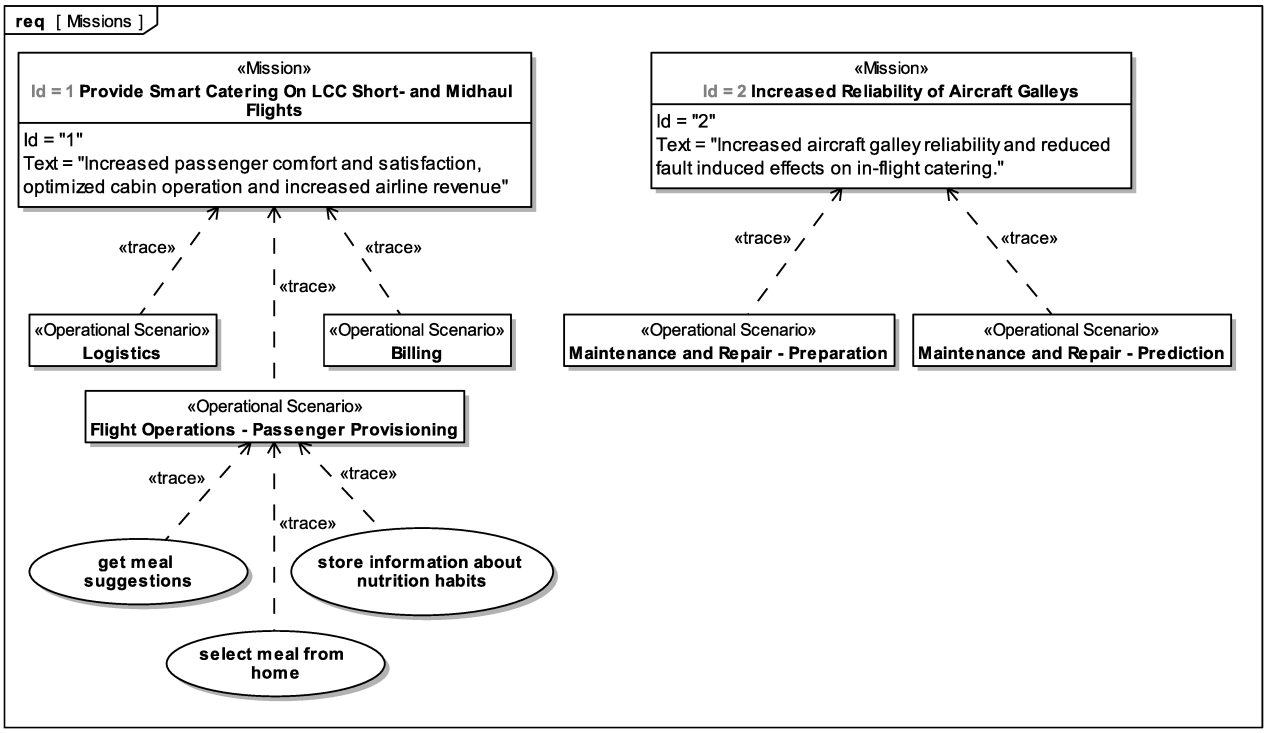


Figure 4.11: Traceability between missions, operational scenarios, and use cases, modeled in a SysML requirements diagram

4.1.4 Non-Functional Requirements and Key Performance Indicators Definition

In the last steps of the mission and operational scenarios definition method, non-functional requirements and KPIs are defined. Non-functional requirements constrain possible realizations of system functions [79] and thus consider aspects that must be respected for realization of functional requirements. Source information in the form of standards and regulations can provide possible non-functional requirements. Further non-functional requirements can be identified during discussions with stakeholders about operational scenarios and use cases. They are modeled with a non-functional requirements stereotype and are described in a textual manner. Non-functional requirements are considered in the subse-

quent CPSoS function identification and architecture development method (*cf.* Section 4.2 on page 97) as an input for the identification of CPSoS functions in the functional grouping process step.

KPIs are defined as a preparation for the evaluation of logical CPSoS architecture drafts in the method CPSoS Architecture Evaluation (*cf.* Section 4.3 on page 103). The method for mission and operational scenarios definition is a suitable opportunity for defining KPIs and values that are required for KPI calculation because stakeholder expectations towards the CPSoS are identified in this phase. Expected cost, revenue per passenger, expectations with respect to passenger satisfaction, and experienced privacy are exemplary values for evaluating the intelligent catering CPSoS. These values can be used for the calculation of KPIs in the form of expected margin per passenger and a service evaluation result. In addition, values for inventory accuracy, cost for inventory, and reliability of inventory data transmission are defined for evaluating logical CPSoS architecture drafts with respect to inventory capabilities.

Values and KPIs are modeled as value properties of a logical CPSoS architecture draft block in a SysML block definition diagram as shown in Figure 4.12. Logical CPSoS architecture drafts are modeled as specializations of this block according to the CPSoS function and architecture development method, described in Section 4.2 on page 97. As a specialization of this block, logical CPSoS architecture draft blocks inherit its value properties for KPI calculation. Figure 4.12 shows the KPIs and values for the evaluation of architecture drafts for the intelligent catering CPSoS.

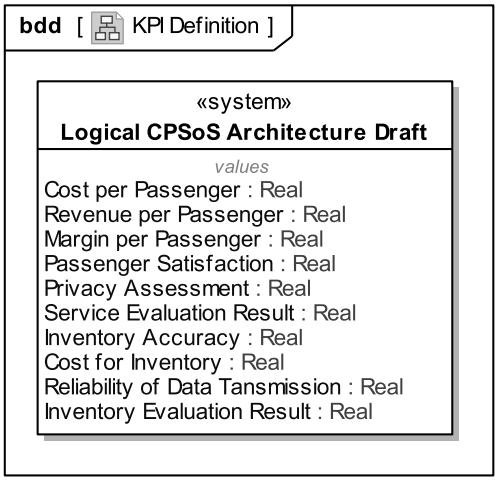


Figure 4.12: KPI definition for the intelligent catering CPSoS, modeled in a SysML block definition diagram

The use of SysML parametric diagrams and instance specifications for automated calculation of KPIs is presented in the CPSoS architecture evaluation method (*cf.* Section 4.3 on page 103).

4.2 CPSoS Function Identification and Architecture Development

The method CPSoS Function Identification and Architecture Development supports detailing and analysis of desired system behavior for function identification and architecture development. Section 3.2 on page 56 describes existing concepts for identification of functions and subsequent functional architecture development and analyses these concepts with respect to CPSoS characteristics. The analyzed approaches are adapted and combined for the method presented in this section. The related process is shown in Figure 4.13 and contains process steps for use case refinement with activities, functional grouping, modeling of functional blocks, interfaces, and the functional architecture. In the last process step, logical CPSoS architecture drafts are derived from the functional architecture. A logical CPSoS architecture considers CPSs as logical elements and allocates functional blocks from the functional CPSoS architecture to CPSs. Since functions can be assigned to CPSs in different ways, multiple logical CPSoS architecture drafts exist. To select the most suitable logical CPSoS architecture draft, the drafts are evaluated on the basis of KPI defined in the mission and operational scenario definition in the subsequent CPSoS architecture evaluation method (*cf.* Section 4.3 on page 103).

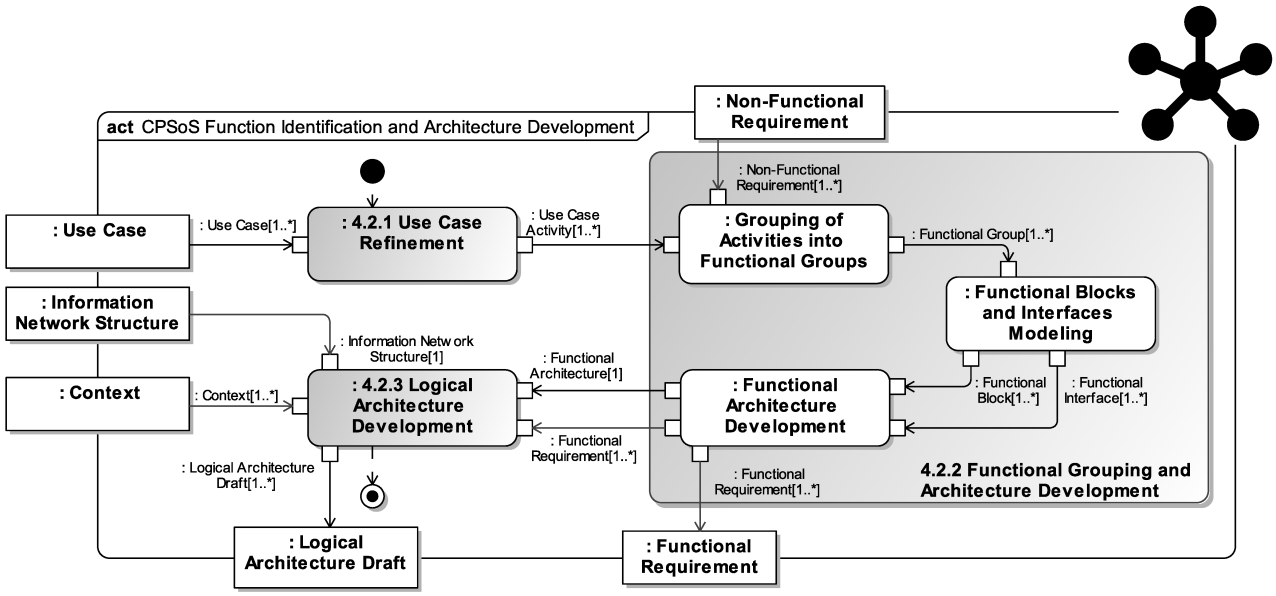


Figure 4.13: SysML activity diagram of the process for CPSoS Function Identification and Architecture Development

4.2.1 Use Case Refinement

First, CPSoS use cases (*cf.* Figure 4.10 on page 94) are refined by use case activities as known from the FAS method [87]. Figure 4.14 shows the activity diagram for the use case

activity of the use case *inform about consumed and unconsumed items*, initially modeled in Figure 4.10 on page 94.

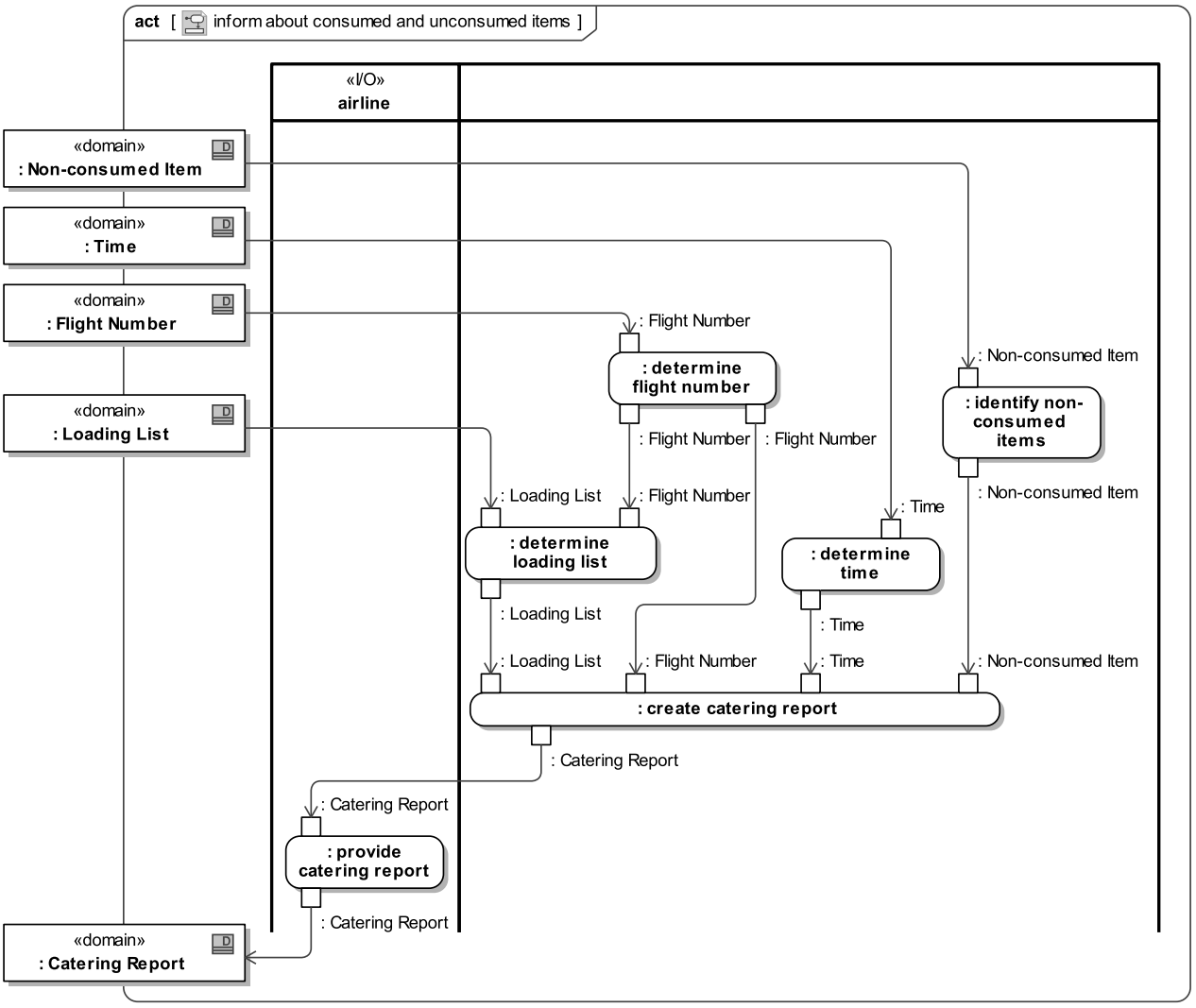


Figure 4.14: CPSoS use case activity, modeled in a SysML activity diagram

The conventional FAS method distinguishes between activities that are performed by the system at its interfaces to actors or other systems and activities inside the systems. Activities with an interface to actors or other systems are allocated to an I/O compartment in activity diagrams. The method for CPSoS function identification and architecture development extends this approach by allocating the respective stakeholders or external systems to the compartment. As an example, the airline is assigned to the I/O compartment in Figure 4.14. This leads to multiple I/O compartments for the distinction of activities at CPSoS interfaces to different actors or external systems. The method thus meets the requirement of being able to consider interfaces to the large number of other systems characteristic of a CPSoS.

The use case activity is modeled by means of call behavior actions. Call behavior actions

own a behavior. In the case of the use case activities, this behavior is an activity. As an example, the upper left call behavior action in Figure 4.14 has the activity *determine flight number* as behavior, identifiable by the colon in the call behavior caption. Call behavior actions are used so that activities can be grouped to functional groups in the next process step. Alternatively, actions could be used instead of call behavior actions to describe use case activities in more detail. However, actions would be owned by the respective use case activity and could not be compared to each other in the following analysis for the identification of functional groups. Activities that are selected as a behavior of call behavior actions are more suitable for subsequent functional grouping, since their owner can be chosen arbitrarily.

Object flows indicate the flow of data, material, and energy and connect pins of call behavior actions. These pins are typed by domain blocks to specify the transferred objects. The same domain blocks typify activity parameter nodes at the diagram frame. Activity parameter nodes represent the inputs and output of the use case activity.

4.2.2 Functional Grouping and Architecture Development

Multiple use cases and associated use case activities are modeled for capturing all demanded CPSoS functions. Thereby, similar CPSoS functions may be required. A function set that covers all stakeholder needs on the one hand but is not providing overlapping functionality on the other hand is favorable for a functional architecture that is as simple as possible while providing all demanded functions. Use cases of the intelligent catering CPSoS that require similar functions are, *e.g.* the selection of meals by passengers from home before flight and storing information about nutrition habits of passengers. Both use cases include collection of data about nutrition habits for different purposes. The use case *select meal from home* includes the activities *collect available meals* and *receive flight information* while the use case *store information about nutrition habits* contains the activity *store nutrition habits*. These three activities have data collection in common so that they are grouped into the functional group *data collection*.

The functional grouping is supported by a dependency matrix, as shown in a segment of a dependency matrix for the intelligent catering CPSoS in Figure 4.15. The use case activities are entered into the columns of the matrix and analyzed for similarities. Based on the similarities, functional groups are modeled as SysML blocks with the stereotype *functional group* and entered in the rows of the matrix. The activities are then allocated to the functional groups using a trace relationship. The example functional group *data collection* is entered in the second row and traced to the aforementioned use case activities.

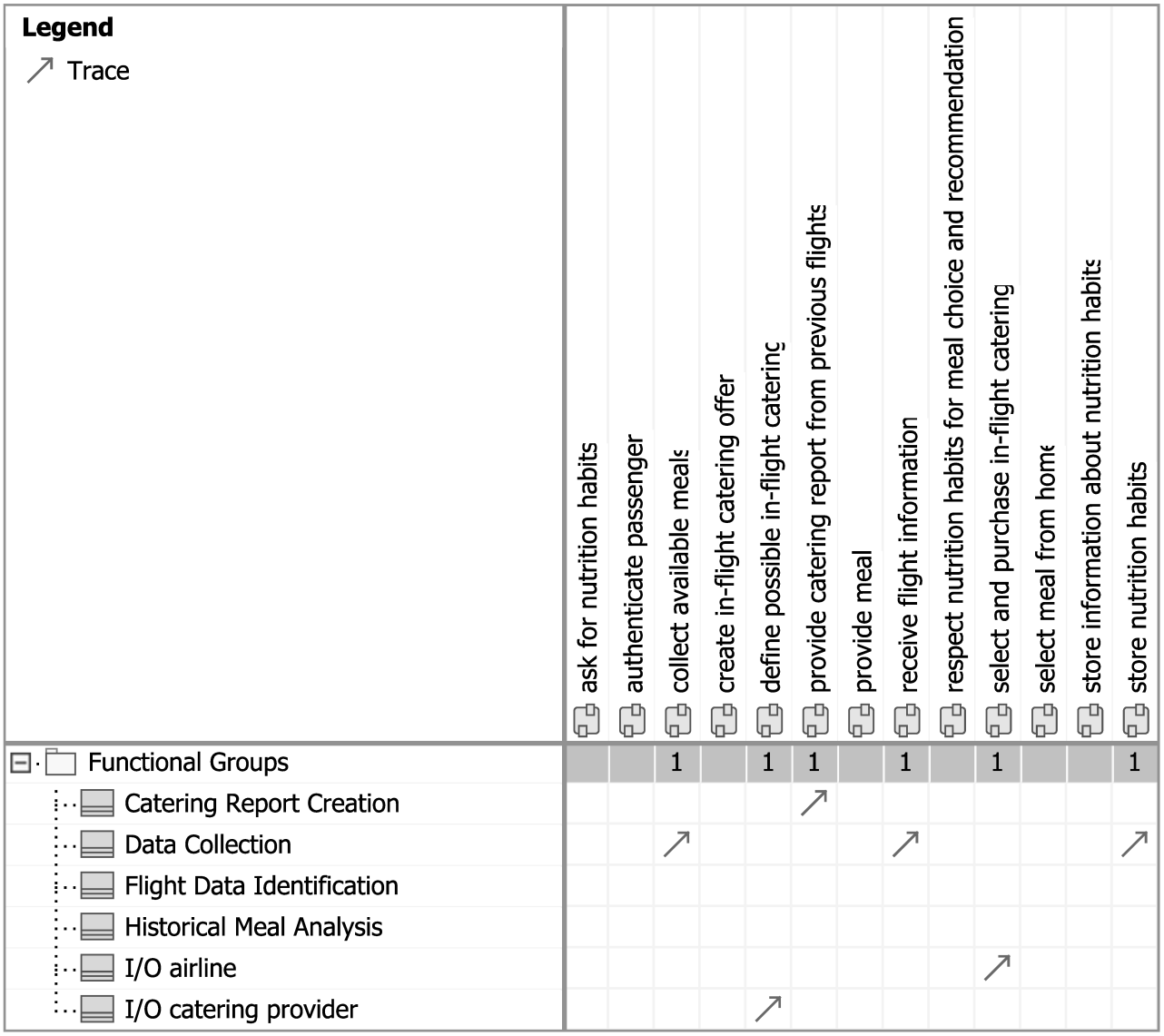


Figure 4.15: Segment of functional grouping of CPSoS use case activities in a dependency matrix of Cameo Systems Modeler

The set of functional groups represents the required CPSoS functions and is the basis for functional architecture development in the next process steps. A functional architecture is based on functional blocks that represent the required system functions and functional interfaces between functional blocks. Figure 4.16 shows an exemplary segment of the functional architecture of the intelligent catering CPSoS.

The functional architecture is modeled in an internal block diagram. For each functional group, a functional block is created with the same name. These functional blocks typify part properties that are depicted in the internal block diagram. The connection between functions is represented by connectors in the internal block diagram. The connectors connect proxy ports of the part properties and are further specified by an item flow, recognizable

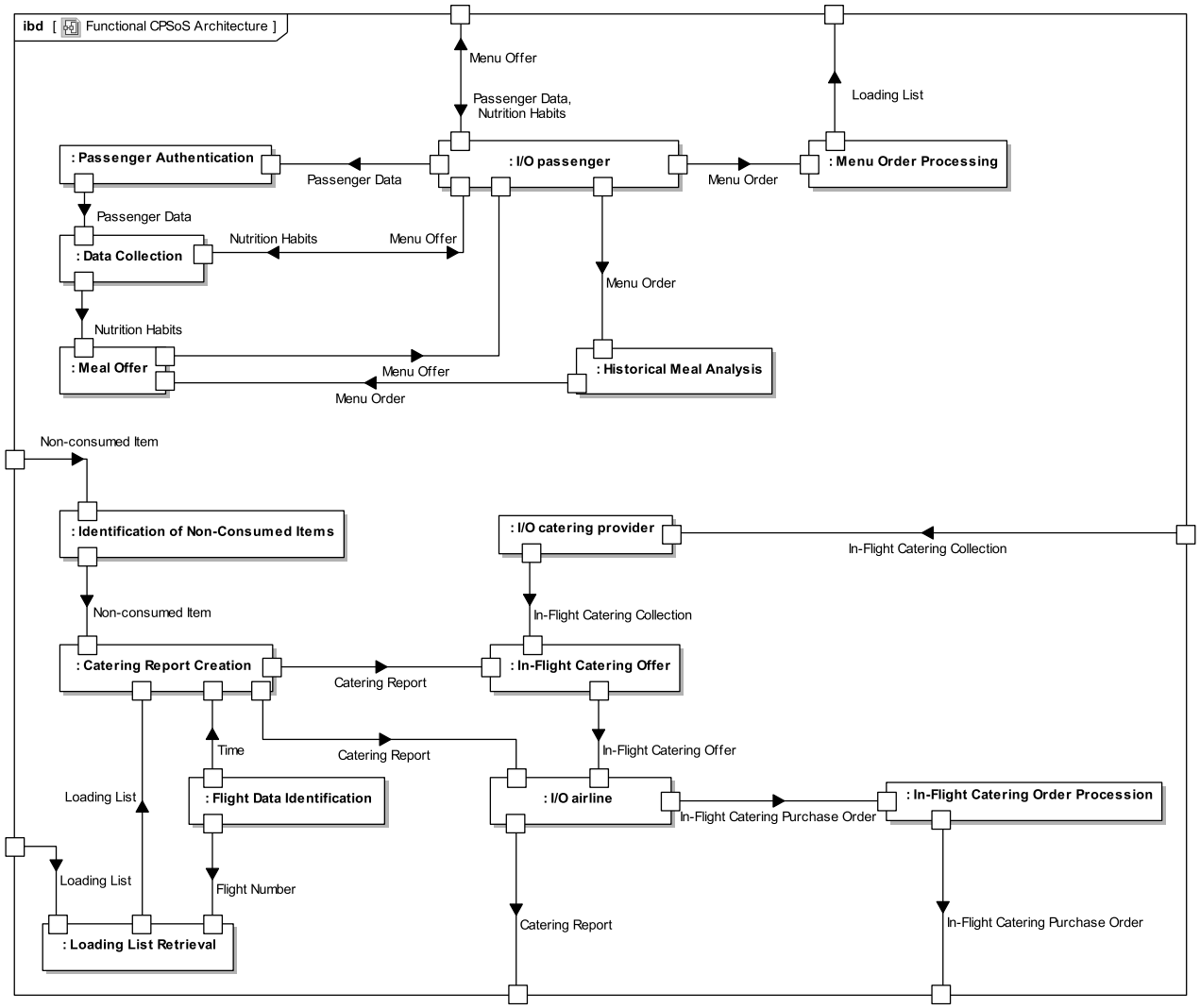


Figure 4.16: Segment of the functional CPSoS architecture of the intelligent catering CPSoS, modeled in a SysML internal block diagram

by the arrow at connectors. The domain blocks that are used for specification of activity parameter nodes and activity pins in use case activity modeling (*cf.* previous Section 4.2.1) are modeled as conveyed items of the item flows. Functions related to menu selection analysis and ordering are modeled in the upper part of the functional CPSoS architecture in Figure 4.16. The lower part of the architecture contains functions for inventory conducting and provision of inventory results. In the next process step, these functions are allocated to CPSs in a logical CPSoS architecture.

4.2.3 Logical Architecture Development

A functional architecture is a solution-neutral representation of desired CPSoS functions and their connections. In the next process step, the functional architecture is the basis for the development of a logical architecture that allocates CPSoS functions to CPSs. Since various

possibilities for function allocation to CPSs exist, multiple logical CPSoS architecture drafts can be developed. Evaluating logical architecture drafts using key performance indicators (KPIs) supports selection of the most suitable architecture draft and is described in the subsequent Section 4.3.

Figure 4.17 shows an exemplary segment of a logical CPSoS architecture draft that considers automated inventory and provision of inventory results. Each logical CPSoS architecture is modeled in an internal block diagram that is owned by a logical CPSoS architecture draft block. CPSs included in the CPSoS are modeled as blocks in the CPSoS system context (*cf.* Figure 4.5 on page 89). These CPS blocks typify part properties that represent CPSs in the logical CPSoS architecture internal block diagram. The part properties are owned by the associated logical CPSoS architecture block. Functions are assigned to CPSs as part properties of the CPS blocks.

In the exemplary architecture in Figure 4.17, the functions *Identification of Non-Consumed Items* and *Flight Data Identification* are assigned to the CPS *aircraft*, drawn at the top left of the internal block diagram. For this purpose, part properties are created for the *aircraft* block. These part properties are typified by the functional blocks *Identification of Non-Consumed Items* and *Flight Data Identification*. Then, the internal structure of the CPS part properties is displayed, as shown for the CPSs *aircraft*, *catering management system*, and *airline catering software* in Figure 4.17. Interfaces between CPSs and functions are modeled with proxy ports and connectors in combination with item flows. Proxy ports of functional blocks and their connections are already modeled in the functional CPSoS architecture and can be displayed automatically without additional modeling effort. Missing communication paths between functional blocks and the boundaries of logical blocks and between logical blocks are added. An example of a communication path between CPSs is the connector between the CPSs *aircraft* and *airline catering software* with the item flow of *Non-Consumed Item*, *Time*, and *Flight Number*. Information about existing communication paths between CPSs comes from the information network structure (*cf.* Figure 4.7 on page 92).

The exemplary segment of the logical CPSoS architecture draft allocates the functions *Identification of Non-Consumed Items* and *Flight Data Identification* to the CPS *aircraft*. Alternative architecture drafts may allocate these functions to a system on the ground in order to avoid the effort for equipping the aircraft with inventory technology. Figure 4.18 shows a segment of a second logical architecture draft in that the functions considered are allocated to a CPS with the name *catering inventory system*. The catering inventory system is a system on the ground for automated inventory after unloading catering from the aircraft. Consequently, the next task is the selection of the most suitable architecture draft from the set of alternative logical CPSoS architectures. The following section describes how KPIs from mission and operational scenarios definition (*cf.* Section 4.1.4 on page 95) are used for the evaluation of architecture drafts.

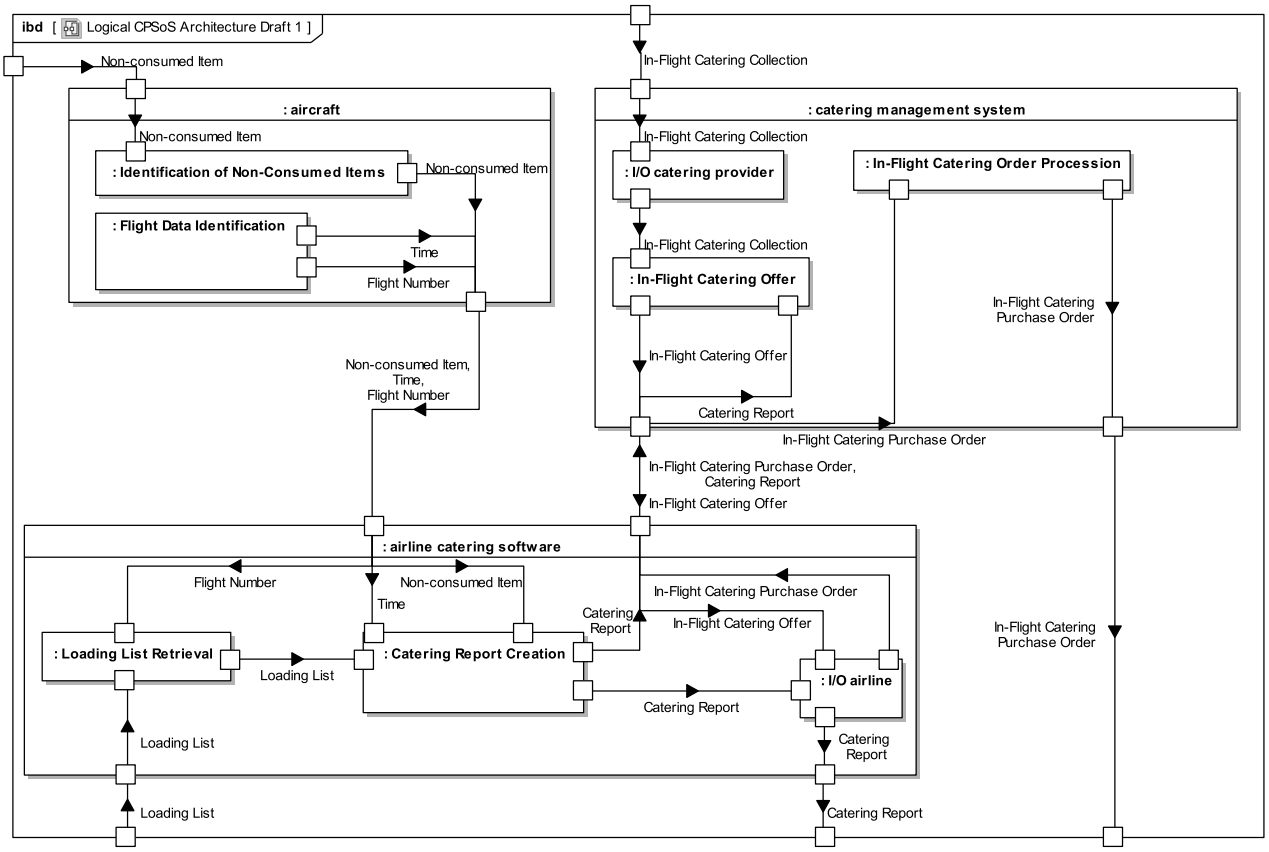


Figure 4.17: Exemplary logical CPSoS architecture draft for the intelligent catering CPSoS, modeled in a SysML internal block diagram

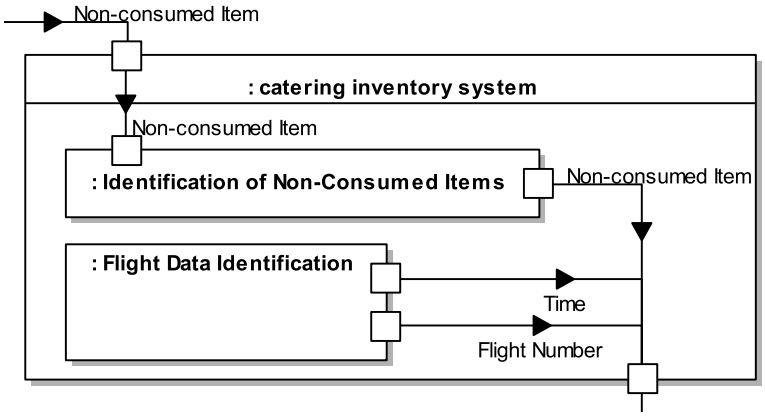


Figure 4.18: Segment of an alternative logical CPSoS architecture draft for the intelligent catering CPSoS, modeled in a SysML internal block diagram

4.3 CPSoS Architecture Evaluation

A set of different logical CPSoS architecture drafts is developed during CPSoS function identification and architecture development as presented in the previous Section 4.2. The developed architecture drafts differ in the allocation of functions to CPSs. This section

presents an approach for selecting the most suitable CPSoS architecture draft. For this purpose, architecture drafts are evaluated against KPIs that are defined during the mission and operational scenarios method as described in Section 4.1.4 on page 95. Figure 4.19 presents the associated process. As a preparation for the calculation of evaluation results, evaluation rules are defined, the system behavior is modeled, architecture drafts are instantiated and manually evaluated. The results of these activities are used for the calculation of an evaluation result that supports the selection of the most suitable architecture draft.

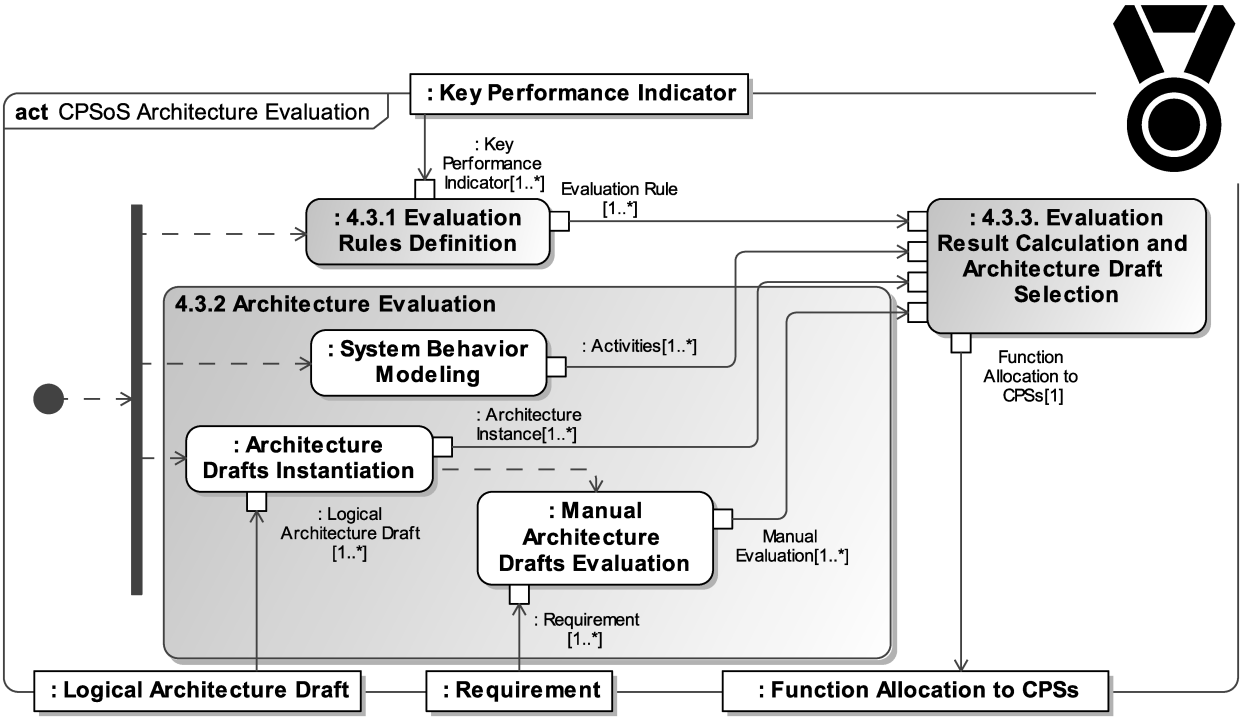


Figure 4.19: SysML activity diagram of the process for CPSoS Architecture Evaluation

4.3.1 Evaluation Rules Definition

The block definition diagram in Figure 4.20 shows the model elements for logical CPSoS architecture drafts evaluation. Blocks with the stereotype *system* at the bottom of the diagram represent the architecture drafts and are already modeled during the last process step of CPSoS function identification and architecture development. These blocks own internal block diagrams that show the logical CPSoS architecture, as shown in Figure 4.17 on page 103. All architecture draft blocks are specializations of the *Logical CPSoS Architecture Draft* block, shown in the middle of Figure 4.20. This block is modeled in the definition of non-functional requirements and KPIs phase of the mission and operational scenarios definition method, as shown in Figure 4.12 on page 96. KPIs and values for KPI calculation are modeled as value properties and evaluation rules are defined as SysML constraints.

In the case of the intelligent catering CPSoS, the exemplary values and KPIs

- Inventory Accuracy
- Cost for Inventory
- Reliability of Data Transmission
- Inventory Evaluation Result

are introduced for evaluating inventory solutions.

A SysML simulation calculates KPI on the basis of the modeled value properties and constraints. A block with the name *Logical CPSoS Architecture Assessment* (*cf.* top of Figure 4.20) is modeled and connected to the draft block with a composition association. This ensures that all architecture instances are contained in one instance specification with the classifier block *Logical CPSoS Architecture Assessment* and thus assessment results for all architecture drafts are calculated in a single simulation. Alternatively, assessment results would have to be calculated for each architecture draft individually.

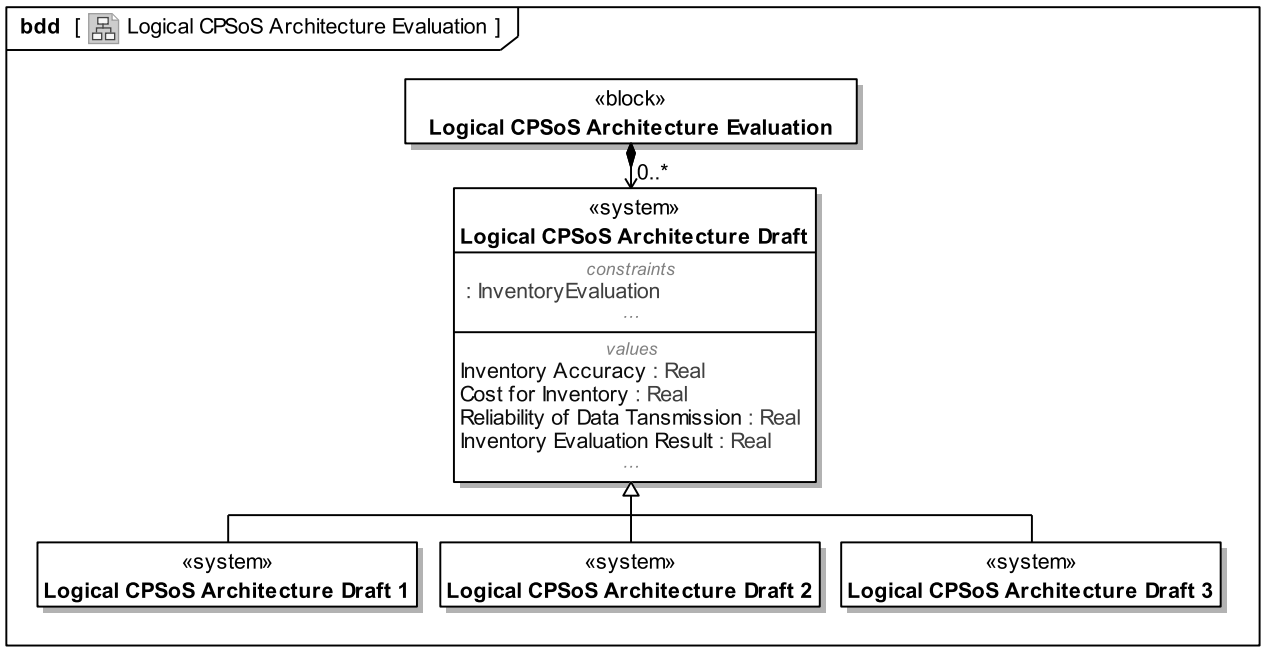


Figure 4.20: SysML block definition diagram for evaluating logical architecture drafts

The rules for KPI calculation are defined as constraints in a SysML parametric diagram. The parametric diagram in Figure 4.21 shows the calculation of an inventory evaluation result that considers the value properties *Inventory Accuracy*, *Cost for Inventory*, and *Reliability of Data Transmission* of the *Logical CPSoS Architecture Draft* block (*cf.* Figure 4.20). The *Inventory Evaluation Result* is calculated as a weighted sum of the value properties as defined in the *InventoryCalculation* constraint.

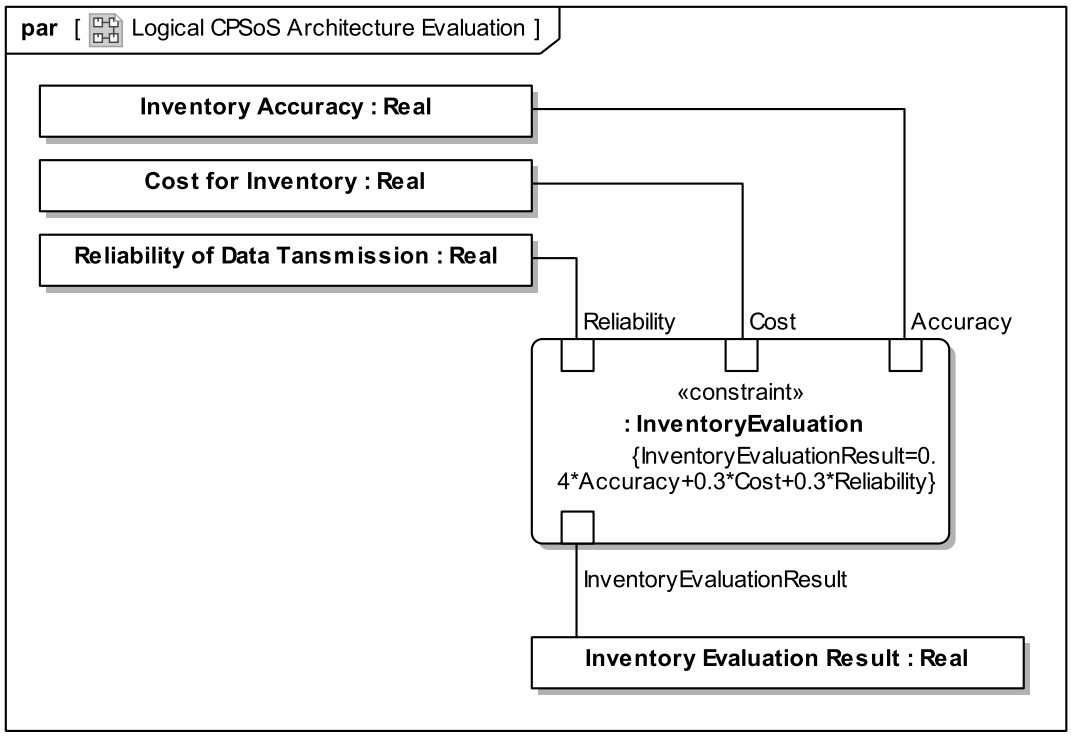


Figure 4.21: Logical CPSoS architecture evaluation in a SysML parametric diagram

4.3.2 Architecture Evaluation

For the evaluation of architecture drafts, numerical values must be assigned to the value properties of the logical architecture blocks. Numerical values can be added to value properties of blocks as so-called default values. However, modeling of instance specifications that have the classifier of the logical blocks is favorable because values of instance specifications can be read, modified, and created during SysML simulations. Therefore, one instance specification with the associated block as classifier is created for each logical CPSoS architecture draft block. As an example, the instance specification *Logical CPSoS Architecture Draft 1* in the first row of the instance table in Figure 4.22 is created for the *Logical CPSoS Architecture Draft 1* block in Figure 4.20 and uses this block as classifier. In addition to instance specifications for the logical CPSoS architecture draft blocks, one instance specification with the classifier *Logical CPSoS Architecture Evaluation* (*cf.* top of Figure 4.20 on page 105) is created. The logical CPSoS architecture draft instance specifications are added as slot values to the evaluation instance specification so that the subsequent simulation for KPI calculation only has to be executed once for all architecture drafts. Slot values of instance specifications correspond to properties of blocks. A slot value can be created for each property of a block that classifies the instance specification that owns this slot value.

Numerical evaluation values, *e.g.* for the *Inventory Accuracy* value property, can be determined either by manual evaluation or in simulations of system behavior, as described in the

following subsection. For manual evaluation, evaluation values are added as slot values to instance specifications of the logical CPSoS architecture drafts. As an example, the evaluation value *Inventory Accuracy* is considered in the following. It describes the expected accuracy of inventory solutions in a value range between zero for poor and one for good accuracy. The logical CPSoS architecture draft 1, partially shown in Figure 4.17 on page 103, assigns the function *Identification of Non-Consumed Items* to the aircraft. This results in an inventory accuracy value of 0.9 (*cf.* third column in the table shown in Figure 4.22) since items are identified in the cabin and thus close to the origin of changes. As presented in a segment of the logical CPSoS architecture draft 2 in Figure 4.18 on page 103, the function *Identification of Non-Consumed Items* is allocated to a catering inventory system on the ground in the second architecture draft. Consequently, the evaluation value for inventory accuracy is reduced to 0.3 since unconsumed catering and trash is carried in the same units back to the catering facility after flight, resulting in difficult differentiation between trash and unconsumed catering. In contrast to that, the second architecture draft is evaluated better with respect to expected reliability of data transmission. Sending catering inventory data from an aircraft during flight or on the ground as suggested in the first draft is more complicated and susceptible to transmission disturbances than sending data from a catering inventory system at the catering production site, as considered in the second draft. For this reason, reliability of data transmission is rated 0.9 for the second and 0.6 for the first architecture draft.

Additional values for logical CPSoS architecture evaluation may result from simulations of system behavior. Simulations in combination with structural feature actions can be used for determining values such as expected times or, if a group of test persons is available, values that evaluate human behavior or reactions towards the system. Structural feature actions in SysML activities allow reading, creating, modifying, and clearing structural features, *i.e.* slot values of instance specifications. The values obtained in simulations are added to logical CPSoS architecture draft instance specifications and can be used for calculating the overall evaluation result.

4.3.3 Evaluation Result Calculation and Architecture Draft Selection

As soon as evaluation values are determined in a manual evaluation and simulations of logical CPSoS architecture drafts, the evaluation results can be calculated. The calculation is defined in constraints, as shown in the parametric diagram in Figure 4.21. This parametric diagram and the constraints used are owned by the *Logical CPSoS Architecture Draft* block (*cf.* Figure 4.20 on page 105). Since all architecture drafts are specializations of the *Logical CPSoS Architecture Draft* block, the parametric diagram and applied constraints are used for calculating the evaluation result for all architecture drafts. In addition to the parametric diagram and constraints, a simulation configuration is modeled as a preparation for the simulation. The instance specification with the classifier *Logical CPSoS Architecture*

Evaluation is set as the execution target and result location of the simulation configuration. The execution target determines which instance specification is executed for the simulation. Simulation results are stored in the result location. When the configured simulation is executed, results are stored as slot values in the instance specifications. The instance table in Figure 4.22 shows the evaluation values and the calculated evaluation result. As described in Section 4.3.2 on page 106 about manual evaluation of architecture drafts, values are in a range between zero for poor and one for good evaluation results. Based on this scale, the logical CPSoS architecture draft 1 reaches the best inventory evaluation result with a value of 0.69. With the support of this evaluation result, the most suitable architecture draft can be selected and thus determines how functions are allocated to CPSs in the CPSoS.

#	Name	Inventory Accuracy : Real	Cost for Inventory : Real	Reliability of Data Tansmission : Real	Inventory Evaluation Result : Real
1	Logical CPSoS Architecture Draft 1	0.9	0.5	0.6	0.69
2	Logical CPSoS Architecture Draft 2	0.3	0.6	0.9	0.57
3	Logical CPSoS Architecture Draft 3	0.4	0.6	0.5	0.49

Figure 4.22: Logical CPSoS architecture evaluation results in a generic table of Cameo Systems Modeler

4.4 CPS Function Identification and Functional Architecture Development

The chosen CPSoS architecture affects the architecture of CPSs that are part of the CPSoS. CPSs must maintain their origin functions in order to satisfy their stakeholders. New CPS functions must be added for proper integration into the CPSoS. For example, a catering trolley for aircraft must store in-flight catering and support the flight crew during flight, thus meeting the needs of its origin stakeholders. Integration of the aircraft and the trolleys contained into an intelligent catering CPSoS requires additional trolley functions, *e.g.* for inventory of catering items. This section presents a method for considering both origin system functions and additional functions for CPSoS integration, for functional CPS architecture development. The associated process is shown in Figure 4.23.

Inputs for the CPS function identification and functional architecture development method are the function allocation from the chosen logical CPSoS architecture (*cf.* Section 4.3 on page 103), origin CPS use cases, and possible requirements that must be considered during the identification of functions. The allocated CPSoS functions from the selected logical CPSoS architecture are modeled as cps-machine use cases. Origin CPS use cases and cps-machine use cases are then refined by use case activities. This procedure is

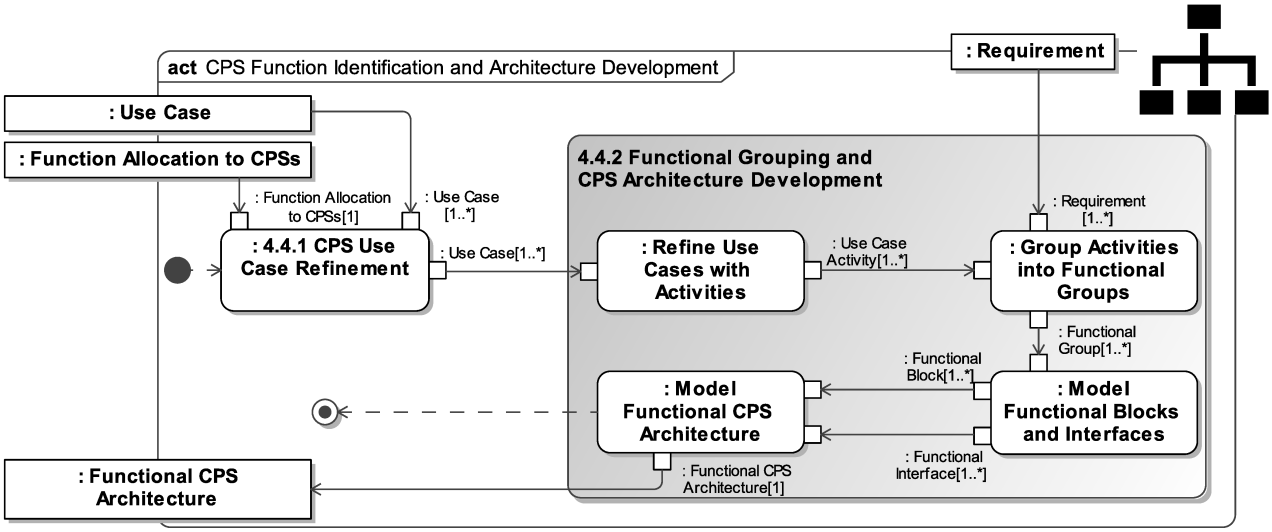


Figure 4.23: SysML activity diagram of the process for CPS Function Identification and Functional Architecture Development

similar to the refinement of CPSoS use cases, as described in the method CPSoS Function Identification and Architecture Development in Section 4.2 on page 97, but is elaborated at the CPS level.

Similar activities are grouped into functional groups. A functional block and related functional interfaces between functional blocks are created for each functional group. The functional blocks and functional interfaces are used for modeling the functional CPS architecture. In the following, the previous example of the intelligent catering CPSoS is taken up and the method is presented by means of the development of a smart trolley. Smart trolleys are in-flight catering trolleys and part of the intelligent catering CPSoS. They provide classic trolley functions such as the storage and transport of in-flight catering and are supplemented by functions for automated identification of the catering items contained.

4.4.1 CPS Use Case Refinement

First, CPS use cases are modeled in use case diagrams, as shown in Figure 4.24. Use cases are marked with stereotypes *cps-human* and *cps-machine*. cps-human use cases consider behavior demanded by human stakeholders, thus resulting in a predominant amount of human interaction. The cps-machine stereotype is used for use cases that utilize communication infrastructure with a predominant amount of interaction to non-human actors. Both stereotypes extend the metaclass “Use Case.” In the example of the smart trolley, inventory of trolley contents is relevant to human stakeholders and to other CPSs that are part of the intelligent catering CPSoS, *i.e.* the airline catering software. The cps-human use case “inform about content” is relevant to the cabin crew and considers their information about trolley content. The corresponding cps-machine use case “identify non-consumed

items” is connected to the airline catering software since information about non-consumed items is provided to the airline catering software. The connection of the smart trolley to the airline catering software for information provision about non-consumed items is determined in the logical CPSoS architecture, shown in Figure 4.17 on page 103. Both use cases are specializations of the abstract “inventory contents” use case. Modeling of an abstract use case for similar cps-human and cps-machine use cases allows reusing activities in subsequent use case refinement. More use cases of the smart trolley are shown at the bottom of Figure 4.24. They represent classic cps-human use cases by the cabin crew and consider the storage and provision of catering.

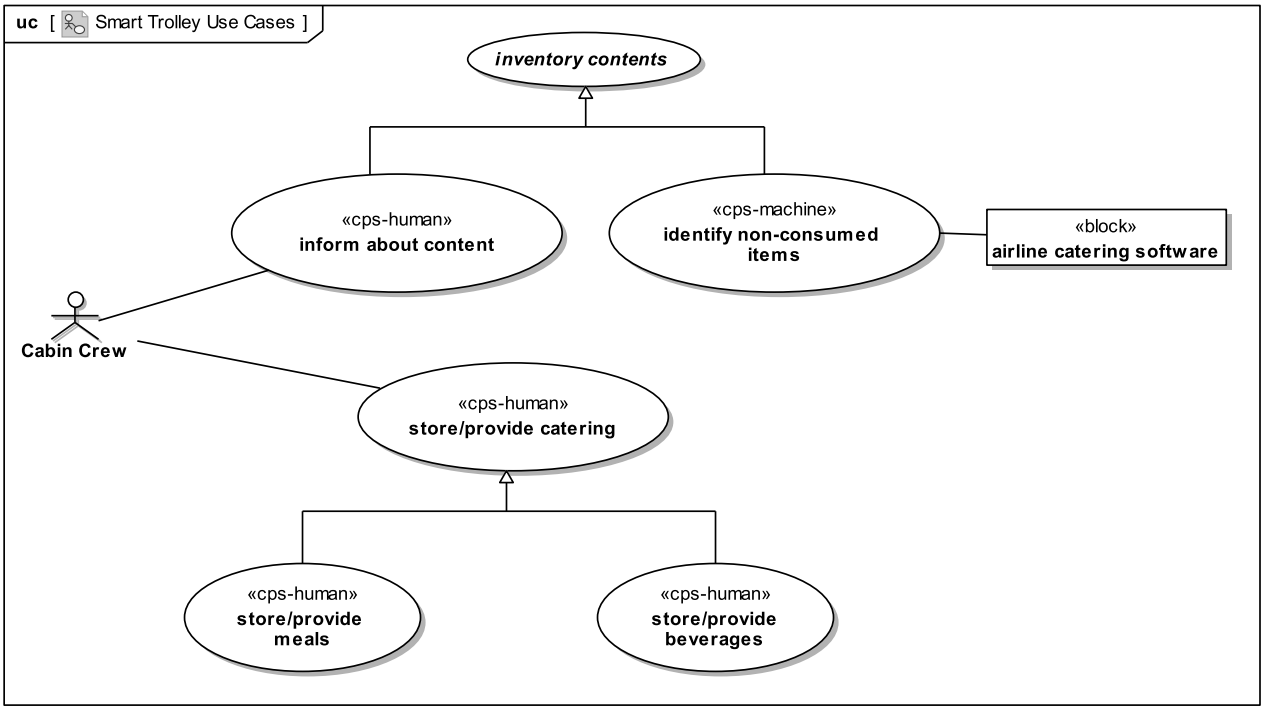


Figure 4.24: CPS-human and CPS-machine use cases for functional CPS architecture development, modeled in a SysML use case diagram

Use cases are refined by use case activities in the second process step of the method. Use case activities describe the desired system behavior in more detail using call behavior actions. The activities called by the call behavior actions are analyzed in subsequent process steps for the identification of CPS functions. Figure 4.25 shows an exemplary activity diagram for the activity of the “identify non-consumed items” use case. The activity diagram is ordered by partitions. Partitions with an I/O stereotype contain activities that are allocated to system interfaces to actors or cooperating CPSs. The airline catering software is an example of a cooperating CPS of the smart trolley: The smart trolley provides data about trolley content to the airline catering software. Activities that are performed by the system without predominant influence by actors or cooperating CPSs are allocated to the partition without I/O stereotype. “scan trolley contents” and “collect scan results” are performed by the smart trolley and therefore allocated to the system partition.

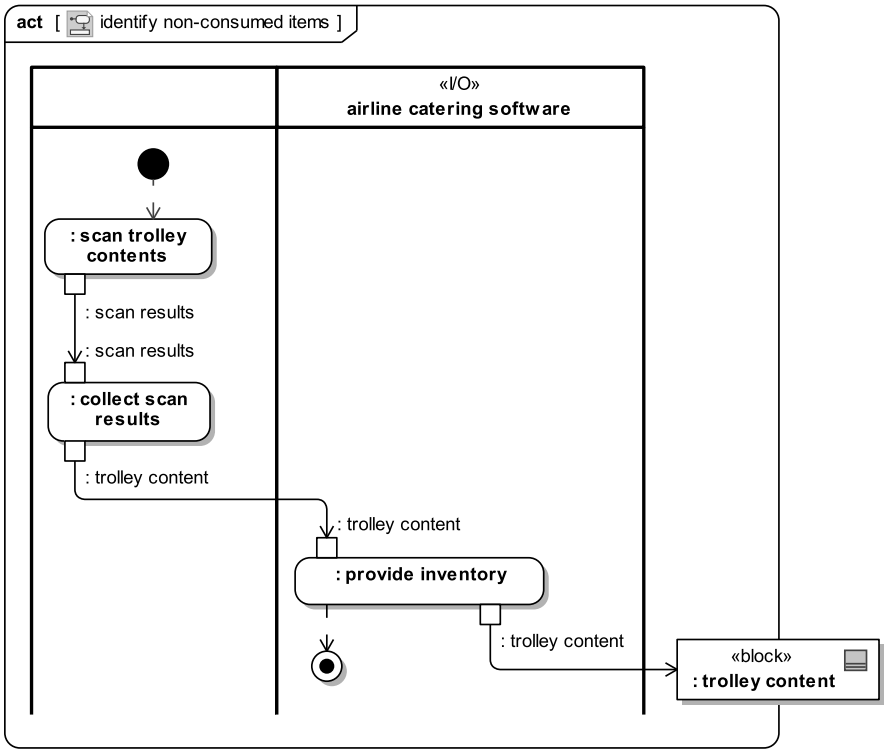


Figure 4.25: Use case activity for the cps-machine use case “identify non-consumed items,” modeled in a SysML activity diagram

4.4.2 Functional Grouping and CPS Architecture Development

The “identify non-consumed items” use case and its owned activity are only one example of the use cases and activities of the smart trolley. The desired system behavior is described in multiple use cases and related use case activities. Different use cases may require identical system functions; similar cps-machine and cps-human use cases in particular are accompanied by a high probability for at least partially identical functions. As an example, the cps-machine use case “identify non-consumed items” and the cps-human use case “inform about content” (*cf.* Figure 4.24) require similar activities, *i.e.* activities for scanning of trolley contents, preparation of scan results, and provision of inventory data to an interface to the airline catering software or the cabin crew respectively. For identifying identical functions such as the identification of smart trolley contents, the previously described activities are compared to each other in a dependency matrix. Identical and similar activities are allocated to functional groups using a trace-relationship as shown in Figure 4.26 according to the FAS method presented in [87]. Thus, activities associated with trolley content identification, *i.e.* “scan trolley contents” activity in cps-machine use case “identify non-consumed items” and “scan content” activity in cps-human use case “inform about content,” are traced to the functional group “Scan Content.”

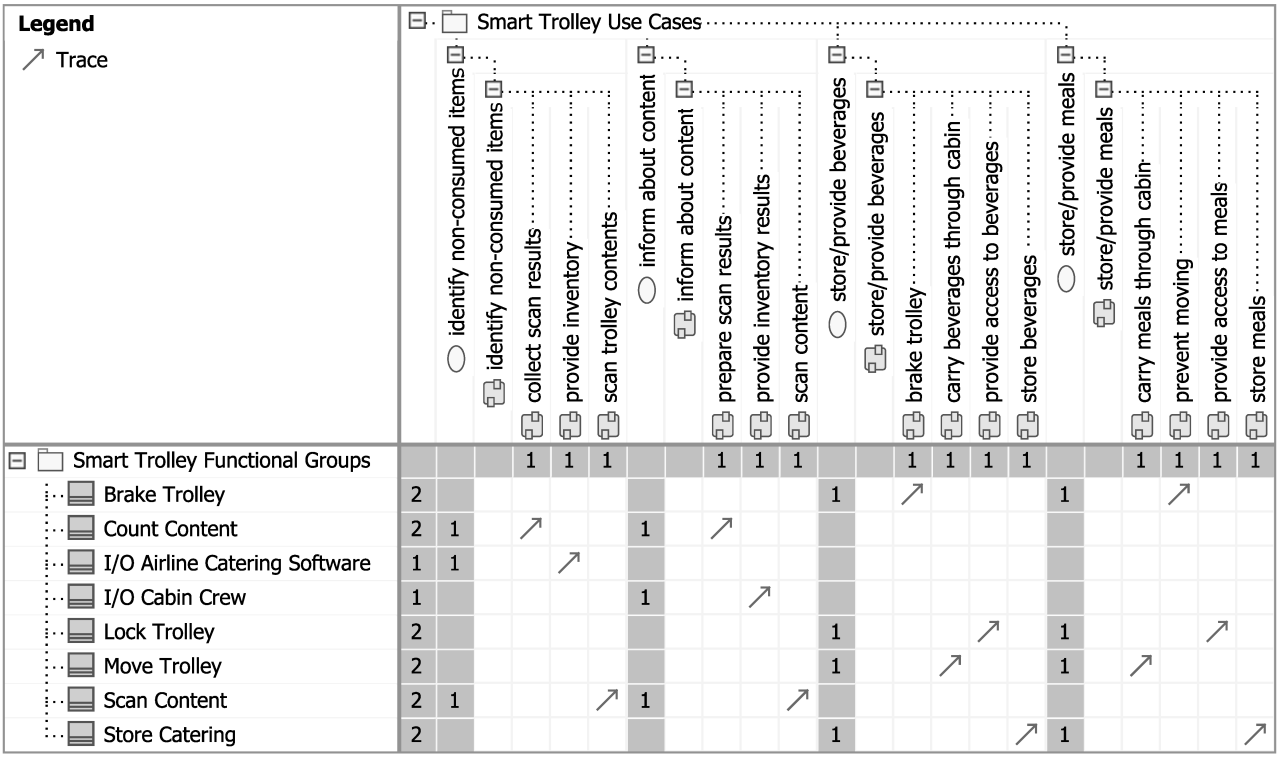


Figure 4.26: Functional grouping in a dependency matrix of Cameo Systems Modeler

Criteria for systematic functional grouping are presented in [87] and include:

- A functional group may take the functions that are relevant to a certain system actor.
- A cluster of functions that call each other may be a candidate for a functional group.
- Activities that share flows of information, energy, and material may be in the same functional group.
- System decomposition from previous development cycles may be a hint for functional groups. However, these groups can contain solution-dependent technical concepts and can therefore constrain the possible solution space for the described functions.

Using the criteria described above, activities from the smart trolley use cases are allocated to functional groups that represent “a set of strongly-related use case activities” [87]. For each functional group a functional element is created with the same name in order to obtain a functional architecture. A functional element is an “abstract system component that defines a relation between at least one input and at least one output by means of a function” [87]. Functional elements are modeled as functional blocks. Figure 4.27 shows the functional blocks of the smart trolley that are derived from the functional groups depicted in the functional groups matrix in Figure 4.26. The functional blocks are part of a “Functional CPS Architecture” block, as indicated by a composition relationship between the architecture block and the functional blocks.

Connections for the exchange of information, energy, and material between functional blocks are modeled in a functional CPS architecture using an internal block diagram. The functional

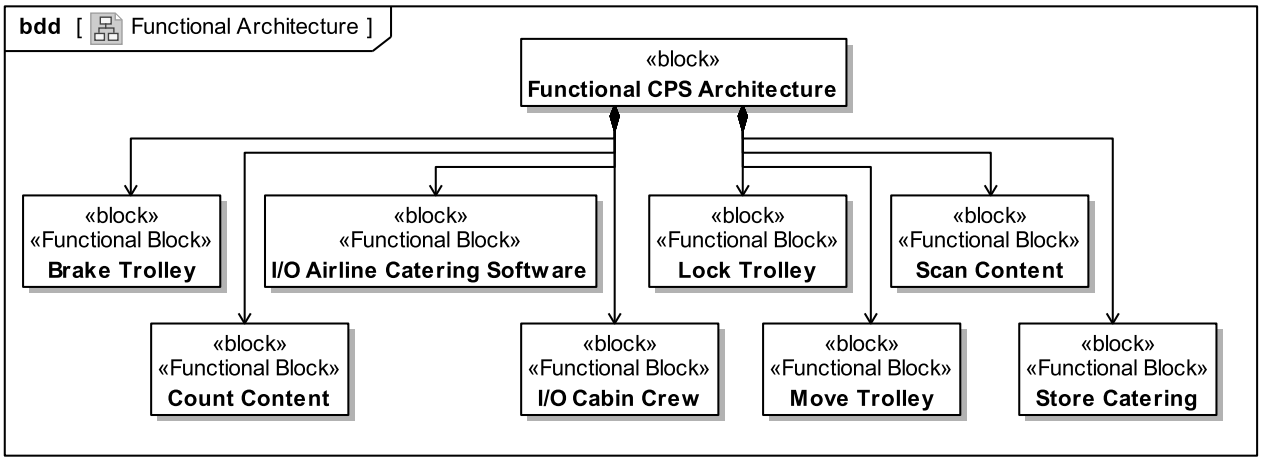


Figure 4.27: Functional blocks of the smart trolley, modeled in a SysML block definition diagram

architecture shows part properties that are typed by the previously defined functional blocks. These part properties are connected by means of proxy ports and connectors. Figure 4.28 shows a segment of the functional architecture for the smart trolley that considers the functional blocks “I/O Cabin Crew,” “Scan Content,” “Count Content,” and “I/O Airline Catering Software.”

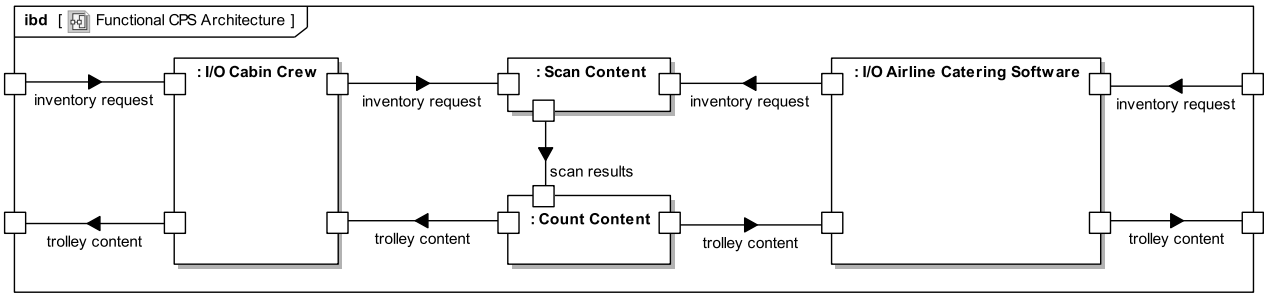


Figure 4.28: Segment of the functional CPS architecture of the smart trolley, modeled in a SysML internal block diagram

Connections between functional blocks, modeled as proxy ports and connectors, are derived from object flows and action pins in use case activities (*cf.* Figure 4.25 on page 111). These action pins are further specified by means of domain blocks, *e.g.* “scan results” that typifies the pins of “: scan trolley contents” and “: collect scan results” call behavior actions in the use case activity diagram in Figure 4.25. The same domain blocks are the basis for ports, connectors, and interface specifications for the functional architecture. As an example, the “scan results” domain block is the origin of the connector between the functional blocks “Scan Content” and “Count Content” and the item flow “scan results,” shown in the functional CPS architecture of the smart trolley in Figure 4.28.

Developing and modeling functional architectures contains some steps that offer potential

for automation, *e.g.* automated use case activity diagram initialization and creation of functional blocks based on functional groups. Automation potential is demonstrated by the example of the modeling tools MagicDraw and Cameo Systems Modeler by No Magic / Dassault Systèmes by the FAS plugin that supports CPS-FAS application with the following functions:

- Initialization of activity diagrams: After modeling cps-human and cps-machine use cases, the plugin creates activity diagrams for each use case. The plugin creates partitions for the allocation of activities to the CPS under consideration and its interfaces to actors and other CPSs, as shown in Figure 4.25 for the “identify non-consumed items” use case activity with an interface to the airline catering software. A partition with I/O stereotype is created automatically for each actor or CPS that is connected to the related use case.
- Initialization of functional groups: The plugin creates a functional group for each I/O partition from use case activities.
- Creation of functional blocks: Functional groups are added to the model by the system architect during functional grouping (*cf.* Figure 4.26 on page 112). The plugin creates a functional block with the same name for each functional group.
- Creation of functional interfaces: Functional interfaces between functional blocks are modeled with interface blocks and connectors. The plugin creates these interfaces by considering object flows in use case activities. If there is an object flow between activities that are traced to different functional groups, the plugin creates a functional interface between the functional blocks that are based on these functional groups.
- Creation of a functional system context: The functional system context shows the interfaces of the CPS under consideration to actors and other CPSs in an internal block diagram. CPSs and actors are modeled as part properties with proxy ports and connectors. The plugin collects all interfaces to actors and other CPSs that are connected to cps-human and cps-machine use cases and automatically creates the functional system context.
- Creation of an internal block diagram for functional architecture representation: The plugin creates an internal block diagram that contains part properties, proxy ports, and connectors for functional architecture modeling. The part properties are typed by the previously created functional blocks and connectors are based on the functional interfaces.
- Creation of a behavior view: The plugin adds functional blocks to partitions in activity diagrams and allocates use case activities to the functional blocks for presenting their joint behavior.

The method for CPS function identification and functional architecture development supports functional CPS architecture development. It includes functions that are demanded by initial stakeholders of the system and functions that are required by the CPSoS. The analysis in the functional grouping process step considers initial and additional CPSoS activities

equally so that the set of functional groups obtained takes into account initial and CPSoS needs. Further development of the CPS can be conducted according to established MBSE approaches, *e.g.* SYSMOD [109] or OOSEM [69].

4.5 CPSoS Communication Architecture Implementation

As soon as the allocation of functions to CPSs is determined in the CPSoS architecture evaluation method, the communication paths between CPSs are designed in order to enable a contract-based communication between CPSs. The method CPSoS Communication Architecture Implementation supports the design of a communication architecture, its integration, and demonstration by using an environmental model. Figure 4.29 shows the associated process.

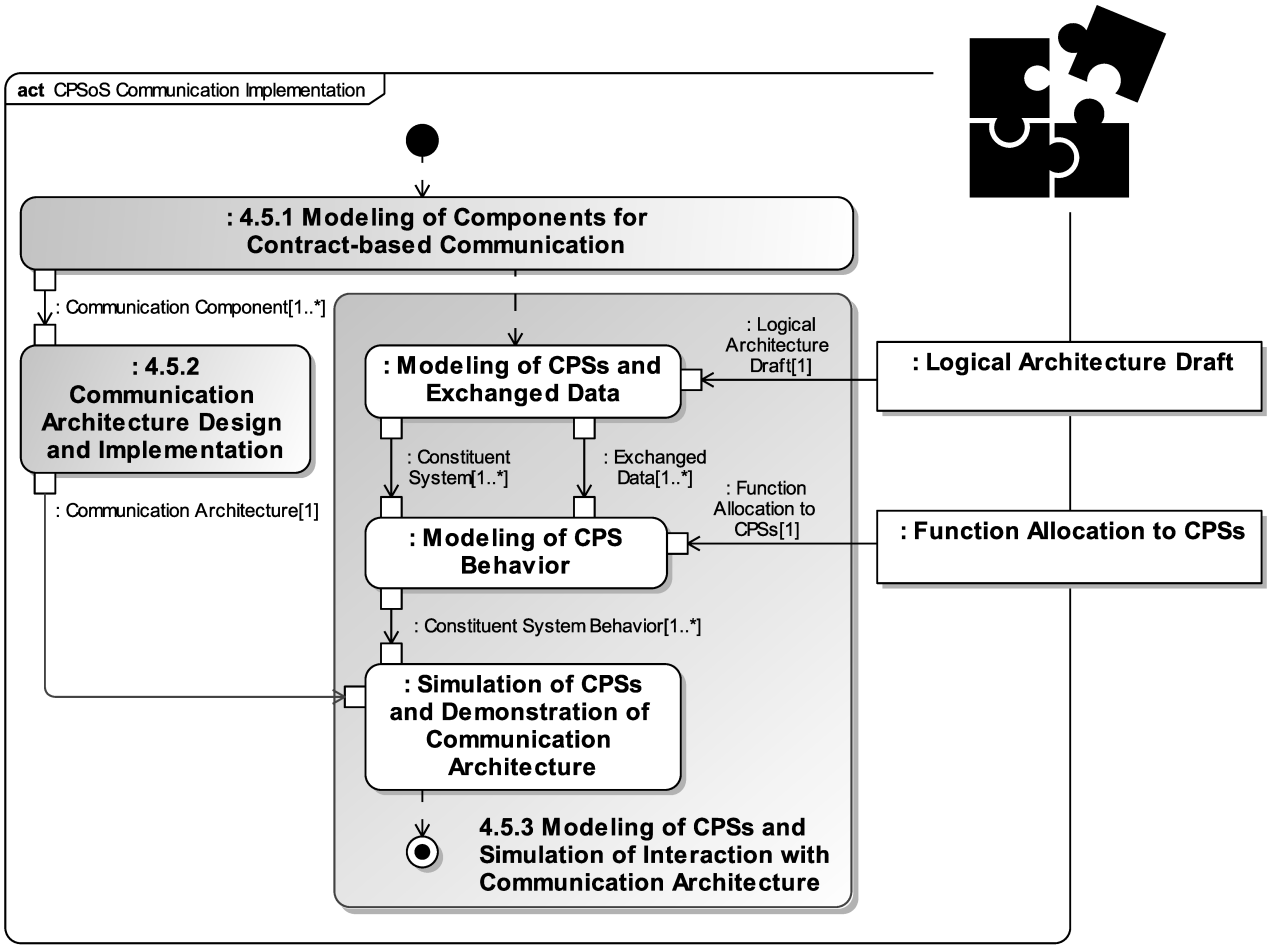


Figure 4.29: SysML activity diagram of the process for CPSoS Communication Architecture Implementation

In the first step of the process, components for contract-based communication are modeled. These components are used for the design of the communication architecture in the subsequent process step. Taking the example of AMQP communication using the message broker

RabbitMQ (*cf.* Section 3.4 on page 78), the concept with brokers, exchanges, queues, etc. is modeled in the first process step. In the design step, these components are used for designing the communication architecture that is implemented in the third process step. Parallel to communication architecture design and implementation, CPSs and exchanged data are modeled. Therefore, CPS blocks from the selected logical architecture draft (*cf.* Section 4.3 on page 103) are used and enhanced, *e.g.* by adding value properties to the blocks. Based on the function allocation to CPSs the behavior of CPSs is modeled, in particular with respect to its communication behavior that affects the CPSoS communication architecture. Finally, relevant CPSs are simulated and the implemented CPSoS communication architecture can be demonstrated.

4.5.1 Modeling of Components for Contract-based Communication

Different concepts for implementation of communication paths between CPSs exist. As described in Section 3.4 on page 78, demands for contract-based data exchange can be met by AMQP. Alternatives for communication architecture implementation are, *e.g.* Apache Kafka and Application Programming Interfaces (API). For the intelligent catering CPSoS, broker communication using AMQP and RabbitMQ brokers is chosen because of the rich possibilities for configuration of exchanged data and simple expansion options for the communication architecture. In the following, modeling of elements for the design of a communication architecture is described with the example AMQP in combination with RabbitMQ brokers. First, elements and their relationships are modeled as blocks and associations in a block definition diagram as shown in Figure 4.30.

Elements for message broker communication and their properties are described by means of SysML blocks and value properties. In the enlarged segment in Figure 4.30, a *Message Broker* block is modeled. Values for the port, login details, host, and virtual host of the message broker are modeled as value properties with default values after the equals sign, *e.g.* “localhost” as the default value for the host value. Associations between blocks represent relationships between elements for message broker communication. Thus, the relationship between message brokers according to the broker federation concept is modeled as an association with the name *Upstream* in the form of a solid line with an arrow end. Associations are described in more detail by association blocks with reference and value properties. The association block is connected to the solid line by means of a dashed line.

Reference properties in the references compartment describe the association ends, *i.e.* the blocks that are connected by the considered association, in more detail. In the case of the *Upstream* association block, reference properties are “upstreamBroker” and “downstreamBroker.” Value properties of the *Upstream* association block are “UpstreamName,” “UpstreamExpiry,” and “URIdownstreamBroker.” Since Upstreams are part of message broker

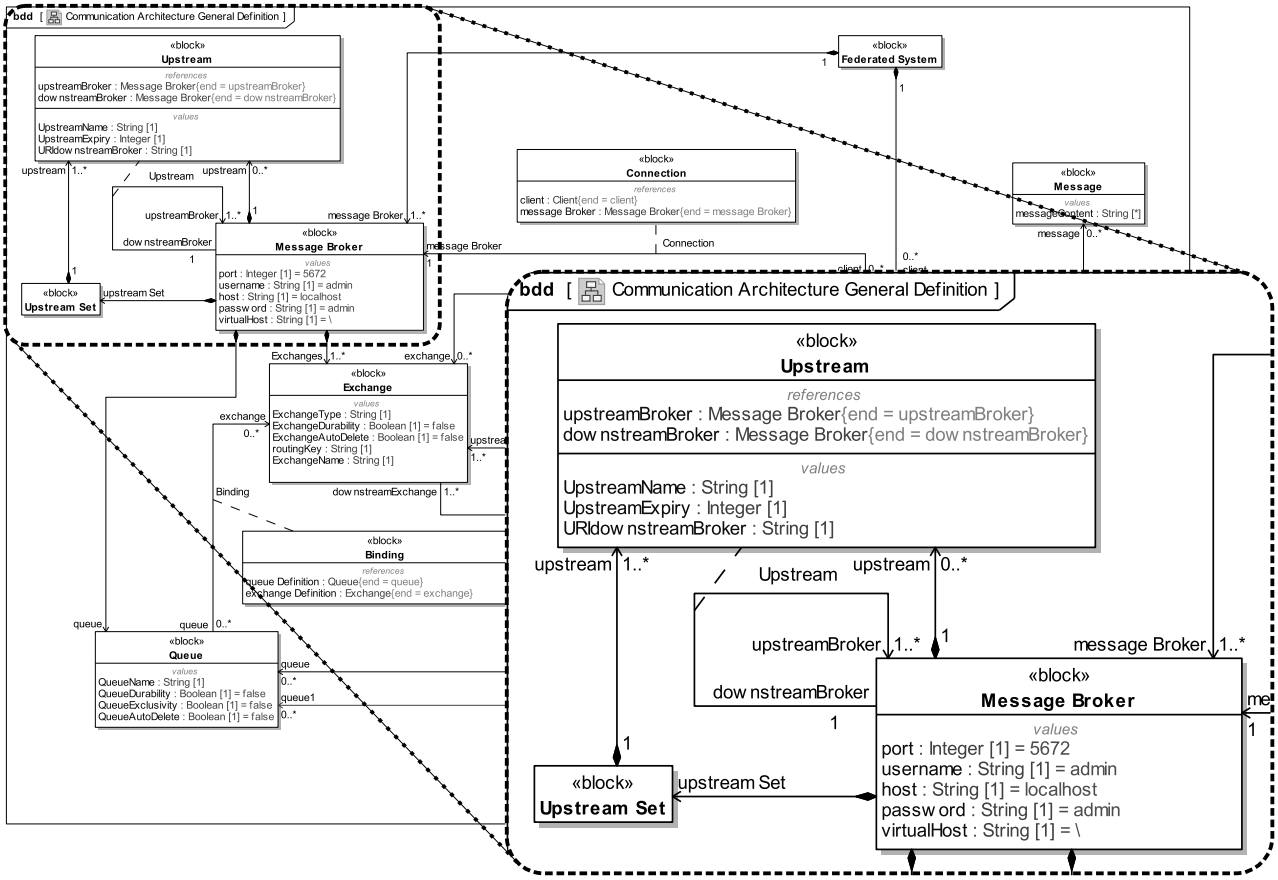


Figure 4.30: Components for communication architecture design using message broker communication, modeled in a SysML block definition diagram

definition, they are connected to the “Message Broker” block by means of a composition relationship. More components for contract-based communication using AMQP and RabbitMQ are:

- Federated System: The Federated System block represents the whole communication architecture and contains message brokers and their clients.
- Client: Clients can send and receive messages. They can publish to exchanges and queues and can receive messages from queues.
- Message: Messages are sent by clients and have a message content, modeled as value property.
- Exchange: Exchanges distribute incoming messages depending on the exchange type, modeled as value property “ExchangeType.” Durability of exchanges is modeled in “ExchangeDurability” and “ExchangeAutoDelete” value properties. More properties of exchanges are their routing key (“routingKey”) and name (“ExchangeName”).
- Queue: Queues are described by their name (“QueueName”) and information about durability by the properties “QueueDurability” and “QueueAutoDelete.” The “QueueExclusivity” value property describes whether a queue is only available for one connection and deleted when this connection is closed.

Connections and relationships between components for AMQP-based communication are modeled by means of association blocks:

- Connection: Connections are the basis for communication between message brokers and clients. Clients can publish messages to exchanges and queues and subscribe to queues only if a connection is established.
- Sending: The sending relationship expresses that clients can send messages.
- Exchange Publication: The exchange publication pattern is chosen if a client shall publish messages to an exchange. An exchange distributes messages depending on the exchange type and routing key.
- Binding: Queues are bound to exchanges so that they can receive messages from the respective exchange.
- Subscription: Clients subscribe to queues so that they can receive messages that are sent to the queue.
- Queue Publication: Queue publication is chosen if a client shall publish directly to a queue and skips an exchange for message distribution.
- Policy: Policies are used for broker federation. When an upstream between two brokers is defined and running, policies match exchanges and queues from the upstream to the downstream broker. Each policy for broker federation has a pattern, modeled as value property. The pattern determines the names of matched exchanges and queues.

Modeling of components for communication architecture design is shown illustratively for AMQP and the message broker RabbitMQ. The same concept can be applied to other communication concepts, *e.g.* Apache Kafka or APIs, and combinations of different modeling concepts. The following section presents how blocks and associations from component modeling can be used for the design of a CPSoS communication architecture.

4.5.2 Communication Architecture Design and Implementation

After modeling required components of the selected communication concept, a specific communication architecture is designed. Details about communication architecture design with blocks and associations from components modeling according to the previous Section 4.5.1 is presented in this section using the example of the intelligent catering CPSoS.

Logical architecture drafts for the intelligent catering CPSoS are evaluated in Section 4.3 on page 103. The first logical CPSoS architecture draft (*cf.* Figure 4.17 on page 103) allocates the function *Identification of Non-Consumed Items* to the aircraft. This function allocation results in a better evaluation of inventory accuracy compared to the second logical CPSoS architecture draft that allocates the same function to a catering inventory system on the ground (*cf.* Figure 4.18 on page 103). The first draft is judged to be the most suitable logical CPSoS architecture and is chosen for further development. Figure 4.31 on page 120 shows

a part of the associated communication architecture. In summary, smart trolleys identify and count the meals and beverages contained and send inventory data to their respective aircraft broker. An airline broker receives catering inventory messages from the whole aircraft fleet using queue federation. The airline resource planning system subscribes to the airline broker and receives the catering inventory messages.

The communication architecture draft is modeled with instance specifications and links. Instance specifications are classified by one of the blocks from the general definition of the communication architecture (*cf.* Figure 4.30 on page 117). Links are classified by association blocks from the general definition of the communication architecture and connect instance specifications. The enlarged segment in Figure 4.31 on page 120 shows message brokers of two aircraft, named “aircraft 1 broker” and “aircraft 2 broker” that are classified by the “Message Broker” block from the general communication architecture definition. Values for value properties of the “Message Broker” block are entered into so-called slot values of the instance specifications, shown in the upper box of the “aircraft 1 broker” instance specification for the host, port, virtual host, and login credentials of the message broker. The lower box of the message broker instance specification shows the elements contained. In the case of the aircraft 1 broker instance specification, a message queue instance specification is created. The instance specification is named “aircraft 1 queue” and has a slot value for the value property “QueueName.” An additional value property for the queue name is required for subsequent simulation of CPS interaction as described in Section 4.5.3.

For transmission of catering inventory data from aircraft to the airline, an upstream between aircraft and airline broker is provided. A link between the instance specifications “aircraft 1 broker” and “airline broker” represents this upstream. In addition to the upstream, a queue policy is defined that matches queues whose names begin with the string “inflight-supplies.” The queue policy is modeled as a link between queues of the aircraft 1 broker and airline broker. The same communication pattern is modeled for a second aircraft broker with the name “aircraft 2 broker,” shown in the lower part of the enlarged segment in Figure 4.31.

The airline resource planning tool is modeled as an instance specification with the classifier “Client.” It subscribes to the “inflightsuppliesCollection” queue of the airline broker in order to receive all inventory messages from the aircraft fleet. Smart Trolleys that are able to detect their content automatically (*cf.* Section 4.4 on page 108) are shown on the left-hand side of the unenlarged segment in Figure 4.31. They are modeled as instance specifications with the classifier “Smart Trolley.”

The simulation of CPS interaction is described in the following Section 4.5.3. For simulation purposes, only one instance specification can be set as the execution target. Since all modeled message brokers and their clients are to be considered by the simulation, they must be assigned as slot values to a single, superordinate instance specification that is

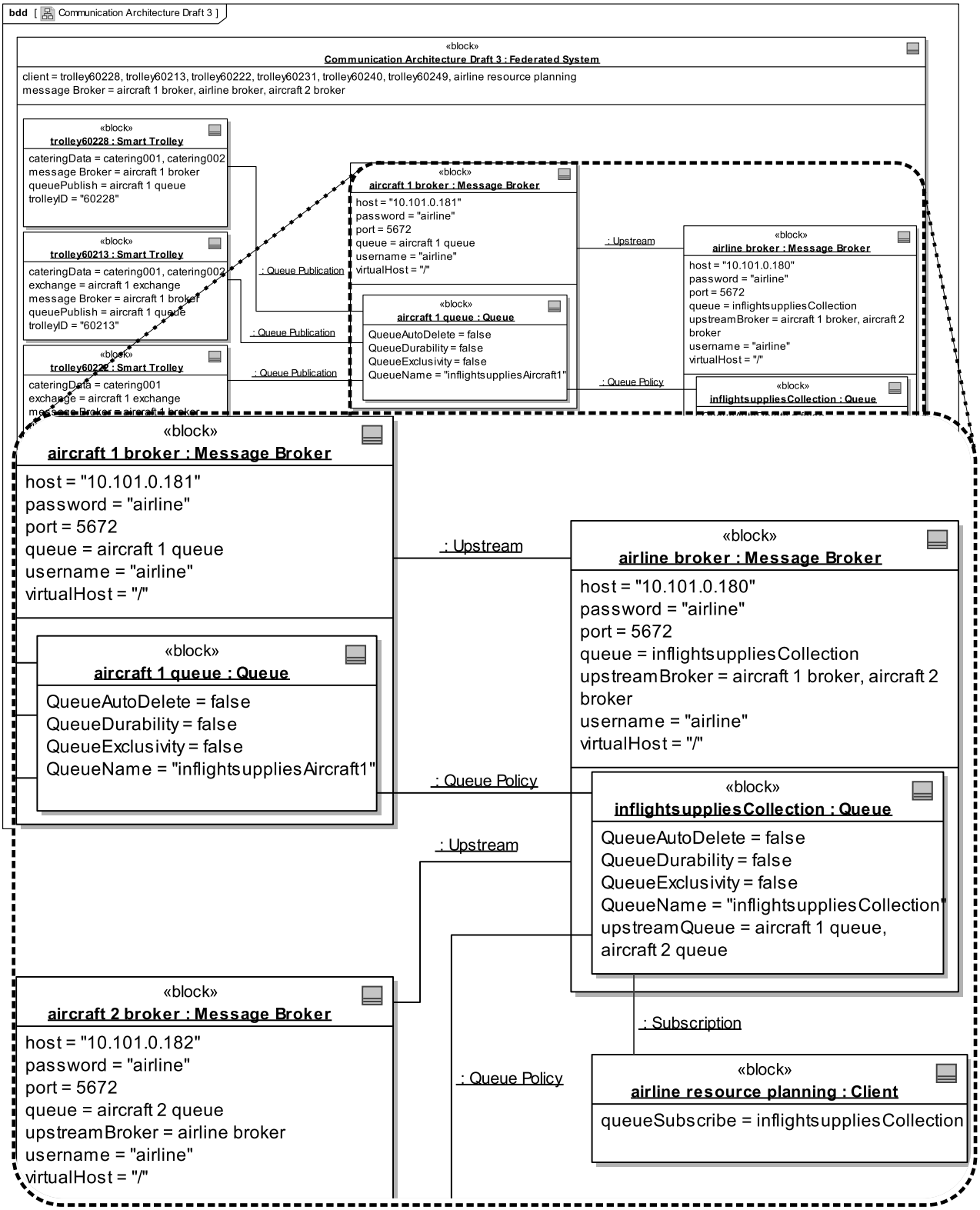


Figure 4.31: Communication architecture draft, modeled in a SysML block definition diagram

set as the execution target of the simulation. This superordinate instance specification is classified by the “Federated System” block from the general definition of the communication architecture (*cf.* Figure 4.30 on page 117).

In summary, the CPSoS communication architecture is represented by an instance specification with the classifier “Federated System.” The federated system instance specification contains message broker and client instance specifications that are connected to each other via links. In the following, the communication architecture can be implemented using the principle of virtualization, real CPSs in a test environment, or a combination of both. Before implementation of the CPSoS communication architecture using real CPSs, it is recommended that its function be demonstrated by using virtualization. For virtualization of the communication architecture, components of the architecture are set up on virtual machines or in containers. After function demonstration using a simulation, the communication architecture can be implemented on real CPSs.

4.5.3 Modeling of CPSs and Simulation of Interaction with Communication Architecture

A CPSoS communication architecture based on broker federation with various configuration options cannot be implemented directly with all relevant CPSs. CPSs may be owned by different organizations and the implementation may require unjustifiable effort. As an example, it cannot be expected that multiple smart trolleys, aircraft galleys, and aircraft communication systems will be upgraded in order to test the designed CPSoS communication architecture. For this reason, this section presents how the designed and implemented communication architecture can be demonstrated by using an environmental model that simulates the behavior of CPSs. This environmental model can either interact with the virtualized communication architecture on virtual machines and in containers or, in subsequent development steps, with real CPSs.

In the following, the approach is demonstrated by simulation of smart trolley behavior in an environmental model. The smart trolley is modeled as a block with according behavior, as shown in Figure 4.32. The “Smart Trolley” block is a specialization of the “Client” block. Value properties and behaviors are displayed in compartments. Each smart trolley has an ID, modeled as a value property with the name “trolleyID.” The desired smart trolley behavior is modeled in activities and opaque behaviors, shown in the “classifier behavior” and “owned behavior” compartments of the “Smart Trolley” block. Exchanged data in the form of a detected inventory of trolleys is modeled as “Catering Data” block, connected to the “Smart Trolley” block by means of an association. The “Catering Data” block has value properties for the flight number, flight time, identification time, and the amount of meals and beverages.

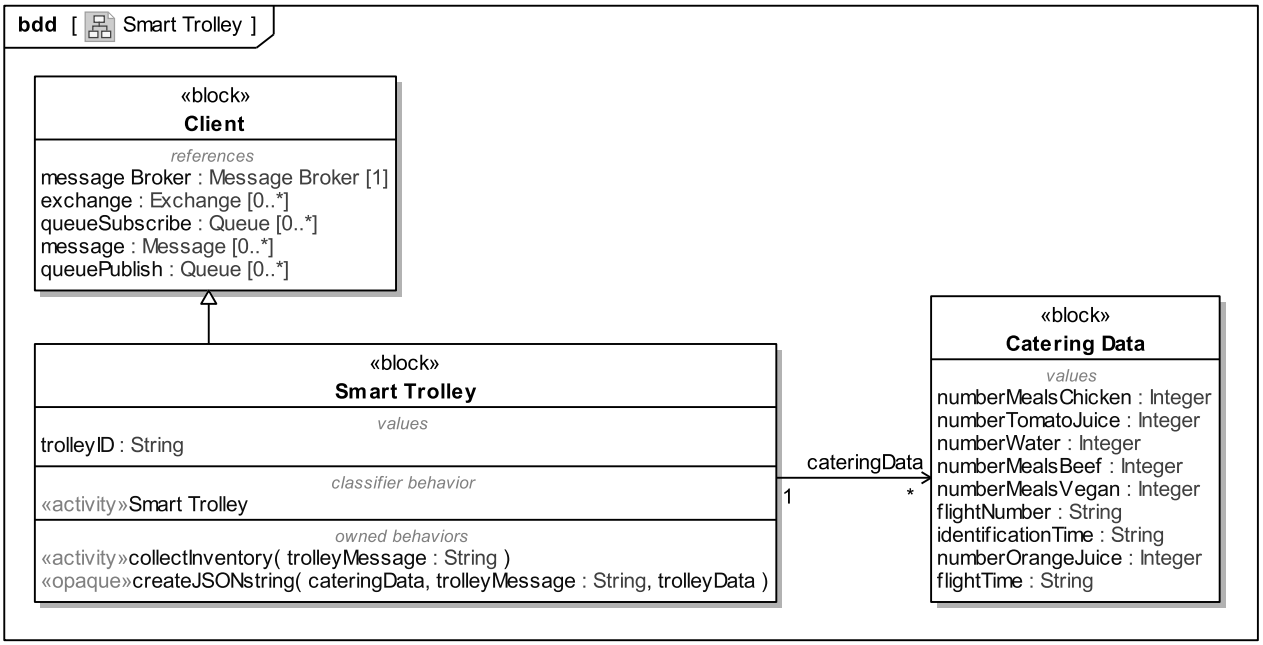


Figure 4.32: Smart trolley definition in a SysML block definition diagram

Next, CPS behavior is modeled. Depending on the chosen communication concept, *i.e.* AMQP-based broker communication, utilization of Apache Kafka, or APIs, actions for interaction with the elements of the communication concept are required. Therefore, opaque behaviors must be modeled that allow communication with elements of the respective communication concept. These opaque behaviors are collected in a library. For simulating the communication of a CPS with the communication architecture, these opaque behaviors are called by call behavior actions in activities that describe the behavior of the simulated CPS. In the example of AMQP-based broker communication, actions for sending and receiving messages to and from queues are needed. Melzer et al. [92] developed the broker-based SysML toolbox for communication with RabbitMQ brokers. The broker-based SysML toolbox includes a library of opaque behaviors in the form of Beanshell scripts for the messaging patterns of RabbitMQ.

Usage of opaque behaviors for simulation of CPS interaction with the communication architecture is demonstrated by means of the smart trolley example. The smart trolley behavior is displayed in the “classifier behavior” and “owned behaviors” compartments of the CPS blocks in Figure 4.32. In the case of the smart trolley, there is a classifier behavior denoted as “Smart Trolley” and owned behaviors with the names “collectInventory” and “createJSONstring.” The classifier behavior is executed automatically when an instance of the owning block is created. Owned behaviors must be called, *e.g.* by a call behavior action or a call operation action. Figure 4.33 shows the classifier behavior “Smart Trolley” on the left-hand side and the owned behavior “collectInventory” on the right-hand side.

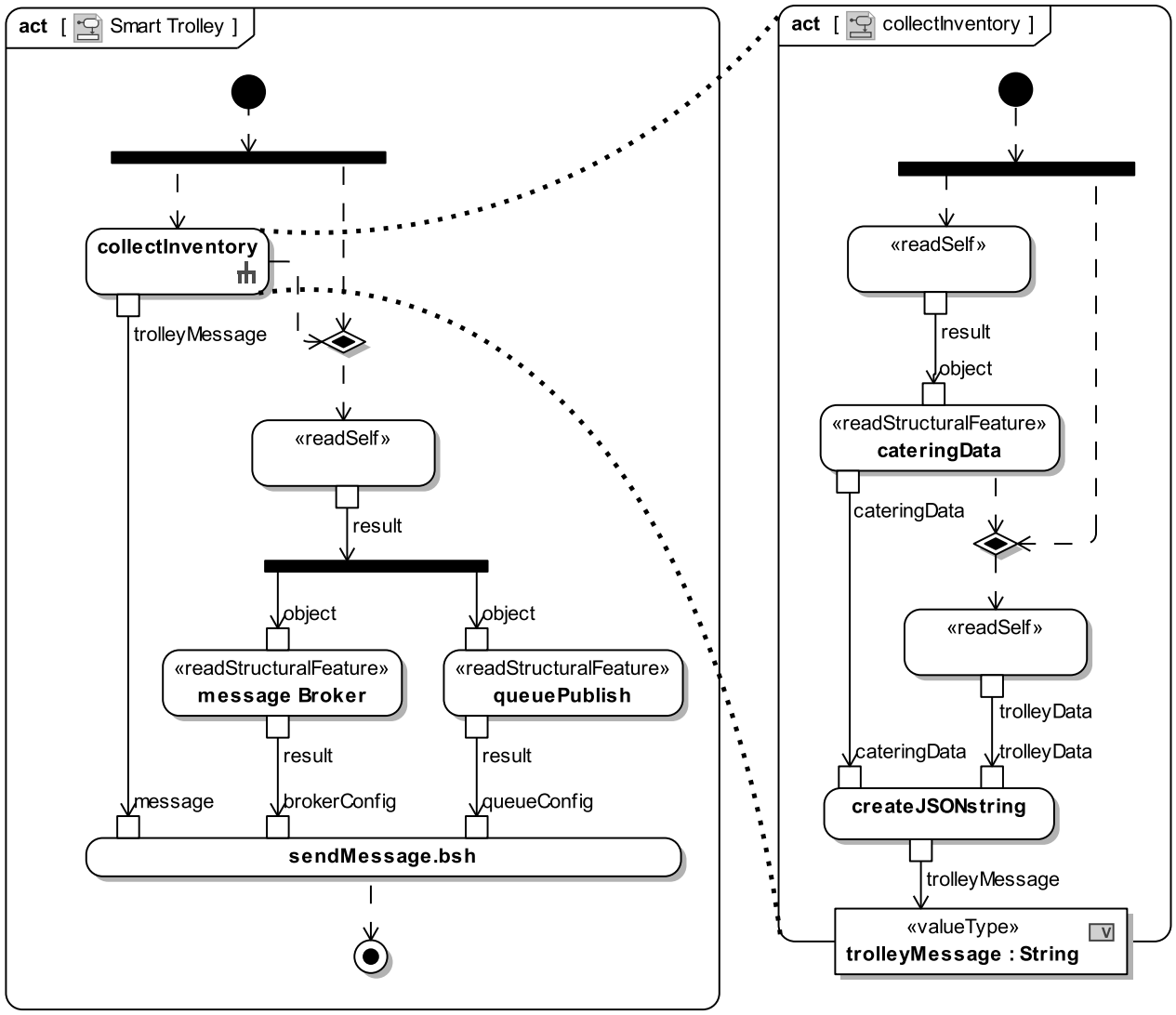


Figure 4.33: Simulation of smart trolley behavior using a SysML activity diagram

At the bottom of the “Smart Trolley” activity, a call behavior action calls the opaque behavior “sendMessage.bsh” from the broker-based SysML toolbox [92] for sending messages to a queue of a RabbitMQ broker using a Beanshell script. The call behavior action has three input pins for providing the message, broker configuration, and queue configuration as input parameters for the Beanshell script. The input parameters for the broker and queue configuration are read from the instance specifications typed by the “Smart Trolley” block. The message parameter depends on the simulation needs. It can be read from a structural feature of an instance specification, determined in an activity, or inserted by an actor via a graphical user interface (GUI). In the case of the smart trolley, a call behavior action calls the “collectInventory” activity, displayed on the right-hand side of Figure 4.33. The “collectInventory” activity simulates the smart trolley behavior with respect to identification of meals and beverages contained. It reads values from instance specifications with the classifiers “Smart Trolley” and “Catering Data” (*cf.* Figure 4.32) and creates a json string. This

json string is sent by the call behavior action with the opaque behavior “sendMessage.bsh” to the message broker.

Figure 4.34 provides an overview of the elements of the CPSoS design and integration method. The method starts with modeling of blocks that represent components for contract-based communication in CPSoS (left-hand side of Figure 4.34). The modeled components are used for the design of a specific communication architecture using instance specifications with specific parameters (right-hand side of Figure 4.34). Next, the communication architecture can be implemented in a virtualized environment or by using real CPS. Then, parameters of the communication architecture are used for simulation of CPS behavior in order to demonstrate the designed communication architecture (middle of Figure 4.34).

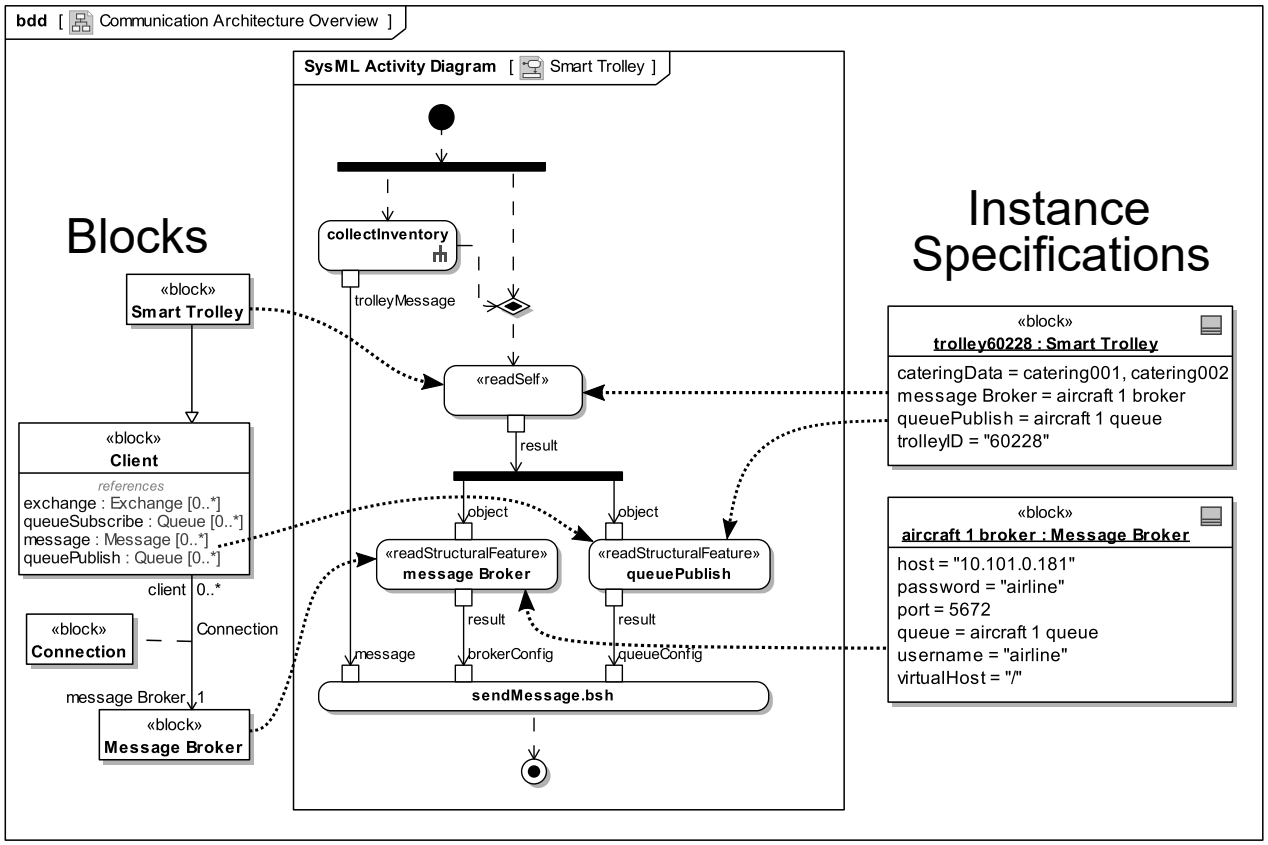


Figure 4.34: Overview of communication architecture design and integration, modeled in SysML block definition diagrams and SysML activity diagrams

4.6 Methodology for the Design of Cyber-physical Systems of Systems

In the following, the methods presented for the design of CPSoS are summarized. Furthermore, it is described how the methods contribute to room for improvement, identified in

Chapter 3 on page 39, for the concept, function, architecture, and design life cycle phases of a system.

As already introduced in Section 2.1 on page 15 about system characteristics, CPSoS are characterized by evolution, *i.e.* CPSoS tasks are modified or new missions are to be accomplished, CPSs leave or join the CPSoS. To address the dynamic character of CPSoS, the whole methodology is model-based and supports changes with traceability analyses in relation maps as well as dependency matrices. Changes to the CPSoS can be modeled as new missions. Then, additional operational scenarios and use cases are added to the model. The functional analysis and grouping allows reusing existing CPSoS functions so that new missions can be taken over by the CPSoS with as little effort as possible. Similarly, CPS functions can be reused and new required CPS functions are added to the functional CPS architectures.

Concerning the aspects mentioned in the room for improvement of existing approaches for the concept phase in Section 3.1 on page 40, CPSoS-characteristic intensive communication with multiple interfaces is considered. The CPSoS Mission and Operational Scenarios Definition method starts with the elaboration of a mission, operational scenarios, and CPSoS use cases with the interaction of multiple CPSs. Following methods support the design of a logical CPSoS architecture with specifiable communication paths between CPSs to the implementation of a CPSoS communication architecture. In addition to the CPSoS, the CPSs included in the CPSoS are also considered in the development of CPSoS through the CPS Function Identification and Functional Architecture Development method. Since the CPSoS is partly based on existing CPSs, the context definition and the identification of the information network structure include existing interfaces of the CPSs for further development.

In Section 3.2 on page 56, room for improvement of existing approaches with respect to the function life cycle phase is summarized. One aspect for improvement is the differentiation between CPSoS and CPS functions. The methodology proposed in this thesis differentiates between functions at the CPSoS level, identified by the CPSoS Function Identification and Architecture Development method, and CPS functions in the CPS Function Identification and Functional Architecture Development method. Meanwhile, both methods support the identification of redundant functions through functional grouping of use case activities in dependency matrices to avoid unnecessarily extensive architectures. Once the required functions are determined, the methodology supports the systematic derivation of architectures and the detailed specification of the interfaces by means of proxy ports.

None of the approaches analyzed in Section 3.3 on page 70 considers the allocation of functions to CPSs in a CPSoS. The methodology proposed in this thesis addresses this issue with logical CPSoS architectures in that CPSoS functions are allocated to CPSs in the form of logical elements. Interfaces can also be specified in logical CPSoS architectures.

As multiple options for the allocation of CPSoS functions to CPSs may exist, multiple logical CPSoS architectures may emerge. The CPSoS Architecture Evaluation method supports the evaluation of logical CPSoS architectures and the determination of the most suitable allocation of functions to CPSs. Finally, the *CPSoS Communication Implementation* method enables the design of a CPSoS communication architecture that can be used for the implementation using various communication concepts, *e.g.* message brokers or RestAPIs.

In conclusion, the proposed methodology for the design of CPSoS meets the research questions introduced in Section 2.3 on page 34 and the needs that are summarized as room for improvement in CPSoS development in the Sections 3.1, 3.2, and 3.3. Figure 4.35 summarizes the contribution of the methods to the life cycle phases of CPSoS.

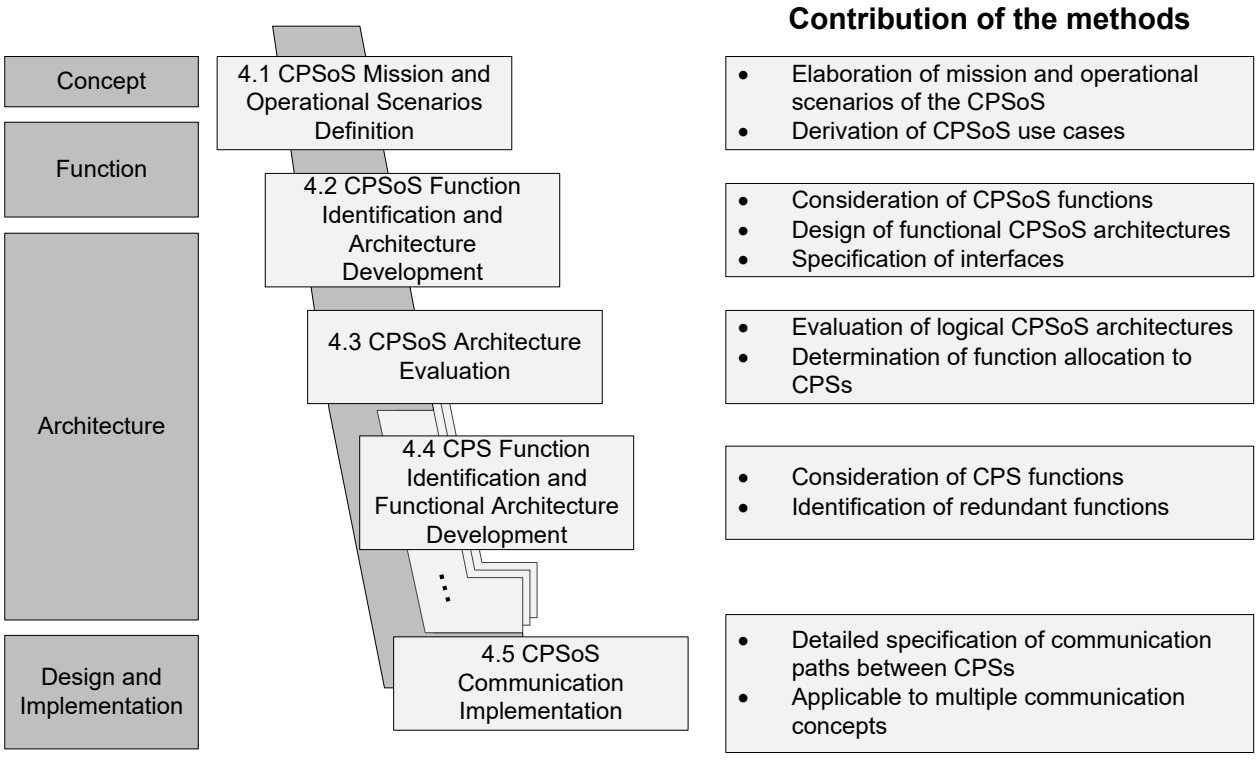


Figure 4.35: Contribution of the methods to the design of CPSoS

The whole methodology is applied and evaluated in more detail by means of application in three CPSoS development projects, as described in the following Chapter 5.

5 Application and Validation

The methodology presented in Chapter 4 was developed based on solutions for tasks from various CPSoS research projects. A solution or method that could be implemented in practice and that could remedy the previous shortcomings of existing development approaches was finally positively validated and integrated into the methodology. Three projects that were selected for validation of the proposed methods included systems that had been operated autonomously and independently from each other by different organizations such as airlines, airports, and catering providers, corresponding to the *autonomy*, *belonging*, and *diversity* attributes of SoS. Stakeholders involved had individual targets and expectations towards the operation of the CPSoS so that a large number of heterogeneous stakeholder expectations had to be considered in the design of the CPSoS. The expectations of the stakeholders in the projects thus related to typical CPSoS examples as opposed to the development of ordinary or simple systems.

Accordingly, a large number of heterogeneous functions was a further consequence. The large number and variety of functions made it difficult to achieve an overview. Their implementation required diverse technologies, hence multiple experts, *e.g.* for aircraft and airport systems, were involved. An important part of the collaboration of the experts was the exchange of knowledge and the establishment of a common understanding of the design task.

Another consequence of the cooperation of multiple CPSs in a CPSoS was a large number of interfaces between the CPSs, corresponding to the *connectivity* attribute of CPSoS. But not only the large number of interfaces between CPSs was challenging. Also the specification of data that could be shared between CPSs that were operated by different organizations led to a high degree of complexity in the design task. As described by the concept of contract-based communication (*cf.* Section 3.4 on page 78), CPS operators did not want to share all their CPS data with other organizations so that means for detailed specification of data exchange were needed.

The particular benefit of the methods is that they were suitable for the development of acknowledged and collaborative CPSoS, as illustrated in Figure 5.1. Directed CPSoS with a central entity for control and management could be developed according to established Systems Engineering approaches because a central managing organization exists, enabling the whole CPSoS to be designed like a large self-contained system.

Three CPSoS design projects were selected for validation of the methods developed in this thesis. In the following, these projects and the challenges encountered during the work in

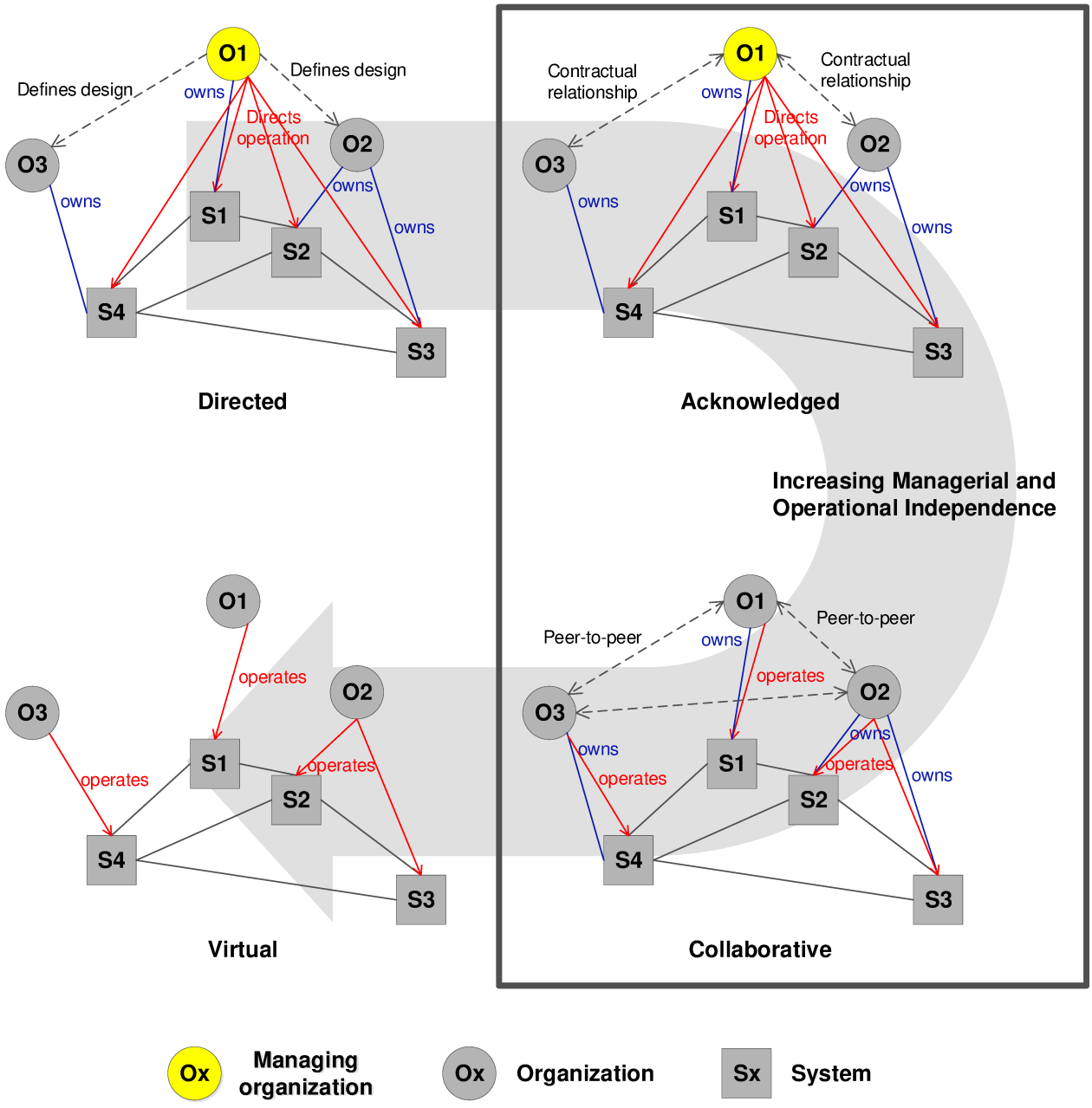


Figure 5.1: The methods proposed in this thesis are applicable to acknowledged and collaborative SoS according to the classification in the ISO/IEC/IEEE standard 21841:2019 [24] and the graphical representation by Ncube and Lim [94]

the projects are described. The CPSoS under consideration are acknowledged and collaborative CPSoS, as the CPSs contained have a higher degree of managerial and operational independence. Virtual CPSoS lack dependencies between operating organizations so that the exchange of data between the CPSs is not determined by the organizations [24, 49, 89]. Uncontrolled exchange of valuable and potentially safety-critical data is not realistic for organizations in the air transport system, so this type of CPSoS is not considered in this thesis. Moreover, the free cooperation of CPSs can lead to unpredictable, emergent CPSoS behavior that is inconsistent with the high safety requirements in the air transport system.

Electronic Passenger Flow Control CPSoS

In the first project, an electronic passenger flow control CPSoS [74] was designed to provide passengers with improved and individualized advice, improve management and control of passenger flows, and increase safety, comfort, and the choice of services. For this purpose, aircraft and other CPSs by airport operators, suppliers of aircraft cabin systems, and providers of airport security systems, analytics software as well as passenger assistance applications for smartphones must cooperate [74].

The CPSs were owned by different organizations such as airports and airlines so that the following challenges were encountered during the design of the CPSoS. First, the design had to consider that the CPSs belonged to different organizations. In addition, aircraft systems, systems for passenger identification at the gate and in other areas of the airport, and passenger smartphones formed a diverse set of CPSs with various interfaces that had to be integrated into the passenger flow control CPSoS. The purpose of the project, passenger flow control and passenger guidance, required a large number of diverse functions, *e.g.* for passenger identification at the gate, other areas of the airport, in the aircraft cabin, and the collection of information about the traffic situation around airports. Consequently, it was a challenge to keep track of the functions and their interfaces. Furthermore, various stakeholders and their needs had to be taken into account. Passengers are identified and guided and authorities' data protection regulations must be observed. Airport operators expected a solution to avoid large accumulations of passengers, and smartphone application providers were interested in a wide distribution of their application in order to generate the highest possible advertising revenue. As a result, various experts for airport processes, software development for airport systems, and aircraft systems were involved and had to develop an understanding of each other [74].

Aviation Industry Supply Chain CPSoS

Increasing connectivity is also affecting the supply chains of the aviation industry. These supply chains are characterized by paper-based information exchange between a large

number of suppliers across multiple levels of the supply chain. Paper-based information exchange is resource-intensive and error-prone because documents are often created manually and the recipient must also manually extract the information. If the enterprises in the aviation supply chain are digitally connected, each enterprise can act as a CPS in an aviation industry CPSoS. The benefits of digital connectivity include increased efficiency, reduced lead times, and reduced risk of delays due to more transparent supply chains, as production status is visible across multiple companies. A possible example is the automated processing of orders in that a customer requests an offer, receives an offer based on the current resource situation at the supplier, and can place an order. Another example is the automated invoicing and payment that could substitute the exchange of mails for sending invoices.

In this project, solutions had to be found for the following challenges. One challenge in the development of an aviation industry CPSoS was the diversity of existing enterprise IT systems. Some suppliers did not use enterprise resource planning (ERP) systems, and other suppliers used different ERP systems, so there was no standardized interface between the systems. Furthermore, IT systems belonged to different companies that wanted to share only selected data with other companies according to the principle of contract-based communication. In addition, the large number of processes and companies involved at multiple supply chain levels resulted in a complex development task. Moreover, even in a single company multiple experts were required for the description of processes because often there was no single person who had an overview of all the company's processes. Thus, the establishment of a common understanding of the processes was a challenge [59].

Data-Driven Catering CPSoS

In the third project, the methods were applied to the design of a data-driven catering CPSoS [96]. The selection of in-flight catering depends on various aspects. Exemplary aspects that influence the demand for in-flight catering are route, time, and number of passengers. Airlines order in-flight catering on the basis of experience from previous flights that are collected in catering reports by cabin crews. Often, these reports are not very accurate because the cabin crew has limited time to report. Usually, catering reports only provide information on underestimated quantities of catering items and do not account for overestimated quantities. Consequently, airlines do not have an overview of actual catering consumption and order catering based on rough empirical values. Regulations require that unconsumed meals be disposed of after flight. Added to inaccurate demand forecasts, this results in a lot of unconsumed catering being disposed of [96].

The CPSoS developed in the third project should reduce catering waste and improve catering demand estimations. During development, the following CPSoS characteristics were encountered by the application of the methodology elaborated in this thesis. The CPSs

being integrated into the CPSoS are operated by airlines and catering providers and fulfill the belonging criteria of CPSoS. The diversity of airlines and catering providers is also represented by their systems so that a diverse set of CPSs has to be integrated into the CPSoS. In addition, catering processes include planning, ordering, production, transportation, storage, loading, serving, and disposal, resulting in a high level of complexity. This complexity is also reflected in the composition of the stakeholders. None of the stakeholders knew all processes so that the documentation of the current processes and the establishment of a common understanding of the processes and the data involved required intensive collaboration. Another important aspect of the CPSoS was the exchange of data. CPSs owned by different organizations must cooperate to generate the desired data about catering consumption. Consequently, there were many discussions about data ownership, requiring appropriate means for specifying CPS interfaces and data exchange. The exchange of data was also dependent on the selected distribution of functions in the CPSoS. For example, a function for evaluating inventory data could be assigned either to a CPS of a catering provider or to a CPS of an airline. Therefore, a way to systematically assign functions to CPSs was also required [96].

The challenges of the previously introduced CPSoS design projects contribute to the application and validation of the methodology elaborated in this thesis. The application and the results of method validation are presented in the following sections.

5.1 Validating CPSoS Mission and Operational Scenarios Definition

The method CPSoS Mission and Operational Scenarios Definition was applied to the development of CPSoS for the improvement of passenger flow control and seamless travel experience and data-driven catering. Figure 5.2 provides a comprehensive overview of the results of the method application.

The mission of the electronic passenger flow control CPSoS can be summarized as reducing processing and waiting times, eliminating congestion, and making the process perceived as smoother. Stakeholder and system contexts were first elaborated based on a literature review about airport and boarding processes. The context drafts were presented to project partners in a web-based graphical collaboration tool and missing systems and stakeholders were added to the contexts. Some project partners already had a concept for the connections between CPSs that were also added to the system context. Figure 5.3 and Figure 5.4 present an extract of the first project partners' ideas. The first concept includes a sensor for passenger identification, a sensor controller, and a computing device that collects signals from all sensors installed in the cabin. The computing device processes the collected sensor signals for identification of passengers in the cabin. Finally, a communication device sends results to other systems inside and outside the aircraft. A passenger welcome screen in

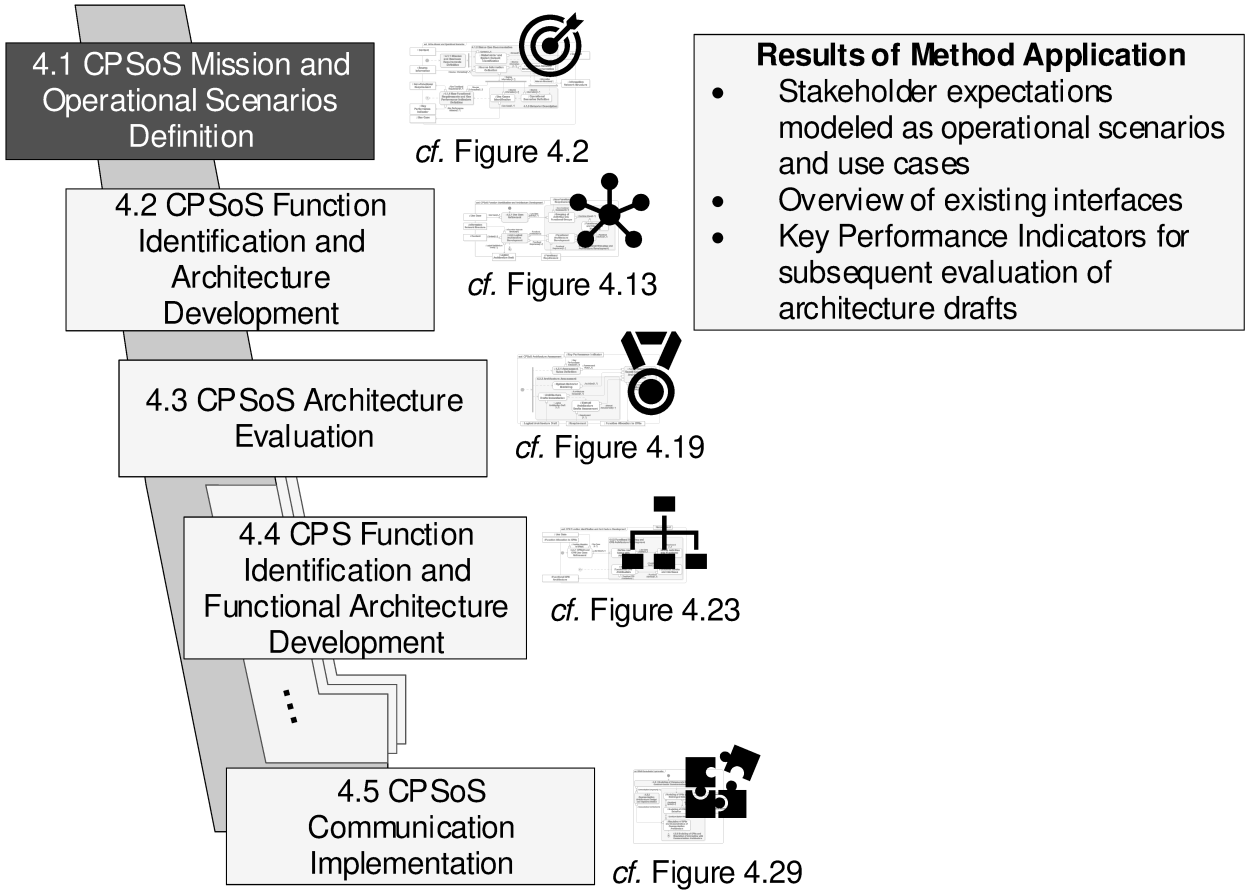


Figure 5.2: Application of the CPSoS Mission and Operational Scenarios Definition method

the aircraft entrance area could control the flow of passengers like a traffic light and is an example of a system inside the aircraft. Gate access control is another system that could be used to control passenger flow and represents a system outside the aircraft cabin. The passenger flow control could prevent congestion in the cabin during boarding and thus improves passenger experience and boarding efficiency.

The initial concepts of the stakeholders revealed stakeholder expectations for the CPSoS to be developed and were documented to be available as inspiration for the design of a logical CPSoS architecture according to the CPSoS Function Identification and Architecture Development method. The concepts also illustrate thinking in terms of solutions. More abstract development elements such as functions are proposed in this thesis and were skipped when sketching the initial ideas. It could be observed that functions were part of the discussions, and multiple suggestions for system design emerged that are shown as examples in Figure 5.3 and Figure 5.4. These suggestions differed in the allocation of functions. As an example, the idea in Figure 5.3 allocates the function for the control of the sensor to an independent sensor controller, while an alternative design, shown in Figure 5.4, allocates the function to a computing device that also evaluates the sensor signals. The joint

consideration of functional and logical elements worked for limited segments of the CPSoS to be developed but became confusing for the complete set of required functions due to the large number and diversity of functions.

Figure 5.3: Initial concept for identifying passengers in the aircraft cabin

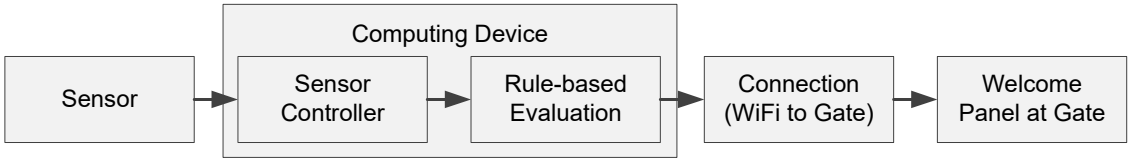


Figure 5.4: Alternative concept for identifying passengers in the aircraft cabin

Consequently, the CPSoS Function Identification and Architecture Development method was applied, allowing all the necessary functions to be identified and architectures to be systematically developed. In preparation for this method, operational scenarios and use cases were defined and modeled according to the CPSoS Mission and Operational Scenarios Definition method. This method suggests the definition of operational scenarios to obtain information about system behavior from an operational perspective. For this purpose, project partners were asked for positive and negative travel experience in a workshop. Based on their experience and airport operators' knowledge, operational scenarios were modeled and insights into passenger processes were derived. Afterwards, a brainstorming session on improvement measures and their benefits was initiated. These improvement measures were formulated as use cases. These use cases formed the basis for the identification of functions and the CPSoS architecture development, as described in the following Section 5.2.

Improved catering demand predictions, reduction of catering waste as well as collection and exploitation of data for new services describe the mission of the data-driven catering CPSoS. General understanding of all processes related to catering was obtained from a literature review on in-flight catering. Open questions were clarified in a workshop with representatives from airlines and caterers so that the operational scenarios of the data-driven catering CPSoS could be modeled. For detailing of the mission, potential for improvement of the catering process, denoted as pain points, was collected and prioritized in the same workshop. These pain points were modeled, marked with a pain points stereotype, and were the basis for the identification of business requirements and use cases for the data-driven catering CPSoS. Project partners showed great interest in the traceability opportunities of the model. Pain points, business requirements, and use cases could be traced with little effort, as shown in the relation map in Figure 5.5 and in additional dependency matrices.

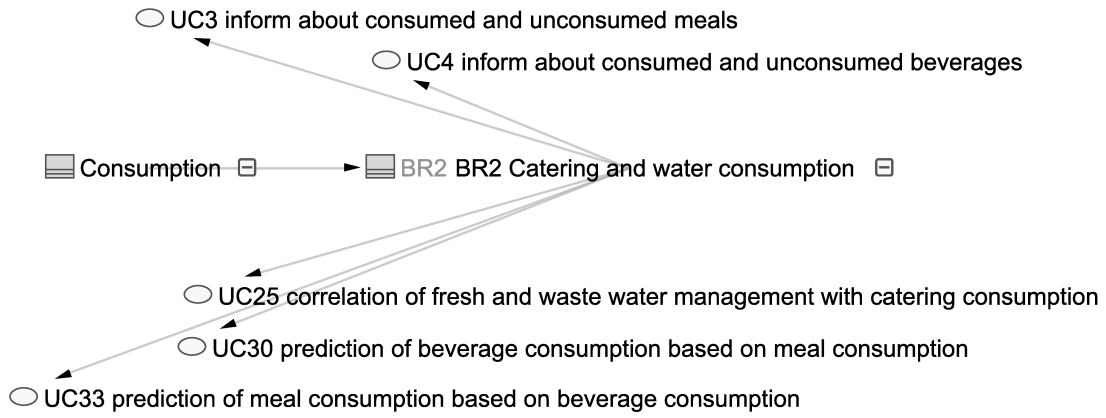


Figure 5.5: Traceability between pain points, business requirements, and use cases in a relation map of Cameo Systems Modeler

The illustrative relation map starts with the pain point *Consumption*, which describes the airlines' desire to get more information about catering and fresh water consumption. This pain point is traced to the business requirement *BR 2 Catering and water consumption* so that the airline need for improved information is included in the requirements. Finally, the relation map shows all use cases that refine this business requirement. These relation maps and dependency matrices enabled verification that all pain points were considered and traceability to use cases. While relation maps only focus on one pain point, a dependency matrix can consider all pain points and use cases at the same time by using a so-called metachain navigation. A metachain navigation promotes the traceability across a chain of relationships in a model. The metachain navigation for traceability between pain points and use cases considered here used the trace relationship between pain points and business requirements and the refine relationship between business requirements and use cases. The application of relation maps, dependency matrices, and metachain navigations enabled traceability and verification opportunities that conventional documents could not offer. The project partners particularly appreciated the opportunity to check that all pain points had been taken into account.

Some non-functional requirements for the data-driven catering CPSoS were already unintentionally mentioned and identified during discussions of pain points. More non-functional requirements were systematically obtained in a separate discussion with project partners. In addition, a collection of general non-functional requirements that are applicable to all development projects of a participating galley manufacturer was used to complete the list of non-functional requirements. For the evaluation of architecture drafts, KPIs were identified in parallel to the non-functional requirements, taking into account technological and economic aspects. Technology considerations included scalability, modularity, security and so on. Expected time to market, weight saving, and cost were part of economic KPIs. All KPIs were prioritized by comparing them in pairs and were further used for evaluating logical CPSoS

architecture drafts as presented in Section 5.3. The results of the method application are summarized in Figure 5.2 and include the following aspects:

- Stakeholder expectations modeled as operational scenarios and use cases: Stakeholder expectations previously collected in spreadsheets could be captured and detailed through missions, operational scenarios, and use cases. As the use cases were connected to use case activities and functions, a traceability could be established that the spreadsheets could not offer.
- Overview of existing interfaces: Existing interfaces of the systems to be integrated into the CPSoS were drawn in sketches that could not be traced to later development results so that the consistency was at risk. The modeling of an information network structure according to the proposed method provided an overview about existing interfaces. These interfaces were used for modeling the architectures in the CPSoS Function Identification and Architecture Development method so that the existing interfaces could be reused in the best possible way and traced to the architectures designed. Furthermore, extensive information network structures could be modeled in multiple consistent diagrams due to the model-based approach. If these information network structures were represented by multiple unrelated sketches, traceability would be greatly endangered.
- Key Performance Indicators for subsequent evaluation of architecture drafts: KPIs could be determined for evaluating architectures designed according to the CPSoS Function Identification and Architecture Development method (*cf.* Sections 4.2 on page 97 and 5.2 on page 135). A major advantage of the model-based approach over a document-based alternative is that these KPIs remain consistent through all possible changes in KPI definition and architecture changes.

The results of the definition of a mission, operational scenarios, use cases, etc. were the basis for the identification of CPSoS functions, as described in the following Section 5.2.

5.2 Validating CPSoS Function Identification and Architecture Development

The method CPSoS Function Identification and Architecture Development was applied to the development of CPSoS for the improvement of passenger flow control and seamless travel experience and data-driven catering. A summary of the results of method application is provided in Figure 5.6.

The refinement of CPSoS use cases according to the proposed method and the systematic analysis of use case activities in dependency matrices resulted in a minimum set of required functions. In both application projects, the number of functions and their interfaces could no longer be managed in conventional documents, so the method and the model significantly

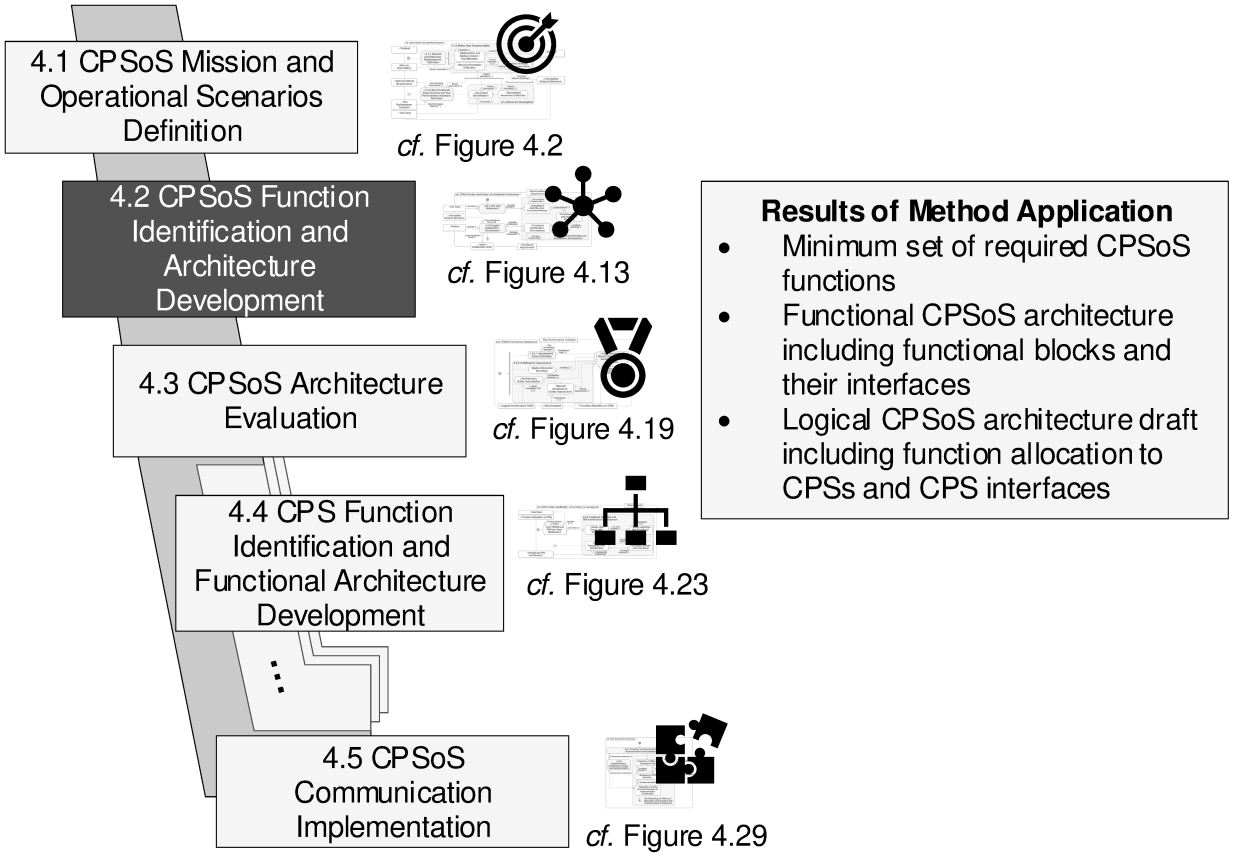


Figure 5.6: Application of the CPSoS Function Identification and Architecture Development method

supported the development. The method enabled the design of functional and logical CPSoS architectures based on the previously identified set of CPSoS functions. The high number of functions, logical elements, and their interfaces led to extensive functional and logical CPSoS architectures. The proposed model-based method supported the design and presentation of these architectures, as diagrams showed different aspects of the architectures. Although the diagrams showed only parts of the architectures for reasons of clarity, the use of the model automatically kept them consistent.

For the design of a CPSoS for the improvement of passenger flow control and seamless travel experience, use cases that were defined according to the method CPSoS Mission and Operational Scenarios Definition were modeled and refined by means of SysML activities. These activities were analyzed with respect to similarities and grouped to functional groups in a dependency matrix. Based on these functional groups, functional and logical CPSoS architectures were designed.

For a better overview, the logical CPSoS architecture is presented in Figure 5.7 without functions. It includes a general analytics platform, an aircraft analytics system, and a finger analytics system responsible for the collection and processing of data. The analytics platform

evaluates different modes of transport for supporting passengers on their way to and from the airport via the e-PAX Flow App. In addition, the analytics platform controls flow of passengers at the airport except for the passenger flow at the gate and in the aircraft. The aircraft analytics system and the finger analytics system take over this task and control the passenger flow during the boarding process.

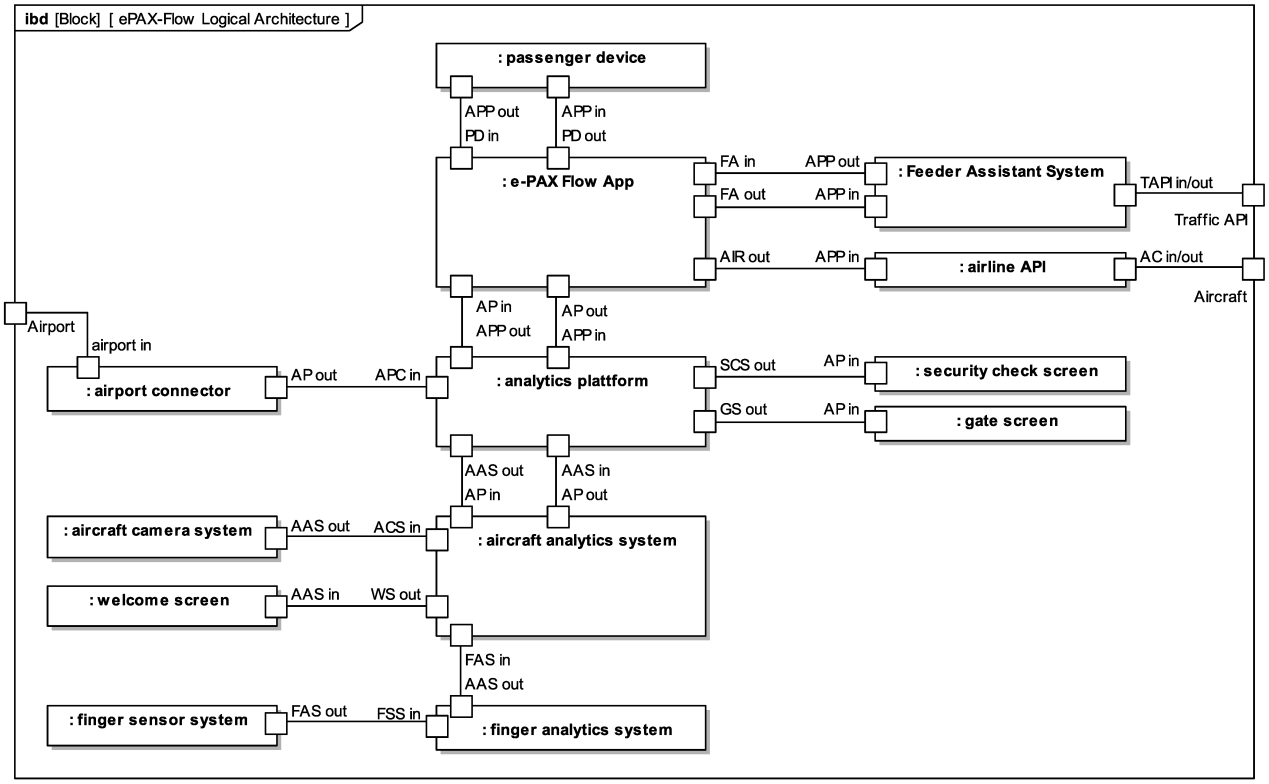


Figure 5.7: Logical Architecture for the improvement of passenger flow control and seamless travel experience, modeled in a SysML internal block diagram

The project partners' initial concepts, shown as examples in Figure 5.3 and Figure 5.4, were considered during the design of the logical CPSoS architecture. The aircraft camera system (left of the center of Figure 5.7) identifies passengers in the cabin and thus takes over the functions that were assigned to the sensor in the initial concept. Passengers are welcomed and the passenger flow is controlled by the welcome screen that provides the functions of the communication device of the draft in Figure 5.3 and the welcome panel at the gate in Figure 5.4. The functions of the computing device in the system design drafts are assigned to the aircraft analytics system and to the analytics platform.

The so-called e-PAX Flow App is a smartphone application and represents the interface to the passengers and assists passengers on their way to and from the airport. It provides different route options and warns about potential delays. If a risk for delays is identified, the app suggests alternative route options. The e-PAX Flow App has interfaces to the analytics platform to obtain information about possible delays at the airport, *e.g.* at the security check.

An interface to an airline API provides information about possible delays of the aircraft and a Feeder Assistant System evaluates data from public transport providers and road traffic information services.

The logical CPSoS architecture supported the assignment of CPSoS functions to the CPSs involved. It was observed that partners tended to propose initial concepts that were collected and used as an information source for the architecture design according to the proposed CPSoS Function Identification and Architecture Development method. Systematic application of the methods proposed in this thesis enabled the identification of all functions with as few function duplicates as possible. The logical CPSoS architecture supported the assignment of functions to CPSs, *e.g.* warning passengers about delays to the smartphone application, and the verification that all functions were considered and assigned to CPSs. Moreover, the logical CPSoS architecture allowed the design of interfaces in an environment where several project partners were developing CPSs to be integrated into the CPSoS.

Besides the design of a CPSoS for the improvement of passenger flow control and seamless travel experience, the method CPSoS Function Identification and Architecture Development was also applied to the development of a data-driven catering CPSoS. Use cases for the data-driven catering CPSoS were not directly refined by use case activities for two reasons. First, the large number of use cases resulted in a large effort to refine the use cases individually, and second, the project partners requested technical solutions to design an early version of a demonstrator representing the project results. Consequently, a combination of a top-down and a bottom-up approach was chosen. In the top-down portion of the approach, functional and logical architecture drafts were elaborated according to the method CPSoS Function Identification and Architecture Development. The bottom-up portion of the approach used a service-oriented broker architecture to provide a view of technical possibilities and inspire function identification. In the following, both directions, *i.e.* the top-down and the bottom-up portion of the approach, are presented.

The project partners were used to requirements-oriented development approaches so that functional requirements were identified based on use cases in the top-down portion of the approach. These functional requirements provided insights into the desired system behavior and were used for refining use cases with use case activities according to the proposed method. The use case activities were then analyzed for commonalities, and similar activities were traced to the same functional groups.

Parallel to the top-down portion of the chosen approach, a logical architecture was designed in the bottom-up portion of the approach. A service-oriented broker architecture from another project contained a single message broker for the connection of services in an aircraft cabin and was used as a basis. The system boundary of the data-driven catering CPSoS included galleys, a cabin server, and a ground system as shown in Figure 5.8.

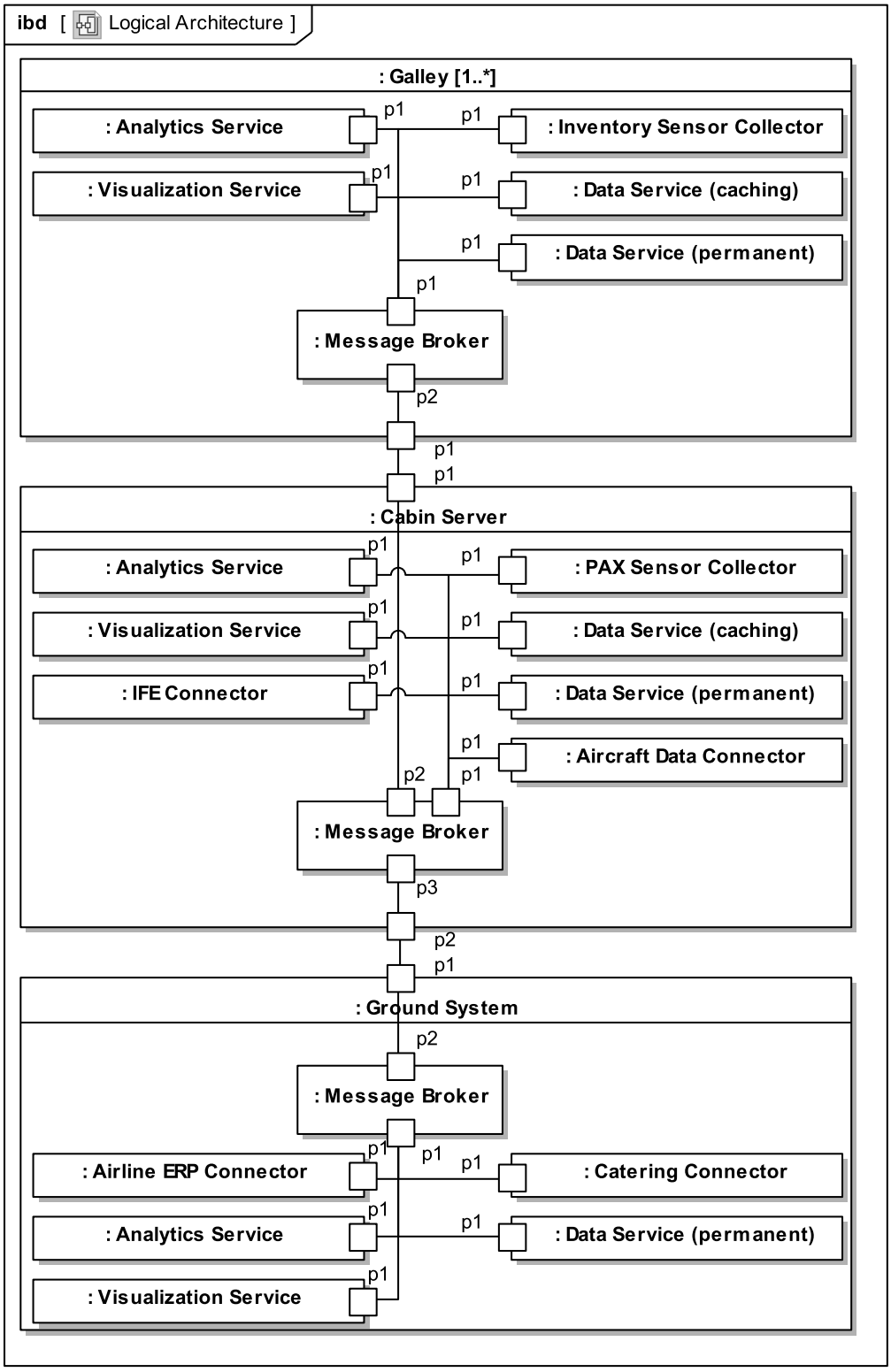


Figure 5.8: First draft of the logical CPSoS architecture for the data-driven catering CPSoS in a SysML internal block diagram

Each CPS in the form of galleys, cabin servers, and ground systems is composed of a message broker block and multiple services blocks. The message brokers connect the services within a CPS group. Besides the connection to services within a CPS, galley brokers are connected to cabin server brokers and cabin server brokers are connected to ground system brokers by means of broker federation. All CPSs offer the following services:

- *Analytics Service*: The *Analytics Service* is responsible for data analysis using statistical evaluation and artificial intelligence.
- *Visualization Service*: Inventory data and analysis results should be presented graphically by a *Visualization Service*.
- *Data Service (permanent)*: Data should be stored by the *Data Service (permanent)* and is the basis for evaluation by the *Analytics Service*.

Additional services that are not part of all CPSs include:

- *Inventory Sensor Collector*: Inventory of in-flight catering and consumption evaluation require the collection of sensor data in the galley. The *Inventory Sensor Collector* represents the interface to these sensors.
- *Data Service (caching)*: Data should be stored by the *Data Service (caching)* if the connection between message brokers is disrupted. When the connection is restored, the data is sent to connected message brokers via broker federation.
- *IFE Connector*: For additional passenger services that require the interaction with passengers during flight using the in-flight entertainment system (IFE), *e.g.* for meal ordering via IFE, a connection to the IFE system should be established.
- *PAX Sensor Collector*: Some use cases require the collection of additional data about the passenger. As an example, health-related use cases call for heart rate monitoring via pulse sensors. The *PAX Sensor Collector* is the interface to these passenger sensors.
- *Aircraft Data Connector*: Maintenance-related analyses use flight data for evaluation of operational data from galleys. The *Aircraft Data Connector* provides flight data, *e.g.* about flight conditions, for analyses by the *Analytics Service*.
- *Airline ERP Connector*: The *Airline ERP Connector* is the interface to the airline resource planning system in order to obtain information about flights and passengers.
- *Catering Connector*: The *Catering Connector* is the interface to the catering provider database.

For integrating project partners that are used to requirements-oriented development approaches, the identification of functional requirements was added to the method. It was noted that many functional requirements describe similar CPSoS functions, although derived from different use cases. As an example, inventory and prediction-related use cases require identification and counting of catering so that an inventory function can be derived from both use cases. In a graphical collaboration tool, similar functional requirements were indicated by means of arrows. The number of functional requirements and connecting arrays

became confusing as the number increased. Functional analysis and grouping according to the method CPSoS Function Identification and Architecture Development in combination with a SysML model enabled the identification of a set of functions that cover the entire desired system behavior and exhibit minimal redundancy. In addition, it was noted that functions for data handling had similarities. Often, similar functions emerged for different kinds of data. For an example, data on both passengers and catering consumption must be created, stored, and modified. The results from the functional analysis and grouping in the model supported the revision of functional requirements in the graphical collaboration tool so that redundant functions were consolidated. However, the identification of functional requirements was a valuable step for discussions about desired system behavior with project partners.

The parallel bottom-up approach with the presentation of the broker federation architecture showed project partners what was technically possible, especially with regard to configuration options for data exchange, so that reservations about data exchange with third parties were reduced. The parallel specification of functional requirements and the identification of functions were supported by reduced reservations about data exchange and an outlook on technical possibilities.

In conclusion, the application of the CPSoS Function Identification and Architecture Development led to the following results, summarized in Figure 5.6:

- Minimum set of required CPSoS functions: The functional analysis provided a minimum set of required CPSoS functions that satisfy all stakeholder needs while minimizing functional redundancy.
- Functional CPSoS architecture including functional blocks and their interfaces: The method supported the systematic derivation of a functional CPSoS architecture. This architecture is independent from technical solutions and provided a good overview about the required CPSoS functions and their interfaces.
- Logical CPSoS architecture draft including function allocation to CPSs and CPS interfaces: The logical CPSoS architectures enabled the allocation of CPSoS functions to CPSs. Since many CPSoS have a large number of functions and include multiple CPSs, the model-based approach chosen in the CPSoS Function Identification and Architecture Development method allowed the architecture design in consistent SysML diagrams that a document-based approach could not offer.

The CPSoS functions and functional and logical architecture were the prerequisite for the evaluation of logical CPSoS architectures, presented in the following Section 5.3.

5.3 Validating CPSoS Architecture Evaluation

The method CPSoS Architecture Evaluation was applied to the development of a CPSoS for data-driven catering. A summary of the results of method application is provided in

Figure 5.9.

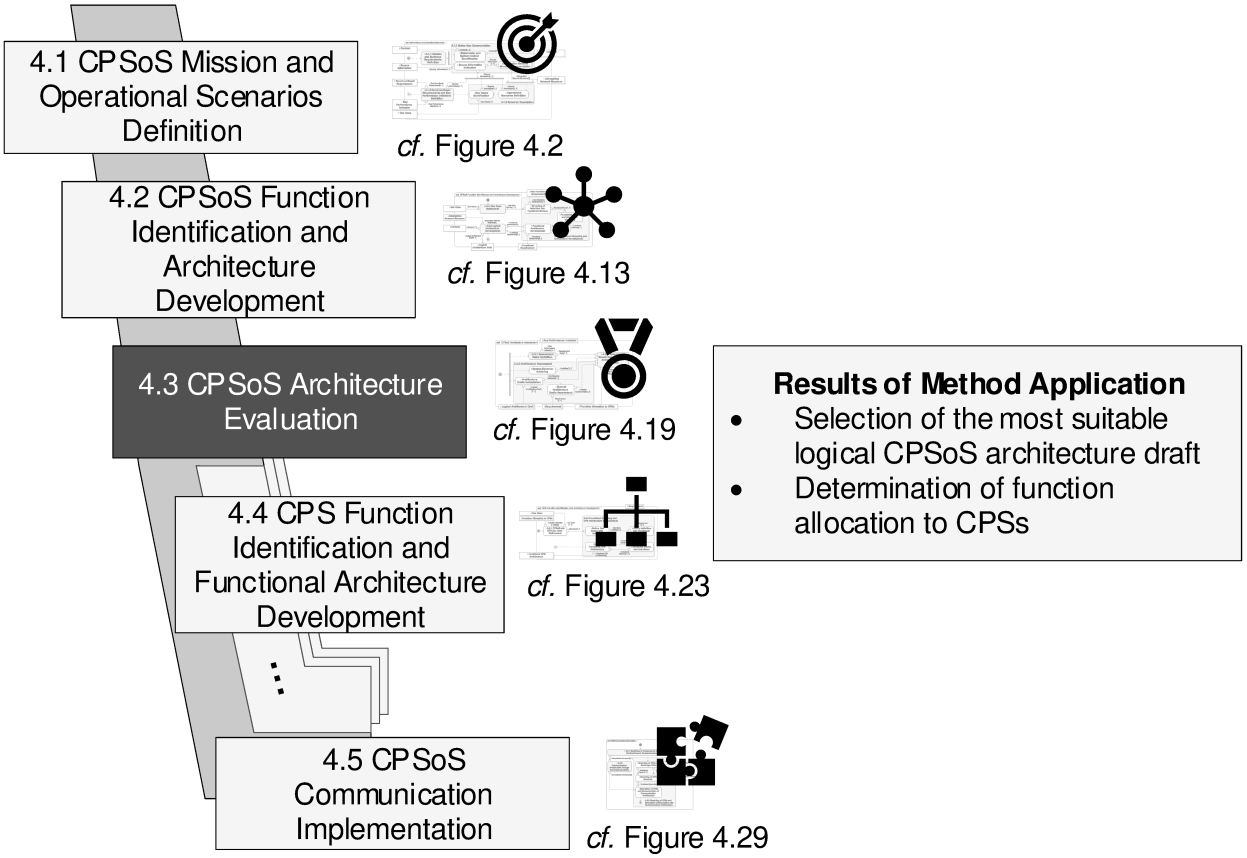


Figure 5.9: Application of the CPSoS Architecture Evaluation method

In the application project, the large number and the variety of functions led to multiple ways of allocating CPSoS functions to the CPSs involved. These allocations were represented by the different designs of the logical architecture. The CPSoS Architecture Evaluation method enabled the evaluation of these architecture designs and the selection of the most suitable CPSoS architecture draft. Thus, the allocation of CPSoS functions to the CPSs involved was determined.

The partners in the project already had a commonly used collection of technical and economic criteria for the evaluation of development projects. While technical evaluation criteria included considerations about scalability, modularity, maintainability, usability, and so on, economic criteria evaluated expected time to market, weight, and costs. Additional evaluation criteria for the data-driven catering CPSoS were identified in a workshop with all project partners and considered project-specific aspects for evaluation. Examples for project-specific criteria are privacy and expected accuracy of catering demand predictions. All criteria are evaluated with a value between zero and one, ranging from poor to good evaluation results. Figure 5.10 shows part of the values for the technological evaluation of the logical CPSoS architecture drafts.

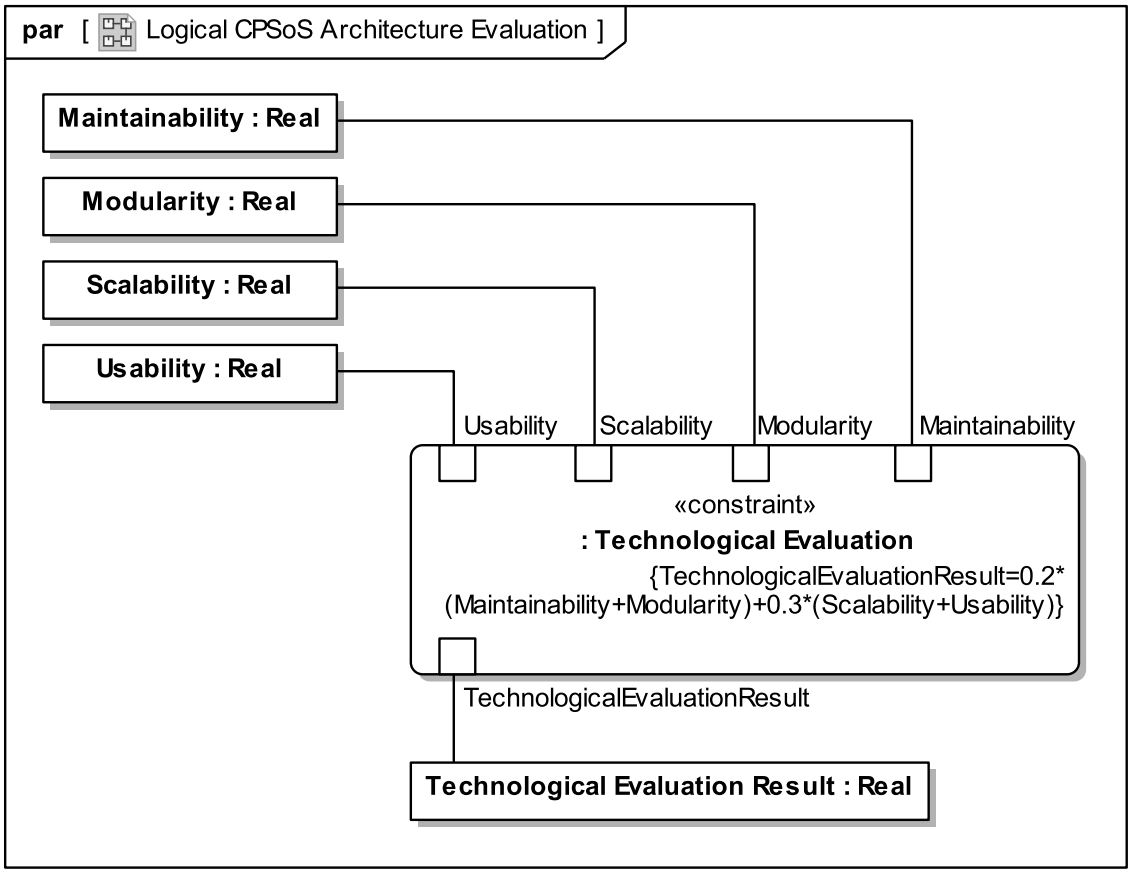


Figure 5.10: Parametric diagram for the technological evaluation of logical CPSoS architecture drafts for the data-driven catering CPSoS, modeled in a SysML parametric diagram

The constraint in the middle of the parametric diagram calculates the technological evaluation result. Scalability and usability are more important to project partners so that they are multiplied by the weighting factor with the value 0.3. Maintainability and modularity are not as important and are therefore multiplied by the weighting factor 0.2. The sum of the weighted criteria represents the technological evaluation result. The following describes the application of this constraint to the technological evaluation of two illustrative logical CPSoS architecture drafts of the data-driven catering CPSoS. The first architecture draft is shown in Figure 5.8 on page 139 in Section 5.2 about the application of the CPSoS Function Identification and Architecture Development method. The following Figure 5.11 presents a section of an illustrative alternative architecture draft that was discussed in the project. This alternative architecture does not allocate a message broker to the galleys. Instead, the galley services are directly connected to the message broker of the cabin server.

As the alternative architecture draft contained fewer message brokers, its maintainability was rated better than the initial draft with message brokers installed in all galleys. Fur-

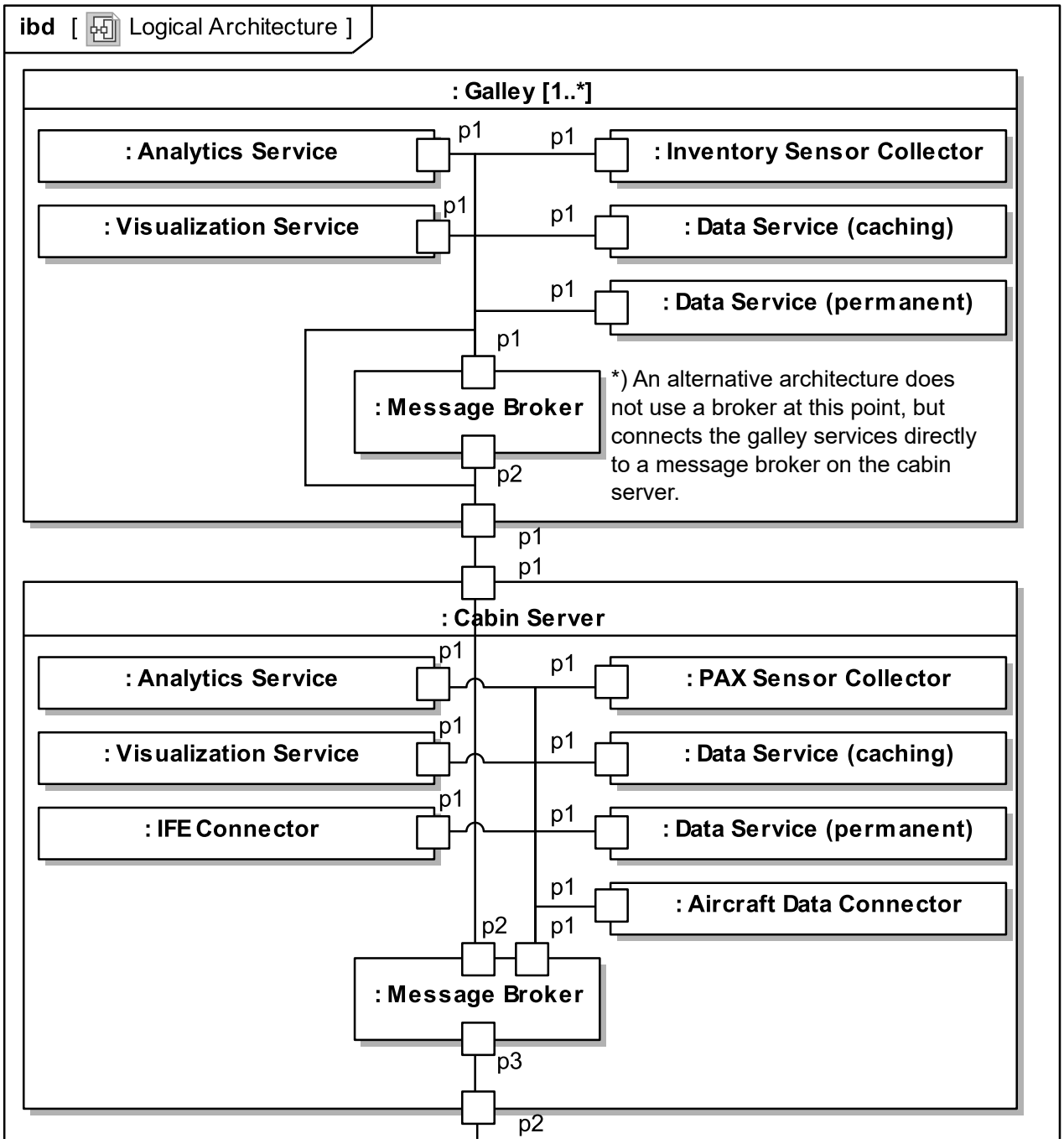


Figure 5.11: An alternative logical CPSoS architecture draft for the data-driven catering CPSoS does not integrate a message broker into the galleys. Instead, galley services are connected to the message broker of the cabin server.

thermore, the usability of the architecture drafts did not differ so that they were rated the same. However, the modularity and the scalability of the alternative architecture were worse than that of the initial architecture for several reasons. First, the message exchange between the services in the galleys and the message broker in the cabin server must be configured individually for each service in the alternative architecture. Consequently, the

galley could not be connected to any cabin server by different server providers. Second, the increased effort for message exchange configuration between multiple galley services and a cabin server contradicts the principle of contract-based communication (*cf.* Section 3.4 on page 78). The broker federation in the initial architecture enables just this contract-based communication between the message brokers of the galleys and the cabin server so that the data exchange between the galley and the cabin server can be configured in detail and according to the chosen agreement between the airline, providers of galleys, and the catering provider.

All of these considerations are illustrative and were incorporated into the evaluation of the logical CPSoS architecture drafts. Other criteria for the evaluation included economical aspects in the form of costs and expected time to market as well as project-specific criteria such as the accuracy of catering demand predictions and the privacy perceived by passengers. Some of these evaluation criteria were already collected during the use case definition as project partners initiated an evaluation of use cases during the application of the CPSoS Mission and Operational Scenarios Definition method. This evaluation revealed some project-specific evaluation criteria that were integrated into the architecture draft evaluation according to the CPSoS Architecture Evaluation method.

The application of the CPSoS Architecture Evaluation method supported the overview of the variety of evaluation criteria in the evaluation process of the designed logical CPSoS architecture drafts. Moreover, the integration of the evaluation into the model enabled the connection and traceability of the evaluation criteria, the architecture drafts, and the evaluation results. The application of the method showed that new ideas can emerge in discussions about architecture evaluation. Therefore, it may be useful to initiate the CPSoS Function Identification and Architecture Development method for realization of the ideas. Depending on the extent of ideas, existing architecture drafts can be edited or new architecture drafts can be developed.

Figure 5.9 on page 142 summarizes the results of the method application. The results include the following aspects:

- Selection of the most suitable logical CPSoS architecture draft: The CPSoS functions could be allocated to different CPSs so that multiple possible logical CPSoS architectures could be designed. The CPSoS Architecture Evaluation method supported the evaluation of these architecture alternatives so that the most suitable logical CPSoS architecture could be chosen.
- Determination of function allocation to CPSs: With the selection of the most suitable CPSoS architecture, the allocation of CPSoS functions to CPSs was determined.

The evaluation of logical CPSoS architectures revealed the most suitable allocation of CPSoS functions to CPSs. In the following Section 5.4, the consideration of these CPSoS functions for the design of functional CPS architectures is shown.

5.4 Validating CPS Function Identification and Functional Architecture Development

The method CPS Function Identification and Functional Architecture Development was applied to the development of the data-driven catering CPSoS. Figure 5.12 shows the related results of the application of the method.

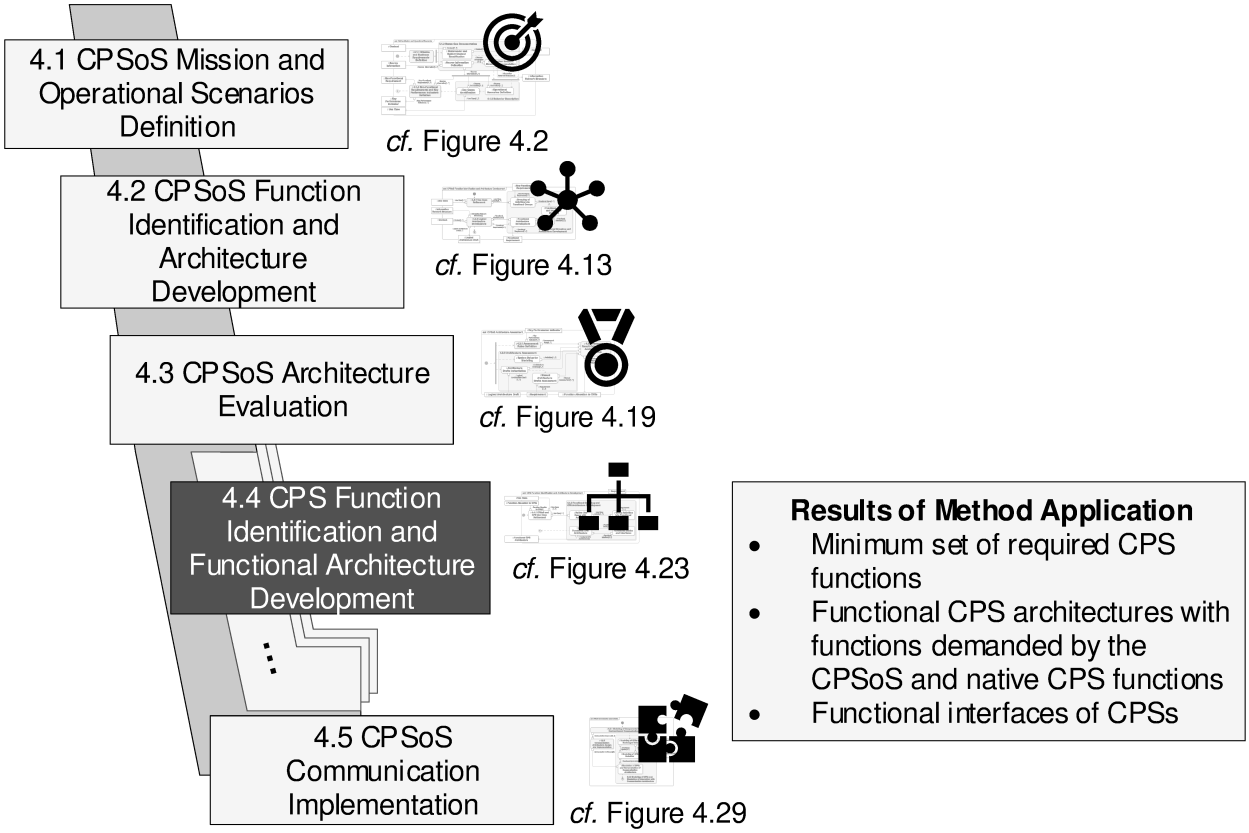


Figure 5.12: Application of the CPS Function Identification and Functional Architecture Development method

Section 5.2 on page 135 describes the design of a functional CPSoS architecture for the data-driven catering CPSoS. This functional CPSoS architecture assigns the functions “Present Meal Consumption” and “Present Beverage Consumption” to the visualization service on the ground (*cf.* Figure 5.8 on page 139). This visualization service is offered to airlines and provides an overview of catering consumption to airline employees. For application of the method CPS Function Identification and Functional Architecture Development, the CPSoS functions were modeled as cps-machine use cases of the visualization service on the ground. The use cases were refined by use case activities and analyzed for deriving functional groups. Based on functional groups and object flows in use case activities, the functional CPS architecture for the visualization service was designed, as shown in Figure 5.13.

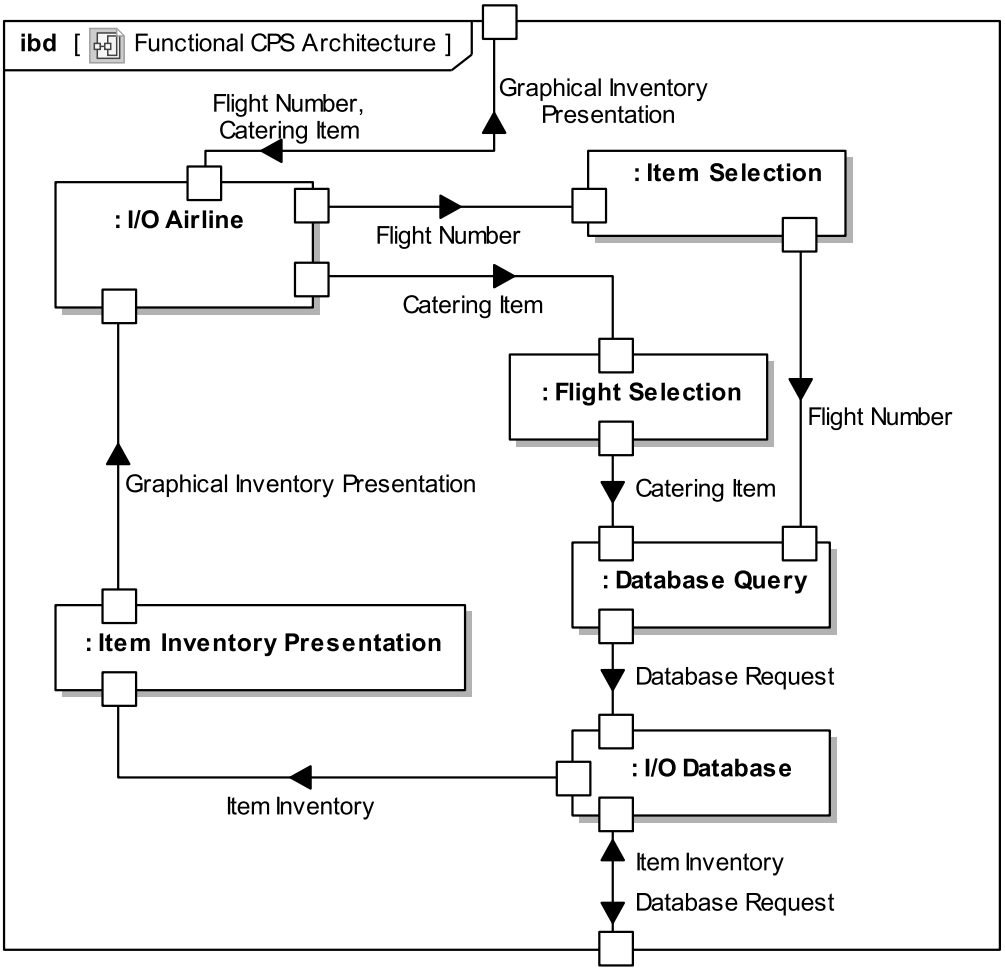


Figure 5.13: Segment of the functional CPS architecture for the visualization service for presentation of catering and beverage consumption, modeled in a SysML internal block diagram

The functional block *I/O Airline* represents the interface to the airline and receives flight number and catering item type as input from airline employees. The flight number and catering item type are provided to the functional blocks *Flight Selection* and *Item Selection*. Based on chosen flight number and catering item type, a database request is created by the functional block *Database Query* and provided to the functional interface block *I/O Database*. The database responds to the request with inventory data for the selected catering item type. Inventory data is provided to the *Item Inventory Presentation* functional block that prepares a graphical presentation of inventory data to *I/O Airline*.

The functional architecture for the visualization service was implemented for demonstration purposes. Figure 5.14 shows a prototype for the visualization service, which includes a backend for data processing in the background and a frontend for the preparation of the graphical presentation. The drop-down menus on top of the visualization service implement the functions *Flight Selection* and *Item Selection*. Inventory data is requested and

provided via an interface to the *Data Service* in the form of a database. The backend of the visualization service connects to a database and implements the functions *Database Query* and *I/O Database*. Received inventory data are then processed by the visualization service frontend that offers the *Item Inventory Presentation* and *I/O Airline* functions. Additional pages of the visualization service provide information about catering consumption of recent flights and revenue from catering sales.

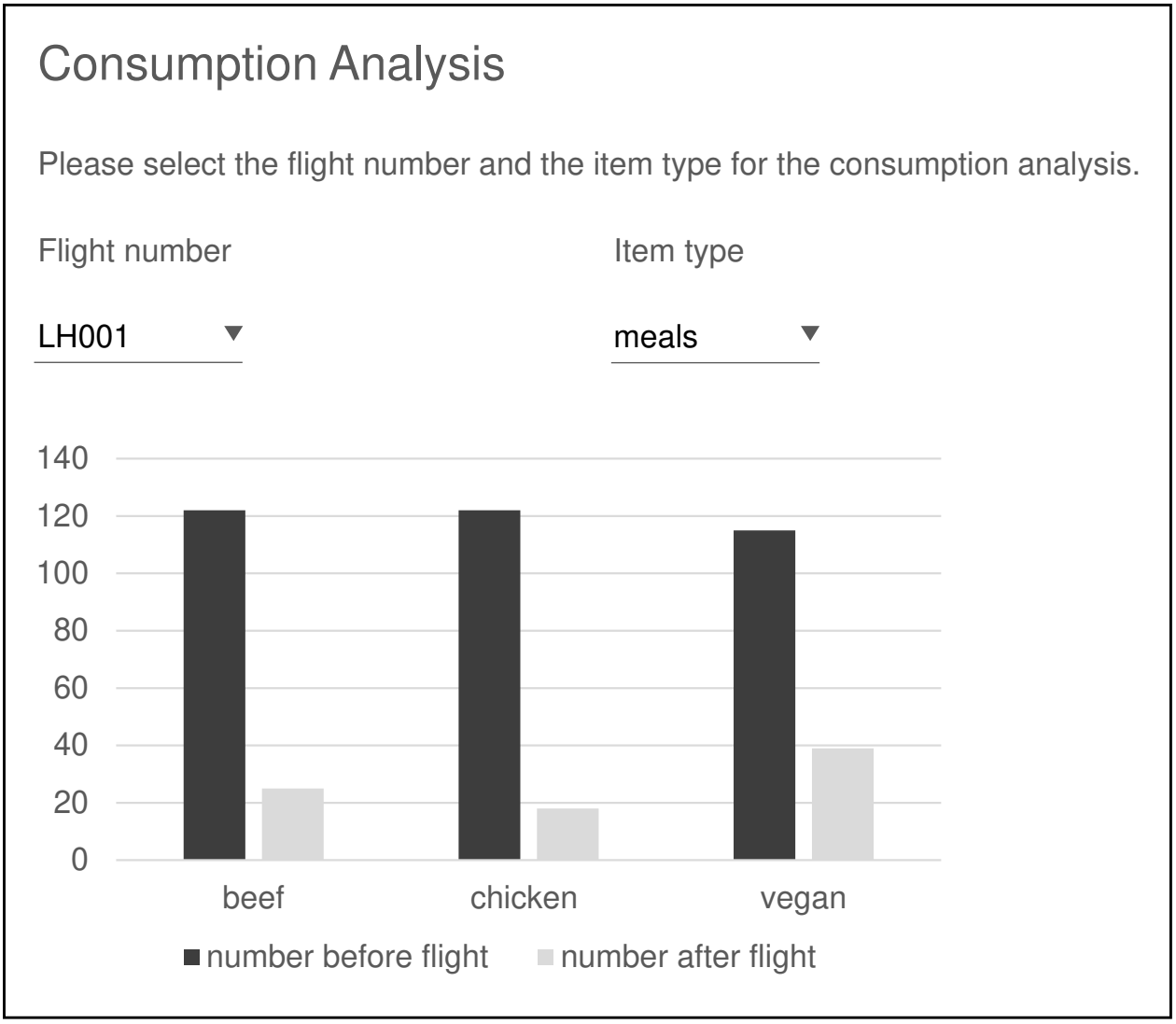


Figure 5.14: Illustrative implementation of a visualization service for presentation of catering and beverage consumption

Application of the method CPS Function Identification and Functional Architecture Development showed that a functional CPS architecture could be developed that implements CPS-human and CPS-machine use cases, following the definition in Section 4.4.1 on page 109. CPS-machine use cases are defined based on function allocation in the logical CPSoS architecture according to the method CPSoS Function Identification and Architecture Development. An example of a CPS-machine use case is the presentation of catering

inventory results to airline employees. Stakeholder expectations in the system without relation to the CPSoS are expressed in CPS-human use cases, *e.g.* presentation of sales revenue. The functional CPS architecture supported the development of a first prototype of the visualization service for two main reasons. First, the functional analysis of cps-human and cps-machine use cases and derivation of a functional CPS architecture ensured completeness of functions. Secondly, the specification of interfaces between CPS functions and the CPS environment supports the prototype design.

The results of the application of the CPS Function Identification and Functional Architecture Development are also summarized in Figure 5.12 on page 146 and include:

- Minimum set of required CPS functions: The method CPS Function Identification and Functional Architecture Development supported the identification of a minimum set of required CPS functions that satisfy the CPSoS and stakeholder demands with the lowest possible functional redundancy.
- Functional CPS architectures with functions demanded by the CPSoS and native CPS functions: The method enabled the identification and consideration of functions that cover the initial demands of the stakeholders of the CPS and the additional functions that result from the integration of the CPS into a CPSoS.
- Functional interfaces of CPSs: Functional interfaces of the CPSs were specified by means of SysML proxy ports and SysML connectors so that the specification detail of functional interfaces and the traceability of data and information exchange exceeded conventional document-based approaches.

The identified CPS functions and the functional CPS architecture are the basis for the more detailed design of the CPS.

5.5 Validating CPSoS Communication Architecture Implementation

The method CPSoS Communication Architecture Implementation was applied to the design and implementation of CPSoS communication architectures for the data-driven catering CPSoS and the aviation industry supply chain CPSoS.

The logical CPSoS architecture for the data-driven catering CPSoS in Figure 5.8 on page 139 shows that message brokers are part of galleys, the cabin server, and a ground system and that these message brokers are connected to each other. The AMQP-based RabbitMQ broker is chosen as the message broker because its broker federation plugin supports the connection of multiple systems according to the concept of contract-based communication. For demonstration of the data-driven catering CPSoS, three galley brokers, one cabin server broker, and a ground broker were designed and implemented, as shown in the communication architecture in Figure 5.16. In the following, the communication between

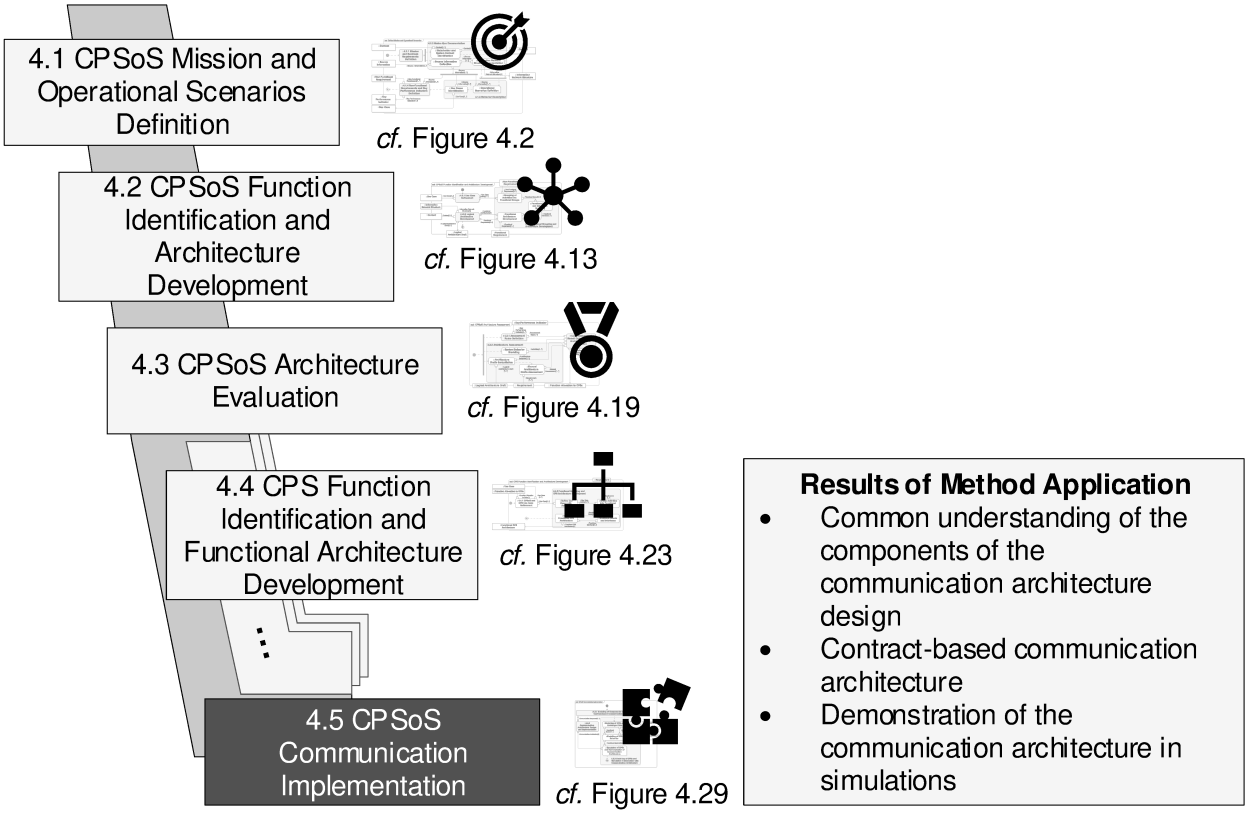


Figure 5.15: Application of the CPSoS Communication Architecture Implementation method

clients and one broker is described. Secondly, the communication between two message brokers is presented.

The logical blocks from the logical CPSoS architecture in Figure 5.8 act as broker clients and are modeled as SysML instance specifications, typified by the *Client* block. As an example, the logical block *Inventory Sensor Collector* is modeled as the instance specification *galley_inventory_sensor_collector* (*cf.* top left of Figure 5.16). Further clients are the *galley_analytics_service*, *galley_visualization_service*, *galley_data_service*, and *data_aggregator*. The clients exchange messages using the publish/subscribe and the remote procedure call (RPC) pattern of RabbitMQ [9]. The *galley_inventory_sensor_collector* publishes all sensor data to a *data_dump* queue. The *galley_data_service* receives all messages from the *data_dump* queue via a subscription and stores them in a database. The remaining services exchange messages using the RPC pattern. For this purpose, the *Message* block in the communication concept (*cf.* Figure 4.30 on page 117) is specialized with *Request* and *Response* blocks that are shown in the additional diagram in the appendix A.1 on page 185. Instance specifications that are classified by the *Request* and *Response* blocks describe the messages that are exchanged between clients in RPCs. RPC communication requires two queues, *i.e.* a task queue and a temporary response queue. As an example, the communication between the *galley_analytics_service* and the *galley_data_service* is considered in

the following. The *galley_analytics_service* publishes a *Request* to the *data_service_tasks* queue. Since the *galley_data_service* is subscribed to the *data_service_tasks* queue, it receives and processes the *Request* and sends a *data_service_response* to the *amqp.auto.gen_data_service* queue. As the name suggests, the *amqp.auto.gen_data_service* queue is created automatically when utilizing the RPC pattern. The *galley_analytics_service* receives these *Responses* via a subscription to the *amqp.auto.gen_data_service* and can use the requested data for data analyses.

Broker federation is used for data and information exchange between clients that are connected to different message brokers. As an example, a connection between the message brokers *galley_fwd* and *cabin_server* is established by means of an upstream and queue federation. As not all data from the galley should be sent to the cabin server, a *federation_output* queue is created on the *galley_fwd* message broker. The *data_aggregator* client collects data from the *galley_data_service* client via RPC and publishes the collected data to the *federation_output* queue so that a defined subset of available data is sent to the cabin server. When a connection between the *galley_fwd* and *cabin_server* is established, all messages stored in the *federation_output* queue are sent to the *federation_input* queue of the *cabin_server* and are therefore available to its clients, which are not included in the excerpt in Figure 5.16 for reasons of clarity.

The exchange of data and information via broker federation can be configured in more detail if exchanges and queues are used in combination with topic or headers exchange of RabbitMQ [9, 33, 45]. Topics and headers can include information about brokers that may receive data and information contained in the respective message. Topic and headers exchanges route incoming messages to queues that are federated to various message brokers depending on those topics and headers. The configuration of topics, headers, associated exchanges, and queues determines the contract-based communication in a CPSoS.

The galleys, cabin server, and the ground system were virtually implemented in the form of virtualized containers. As in-flight catering consumption data was not available, data of a food delivery service was used for demonstration of the clients and the communication architecture.

For the aviation industry supply chain CPSoS, first physical and information technology (IT) processes were identified, modeled, and analyzed with respect to potential for improvement [59]. Based on identified potential for improvement, design, production, and information management processes were developed that use CPS technology, *e.g.* electronic identification (eID). A representative business process with potential for improvement at both enterprises considered was the order process with exchange of orders and order confirmations. Orders and order confirmations were sent as PDF documents via mail, entailing great effort and representing a potential cause of errors, since order content was manually taken from these documents and entered into enterprise databases. Connecting enterprise

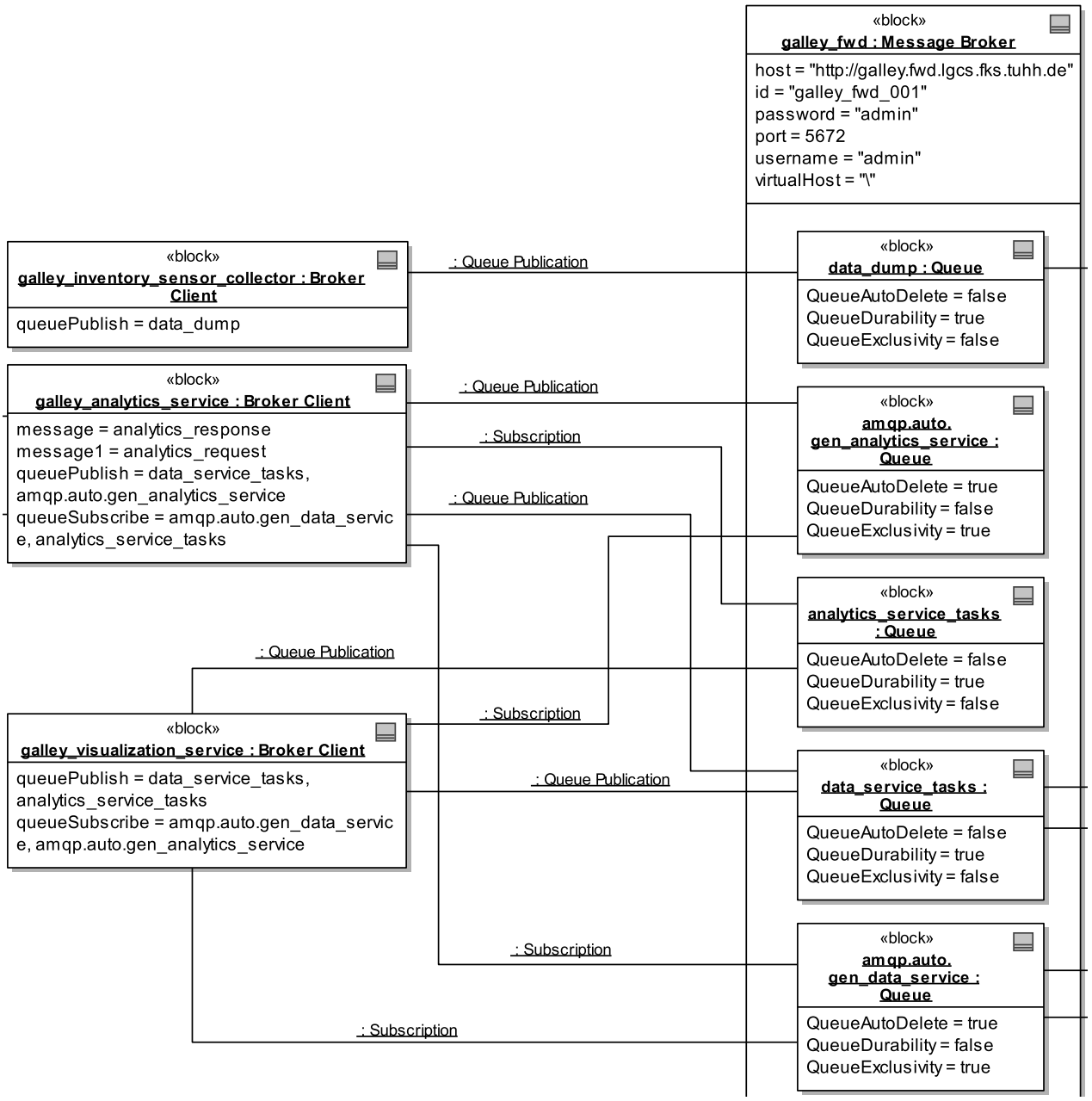


Figure 5.16: Segment of the communication architecture for message brokers in the galley and the cabin server, modeled in a SysML block definition diagram. Another segment is presented in the diagram in the appendix A.1 on page 185.

databases in the form of enterprise resource planning (ERP) systems via a communication architecture offered the potential for automated order management, reducing manual effort and the likelihood of errors. The enterprises interface with various other enterprises, so the data exchange had to be designed in a contract-based communication architecture. This communication architecture was designed and demonstrated using the method CPSoS Communication Architecture Implementation. The designed communication architecture supports the enterprises in the selection and configuration of a suitable communication

concept and ERP system.

The order and order confirmation processes were chosen for demonstration of inter-enterprise communication because their automation can decrease manual work and the likelihood for errors significantly. The related communication architecture is designed as a broker federation system. Each enterprise in the supply chain operates a RabbitMQ broker for data exchange with other companies. The brokers are implemented in separate docker containers on a virtual machine so that the existing enterprise systems are not affected by the demonstration. Figure 5.17 shows a part of the communication architecture. All elements of the communication architecture are modeled as SysML instance specifications and links. All instance specifications and links are typified by the blocks and associations for communication architecture design, as presented in Figure 4.30 on page 117. The part of the communication architecture presented contains brokers for enterprise A (“enterprise A broker”) and enterprise B (“enterprise B broker”). Each broker has a queue for order processing and a queue for order confirmations, denoted as “queue order” and “queue order confirmation.” The ERP systems of enterprise A and enterprise B act as clients and publish and subscribe to these queues. For exchange of orders and order confirmations between enterprises, upstreams and policies are defined. “upstreamA2B” allows the flow of messages from enterprise A broker to enterprise B broker. The “orders” policy defines that orders from the queue with the name “queue order” of enterprise A broker are sent to the ”queue order” queue of the enterprise B broker. For transmission of order confirmations in the opposite direction, the “upstreamB2A” and the policy “order confirmations” are defined. This policy determines that order confirmations from the “queue order confirmation” queue of enterprise B broker are sent to the queue “queue order confirmation” of enterprise A broker.

The communication architecture included multiple brokers in docker containers and was demonstrated to gain experience, to obtain feedback from project partners, and to present development results at the end of the project. Affected ERP systems were not fully implemented so that they could not be used for the demonstration. Instead, ERP system behavior in the context of the order and order confirmation process was simulated in a SysML environmental model that interacted with the brokers. Interaction with stakeholders in connection with the order and order confirmation process was simulated with GUIs for creating orders and order confirmations. The behavior of ERP systems was modeled in activity diagrams and included the GUI interaction and broker communication using Beanshell scripts of the broker-based SysML toolbox [92].

During the design of the communication architectures for the data-driven catering CPSoS and the aviation industry supply chain CPSoS, two roles emerged, *i.e.* the communication architecture expert and other developers. The communication architecture expert had fundamental knowledge of the communication concept and modeled the communication concept according to the process step described in Section 4.5.1 on page 116. Afterwards,

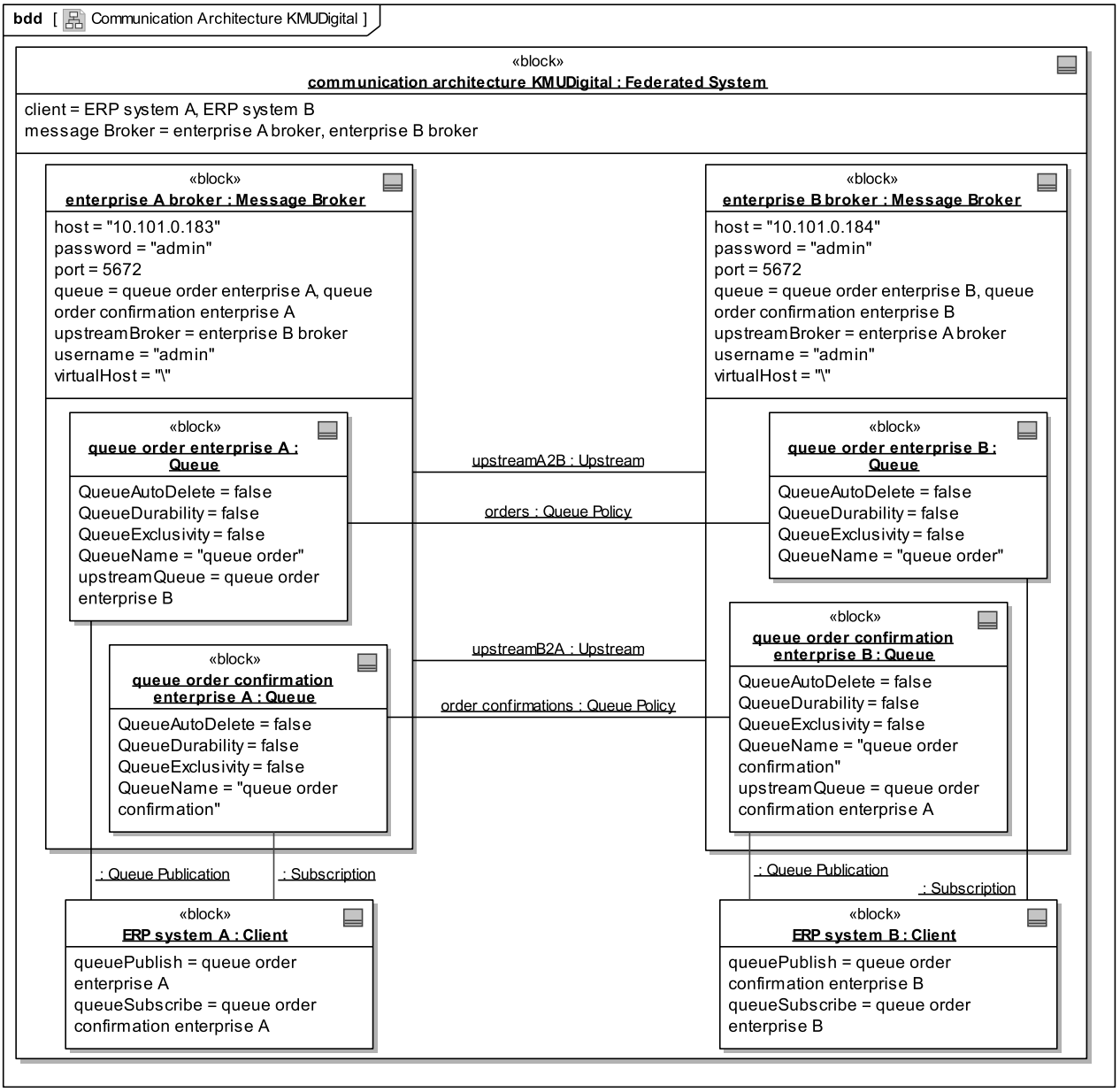


Figure 5.17: Segment of the communication architecture for inter-enterprise communication for the exchange of orders and order confirmations, modeled in a SysML block definition diagram

the communication architecture expert designed a first communication architecture draft and presented this to other developers. After the presentation, architectural aspects could be discussed and the architecture was modified accordingly. The presentation of the communication architecture using instance specifications and block definition diagrams supported the introduction into the chosen communication concept and provided an overview of the proposed architecture. Before using the method, brokers, their elements, and relations were drawn in graphics software. The extent of broker federation and the amount of configuration options created challenges in terms of consistency between diagrams. Model-

based communication architecture design according to the method offered advantages with respect to overview and consistency. The overview of possible configuration options for specific elements of broker communication, *e.g.* required host name and port number of message brokers, was automatically displayed for each instance specification after typification with a block from the communication concept (*cf.* Section 4.5.1). Consistency was maintained as all broker elements and their connections were stored as instance specifications and links in the model repository respectively. Diagrams depicted different aspects of the communication architecture and were based on the model repository so that changes to the communication architecture were automatically integrated in all diagrams. In summary, the proposed method facilitated understanding of contract-based communication, the chosen communication concept in the form of AMQP-based broker federation, and the designed communication architecture. The results of the method application are collected in Figure 5.15 on page 150 and are explained in the following:

- Common understanding of the components of the communication architecture design: The application of the method CPSoS Communication Architecture Implementation established a common understanding of the components for the communication architecture, *e.g.* queues and exchanges of the RabbitMQ message broker, between the stakeholders involved in the development.
- Contract-based communication architecture: Using the components of the chosen communication architecture concept, a communication architecture could be designed. The modeling of these components enabled the model-based design of a contract-based communication architecture before its implementation so that there was a good basis for discussions about contract-based exchange of data and information between the CPSs.
- Demonstration of the communication architecture in simulations: Based on this understanding, a contract-based communication architecture could be designed and implemented. Since the model contained the specification of the communication architecture, elements of the communication architecture could be simulated by the model. In this way, certain elements of the architecture could be simulated and interacted with elements that had already been implemented, so that parts of the architecture could be verified at an early stage.

5.6 Extensive Validation Results of the Developed Methodology

The results from application of the methods to the CPSoS development projects are used for validating the methods on the basis of the CPSoS properties described in Section 2.1. The following Table 5.1 summarizes how well the methods satisfy the CPSoS properties using Harvey balls and provides cross references to the methods and validation sections in this work. The subsequent sections explain the validation results in more detail, with a separate

paragraph dedicated to each method.

	Methods				
	CPSoS Mission and Operational Scenarios Definition	CPSoS Function Identification and Architecture Development	CPSoS Architecture Evaluation	CPS Function Identification and Functional Architecture Development	CPSoS Communication Architecture Implementation
Section for Presentation of the Method	4.1	4.2	4.3	4.4	4.5
Section for Validation of the Method	5.1	5.2	5.3	5.4	5.5
CPSoS Properties					
Complexity management	●	●	◕	●	●
Managerial and operational independence of CPSs	●	●	–	◕	◕
Consideration of CPS diversity	●	●	●	◕	●
Consideration of communication paths	●	●	●	●	●
Distributed control	●	●	●	●	●
Evolution	●	◑	●	◑	●

● fulfilled ◕ mostly fulfilled ◑ partially fulfilled

◔ fulfilled to a lesser extent ○ not fulfilled – not applicable

Table 5.1: Validation results

Complexity Management

The first criterion evaluates *Complexity management* capabilities of the methods. The method CPSoS Mission and Operational Scenarios Definition considers diverse stakeholder expectations and resulting complex CPSoS tasks and CPS interactions. CPSoS tasks are described by missions, related operational scenarios, and use cases on three levels of granularity.

Concepts for the allocation of CPSoS functions to CPSs are elaborated in the method CPSoS Function Identification and Architecture Development. Complexity management is supported by traceability between missions, operational scenarios, use cases, use case activities, and functional groups. This traceability enables analysis of the impact of changes and supports the verification that all stakeholder needs have been addressed. For example, the pain points in the data-driven catering CPSoS project were traced to use cases and derived requirements.

Evaluation of logical CPSoS architectures according to the method CPSoS Architecture Evaluation is supported by SysML parametric diagrams. These diagrams enable automated

evaluation result calculation for multiple logical CPSoS architecture drafts and avoid possible errors in manual calculation. However, the level of automation could be further increased to address the complexity of CPSoS through enhanced automated evaluation of architecture drafts.

Complexity management is supported for functional CPS architecture development and the implementation of CPSoS communication. The consideration of CPSoS functions parallel to native CPS use cases by human stakeholders enables the development of CPSs that are integrated into complex CPSoS.

Contract-based CPSoS communication architectures require multiple configuration options, thus resulting in complexity. These configuration options cannot be implemented ad-hoc during broker implementation and configuration but require prior design and specification. The method CPSoS Communication Architecture Implementation supports model-based design of communication concepts and the development of specific communication architectures using SysML instance specifications and links. The model-based design of a communication concept improved understanding of message broker communication in the projects for the design of a data-driven catering CPSoS and an electronic passenger flow control CPSoS. The implementation of specific CPSoS communication architectures was supported by instance specifications. Consistency of architecture elements in the form of instance specifications was ensured by the model and could not be provided by alternative graphics-based approaches.

Managerial and operational independence of CPSs

Managerial and operational independence of CPSs is one of the core characteristics of CPSoS and is also evident in all projects for method application. In the case of the data-driven catering CPSoS, aircraft systems are independent from systems at catering enterprises and they are operated by different organizations. Similarly, in the aviation industry supply chain CPSoS, systems at OEMs and their suppliers are independent of each other and operated by the respective companies. Application of the methods developed has shown that desired interaction and cooperation of independent CPSs can be described well in operational scenarios in SysML activity diagrams.

CPSoS functions were determined independently from any CPSs in the method CPSoS Function Identification and Architecture Development and were assigned to CPSs in logical CPSoS architecture drafts. The degree of independence of CPSs could be described in the CPSoS system context and was respected during the architecture design.

The evaluation of logical CPSoS architecture drafts according to the method CPSoS Architecture Evaluation does not emphasize managerial and operational independence so that this criterion is not applicable to this method.

The method CPS Function Identification and Functional Architecture Development considers functional interfaces of the CPS to its environment and other CPSs in the form of SysML proxy ports. Consequently, operational and managerial independence is maintained while necessary functional interfaces are taken into account.

In acknowledged and collaborative CPSoS (*cf.* Fig. 5.1 on page 128), which are within the scope of the methods developed in this thesis, organizations can agree on a communication concept so that the method CPSoS Communication Architecture Implementation can be applied well. In virtual CPSoS with a higher degree of CPSs independence, a convention with respect to the general communication concept could be challenging.

Consideration of CPS diversity

Missions, operational scenarios, and use cases identified with the CPSoS Mission and Operational Scenarios Definition method as well as functional CPSoS architectures designed according to the CPSoS Function Identification and Architecture Development method are independent of CPSs and describe stakeholder needs as well as resulting functions and their interfaces so that CPS diversity is not restricted. CPS diversity is considered in logical CPSoS architecture drafts by assigning functions to the diverse set of CPSs.

For the evaluation of logical CPSoS architecture drafts according to the method CPSoS Architecture Evaluation, CPS diversity can be addressed by KPIs and evaluation rules. KPIs and evaluation rules can be chosen as required so that manual evaluations and simulations of CPS behavior can consider a wide variety of CPSs.

Diversity of CPSs affects their individual development approach and thus has an effect on the method CPS Function Identification and Functional Architecture Development. How the proposed method harmonizes with the development approach chosen for the respective CPS must be investigated. Agile process models can be combined with the method if the proposed functional CPS architecture is reviewed and refined with each iteration.

It is possible that CPS diversity requires the use of multiple communication concepts, such as different message brokers or Application Programming Interfaces (APIs). For this purpose, several communication concepts can be combined by including their components and possible interfaces in the concept according to the first process step of the method CPSoS Communication Architecture Implementation. Based on the modeled concept, specific communication architectures can then be designed using SysML instance specifications, *e.g.* APIs and message brokers in the project for the development of an electronic passenger flow control CPSoS.

Consideration of communication paths

The cooperation of CPSs in a CPSoS requires the communication via communication paths. The method CPSoS Mission and Operational Scenarios Definition documents existing communication paths in the information network structure in SysML internal block diagrams. Further communication that is required for fulfilling the mission is identified in operational scenarios by means of SysML object flows in activity diagrams.

Based on these object flows, connectors and proxy ports indicate required communication paths in the functional CPSoS architecture. These communication paths are incorporated into the logical CPSoS architecture drafts and further refined in the CPSoS communication architecture.

The CPSoS Architecture Evaluation can take communication paths into account by including criteria regarding communication in the evaluation rules. These evaluation criteria can consider the number of interfaces, interface types, the degree of coupling between CPSs, and so on.

Functional CPS architectures that are developed according to the method CPS Function Identification and Functional Architecture Development represent communication paths to other CPSs by SysML proxy ports at the system boundary.

The method CPSoS Communication Architecture Implementation is intended for the design of communication paths between the CPSs involved based on the selected logical CPSoS architecture. SysML instance specifications and links represent the elements and their connections within the chosen communication concept.

Distributed control

Communication paths between CPSs are used for distributed control within a CPSoS. SysML object flows in operational scenarios connect pins of actions and represent communication paths for distributed control. The pins are typified by SysML blocks for more detailed specification of the exchanged control signals.

The same blocks specify proxy ports in the logical CPSoS architecture and can be further refined during application of the method CPSoS Function Identification and Architecture Development.

Different concepts for distributed control can exist, leading to alternative logical CPSoS architecture drafts that are evaluated in the method CPSoS Architecture Evaluation.

After selecting the most suitable logical CPSoS architecture draft, interfaces for distributed control are considered in the functional CPS architectures by means of SysML proxy ports

for modeling the functional interface and I/O functional blocks for the provision of interface functions.

Implementation of a CPSoS communication architecture according to the method CPSoS Communication Architecture Implementation enables distributed control of CPSs based on the specification in the selected logical CPSoS architecture draft.

Evolution

CPSs evolve over time, CPSs are added to the CPSoS, and CPSs leave the CPSoS. When the CPSoS evolves or new services are to be provided by the CPSoS, new missions can be added using the method CPSoS Mission and Operational Scenarios Definition.

New missions affect the use case collection so that new use case activities emerge. Functional grouping and subsequent design of a functional CPSoS architecture must be done again after each change to the mission set so that some effort is caused. Consequently, the method CPSoS Function Identification and Architecture Development partially fulfills the evaluation criterion *Evolution*. However, the effort must be made so that the amount of functions does not become unnecessarily large by creating redundant functions.

Changes to the functional CPSoS architecture require a revision of logical CPSoS architectures. The method CPSoS Architecture Evaluation supports the evaluation of revised and new architecture drafts. If new KPIs are added, existing evaluation rules can be adjusted without much effort.

If evolution affects the assignment of functions in CPSoS, functional CPS architectures must also be revised. New cps-machine use cases should be added and refined through use case activities. These additional use case activities influence functional groups, blocks, and interfaces so that the functional CPS architecture has to be modified. This is supported by the CPSoS communication architecture that provides an overview of existing communication paths between CPSs. This overview facilitates the implementation of evolutionary changes in the architecture. Existing communication paths are visible and can be reused so that only necessary paths are added to the CPSoS communication architecture.

In conclusion, the application of the methods to the projects allowed a successful validation of the methodology developed. The methodology supports systems engineers during the specification and the design of CPSoS and the CPSs they contain. It ranges from the identification of stakeholder needs at CPSoS level to the implementation of a CPSoS communication architecture and the design of functional architectures of participating CPSs.

Furthermore, CPSoS-characteristic properties are taken into account, which distinguish CPSoS from self-contained systems.

6 Summary

Informatization and connectivity enable new services and more efficient processes in all aspects of aeronautics through cooperation between formerly more isolated systems. These isolated systems were characterized by very few interfaces to other systems and are now evolving into Cyber-physical Systems (CPSs) due to the increasing availability and bandwidth of internet connections on the ground and in flight. CPSs are usually managed and operated by different organizations, *e.g.* airlines, original equipment manufacturers (OEMs), and maintenance, repair, and overhaul (MRO) enterprises so that the cooperating set of CPSs exhibits the properties of Cyber-physical Systems of Systems (CPSoS) with managerially and operationally independent CPSs. While functions in formerly isolated systems were bundled and allocated to specific systems, functions in CPSoS are distributed over several CPSs. The combination of increased communication between systems, distribution of functions, and aviation-specific requirements for safety and security result in a complex development task, which was addressed in this thesis.

The specification and design of systems is methodically supported by process models that organize the development process in a structured manner. In aviation, the V model is an established process model that allows continuous verification and final validation of development results thus meeting aviation-specific requirements with respect to reliability of safety-critical systems. Nevertheless, the intensive communication of CPSs and the distribution of functions challenge the V model which is intended for the specification, design, and integration of self-contained systems with a limited number of interfaces. In addition, existing process models are applied on the basis of documents, *e.g.* requirements documents, thus resulting in consistency and traceability challenges.

Model-based Systems Engineering (MBSE) in combination with the Systems Modeling Language (SysML) offers a promising approach to meet these challenges. In MBSE, models support all life cycle phases of the system in terms of requirements definition and management, design, analysis, verification, and validation. Models allow early simulation of the system behavior and thus verification at an early stage in development. The Unified Architecture Framework (UAF) represents a first model-based approach for SoS development. However, it is intended for enterprise modeling and offers potential for improvement with respect to CPSoS modeling. Moreover, it should be noted that development life cycle phases for civil aircraft and systems are defined in Aerospace Recommended Practice (ARP) 4754A, *i.e. Concept*, *Function*, *Architecture*, *Design*, *Implementation*, and *Operation*. To fill the gap for the design of CPSoS in aviation while taking into account the ARP life cycle phases, this thesis provides a methodology for the development of CPSoS in aviation. This methodology is intended to assist system architects in developing new services based on the cooperation

of CPSs in a CPSoS. It uses the life cycle phases of the ARP 4754A, the SysML, concepts from UAF, and includes methods, processes, and tools.

The methodology proposed in this thesis adapts concepts from existing approaches suitable for CPSoS development and extends them with new concepts to form a complete methodology. This methodology adds a CPSoS development level to the V model and includes five methods. The first method CPSoS Mission and Operational Scenarios Definition is allocated to the concept phase of the ARP 4754A and supports the identification of stakeholder needs. Based on the stakeholder needs, a functional architecture and multiple candidate logical architecture drafts are designed using the method CPSoS Function Identification and Architecture Development that supports the function and architecture life cycle phases of the ARP 4754A. The architecture life cycle phase is further supported by the method CPSoS Architecture Evaluation that facilitates the selection of the most suitable logical CPSoS architecture draft. The selected logical CPSoS architecture determines the distribution of functions among CPSs in the CPSoS. These functions should be considered in the design of functional CPS architectures by means of the method CPS Function Identification and Functional Architecture Development. This method allows both types of functions to be considered, *i.e.* functions required by CPS stakeholders and functions demanded by the CPSoS. The final method CPSoS Communication Architecture Implementation supports the design and implementation life cycle phases of the ARP 4754A and enables the elaboration of a CPSoS communication architecture for the exchange of data and information between the CPSs contained in the CPSoS. All methods are presented using the development of an intelligent catering CPSoS to have a consistent application example. Services provided by the intelligent catering CPSoS should increase passenger comfort and satisfaction, optimize cabin operation, and increase airline revenue. Example use cases include pre-ordering meals by passengers, meal suggestions for passengers, and more detailed catering consumption reports.

The proposed methodology was applied in three projects for validation purposes. Results from the application of the methods to these projects are gathered and provided. The validation showed that CPSoS-specific properties, in particular complexity, managerial and operational independence of CPSs, CPS diversity, multiple communication paths, distributed control, and evolution, are taken into account by the methodology. The end-to-end use of SysML from the identification of stakeholder needs to the implementation and simulation of a CPSoS communication architecture with an environment model, as well as the development of functional architectures of CPSs, is unique and extends the state of the science.

Parts of the results obtained in the course of this thesis are the subject of the publications [52, 53, 55–58, 60, 61, 72, 90–92, 106]. Another publication about the CPS Function Identification and Functional Architecture Development method [54] is in the review process.

The CPSoS-characteristic communication between CPSs offers interfaces to other development disciplines that are presented in Fig. 6.1.

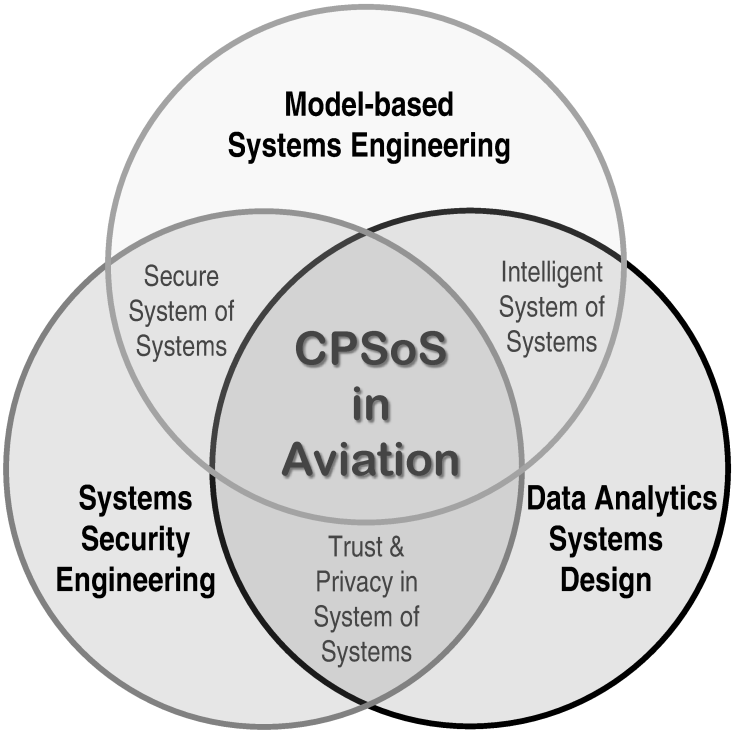


Figure 6.1: Further disciplines for the CPSoS design [52]. A more detailed graphic is presented in the appendix A.3 on page 188.

The increased networking of systems is accompanied by risks for potential attacks. One aspect is the distribution of functions among various CPSs. Depending on the criticality of the functions, the CPSoS mission may be at risk if one of the CPSs is attacked and cannot provide the intended functions. In addition, one of the connections between CPSs could be interrupted so that the exchange of data and information between functions is prevented and functions can no longer be provided. Moreover, each interface of the CPSs represents a potential vulnerability, since it offers attackers a possible access to the system. For this reason, security considerations should already be taken into account during the specification and design of CPSoS to avoid later changes after threats have been identified. Hence, methods and tools of the discipline Systems Security Engineering support systems engineers in the systematic identification and mitigation of potential threats [40, 76–78, 95].

However, the exchange of data and information also offers potential. The increased extent of exchanged data and information from the operation of a CPSoS can be used for the development of artificially intelligent systems [52]. An artificially intelligent CPSoS can be used to support the passenger travel process, airport operations, air traffic management, airline operations, and the air cargo transport chain. This includes, for example, increasing

the availability of systems in aircraft by evaluating their operating data [73]. To exploit the potential of artificially intelligent systems, the Data Analytics System Design discipline is dedicated to the development of architectures for data analysis and machine learning including the use of artificial intelligence.

Bibliography

[1] Smart Systems - Vernetzte, intelligente Systeme für moderne Anwendungen. https://www.wirtschaft-digital-bw.de/themen/thema-des-monats/smart-systems-vernetzte-intelligente-systeme-fuer-moderne-anwendungen/. Accessed: 2024-02-01.

[2] Systematic engineering design of technical systems and products. Standard VDI 2221, The Association of German Engineers, 1985.

[3] Design methodology for mechatronic systems. Standard VDI 2206, The Association of German Engineers, 2004.

[4] OMG Systems Modeling Language (OMG SysML™): Version 1.0. Specification, Object Management Group, 2007.

[5] Systems Engineering Vision 2020. Technical report, International Council on Systems Engineering, 2007.

[6] Guidelines For Development of Civil Aircraft and Systems. Standard ARP 4754A, Society of Automotive Engineers, Inc, Dec. 2010.

[7] Systems and Software Engineering – Vocabulary. Standard ISO/IEC/IEEE 24765-2017(E), Institute of Electrical and Electronics Engineers, 2017.

[8] Unified Architecture Framework (UAFP). Specification Version 1.0, Object Management Group, 2017.

[9] AMQP 0-9-1 Model Explained. https://www.rabbitmq.com/tutorials/amqp-concepts.html, 2019. Accessed: 2022-11-11.

[10] Design of technical products and systems - Model of product design. Standard VDI 2221, The Association of German Engineers, 2019.

[11] MQTT Topics & Best Practices - MQTT Essentials: Part 5. https://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices/, 2019. Accessed: 2024-02-01.

[12] OMG Systems Modeling Language (OMG SysML™): Version 1.6. Specification, Object Management Group, 2019.

[13] Industry 4.0 - Guide to Fourth Industrial Revolution. https://www.knowledgepublisher.com/article/1437/industry-4-0-guide-to-fourth-industrial-revolution.html, 2020. Accessed: 2022-11-11.

[14] UAF Wiki. https://www.omgwiki.org/uaf/doku.php, 2020. Accessed: 2022-11-23.

[15] Anteil der Smartphone-Nutzer in Deutschland nach Altersgruppe im Jahr 2021. https://de.statista.com/statistik/daten/studie/459963/umfrage/anteil-der-smartphone-nutzer-in-deutschland-nach-altersgruppe/, 2021. Accessed: 2022-11-25.

[16] Development of mechatronic and cyber-physical systems. Standard VDI/VDE 2206, The Association of German Engineers, 2021.

[17] Federation Plugin for RabbitMQ. https://www.rabbitmq.com/federation.html, 2021. Accessed: 2022-11-23.

[18] Systems Engineering Book of Knowledge. https://www.sebokwiki.org/wiki/Functional_Architecture_(glossary), 2021. Accessed: 2022-11-25.

[19] Unified Architecture Framework (UAFP) Domain Metamodel. Specification Version 1.1, Object Management Group, April 2020.

[20] Systems and software engineering – System life cycle processes. Standard ISO/IEC 15288:2008, Feb. 2008.

[21] Systems and software engineering – System life cycle processes. Standard ISO/IEC/IEEE 15288:2015, May 2015.

[22] Systems and software engineering – System of systems (SoS) considerations in life cycle stages of a system. Standard ISO/IEC/IEEE 21839:2019, July 2019.

[23] Systems and software engineering – Guidelines for the utilization of ISO/IEC/IEEE 15288 in the context of system of systems (SoS). Standard ISO/IEC/IEEE 21840:2019, Dec. 2019.

[24] Systems and software engineering – Taxonomy of systems of systems. Standard ISO/IEC/IEEE 21841:2019, July 2019.

[25] Software, systems and enterprise – Architecture description. Standard ISO/IEC/IEEE 42010:2022.

[26] Aircraft / Ground IP Communication. Standard ARINC 822, June 2008.

[27] R. L. Ackoff. Towards a System of Systems Concepts. *Management Science*, 17(11):661–671, 1971.

[28] G. Akhras. Smart materials and smart systems for the future. *Canadian Military Journal*, 1(3):25–31, 2000.

[29] M. A. Allison, R. Batdorf, and H. Chen. The Characteristics and Emerging Behaviors of System of Systems. 2005.

[30] O. Alt. *Modell-basierte Systementwicklung mit SysML*. Hanser, München, 2012.

[31] H.-H. Altfeld. *Commercial Aircraft Projects: Managing the Development of Highly Complex Products*. Ashgate Pub., Farnham, Surrey, England and Burlington, VT, 2010.

[32] H. Ametsreiter. Smartphone-Markt: Konjunktur und Trends. https://www.bitkom.org/, 2017. Accessed: 2022-11-13.

[33] E. Ayanoglu, D. Nahum, and Y. Aytas. *Mastering RabbitMQ*. Community Experience Distilled. Packt Publishing, Birmingham, UK, 2015.

[34] T. Bahns, S. Melzer, R. God, and D. Krause. Ein modellbasiertes Vorgehen zur variantengerechten Entwicklung modularer Produktfamilien. In S.-O. Schulze and C. Tschirmer, editors, *Tag des Systems Engineering, Ulm, 11.-13. November 2015*, pages 141–150. Hanser, München, 2016.

[35] T. S. Baines, H. W. Lightfoot, S. Evans, A. Neely, R. Greenough, J. Peppard, R. Roy, E. Shehab, A. Braganza, A. Tiwari, J. R. Alcock, J. P. Angus, M. Bastl, A. Cousens, P. Irving, M. Johnson, J. Kingston, H. Lockett, V. Martinez, P. Michele, D. Tranfield, I. M. Walton, and H. Wilson. State-of-the-art in product-service systems. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, 221(10):1543–1552, 2007.

[36] D. Balkhausen. *Die dritte industrielle Revolution - Wie die Mikroelektronik unser Leben verändert*. Econ Verlag, 1978.

[37] S. Beer. *Brain of the Firm John*. John Wiley & Sons, New York, 1972.

[38] H. D. Benington. Production of Large Computer Programs. *Annals of the History of Computing*, 5(4):350–361, 1983.

[39] B. J. L. Berry. Cities as systems within systems of cities. *Papers in regional science*, 13(1):147–163, 1964.

[40] B. Best, J. Jurjens, and B. Nuseibeh. Model-based Security Engineering of Distributed Information Systems Using UMLsec. In *29th International Conference on Software Engineering (ICSE'07)*, pages 581–590. IEEE, 2007.

[41] J. Boardman and B. Sauser. System of Systems - the meaning of of. In *IEEE/SMC International Conference on System of Systems Engineering*, pages 118–123, Piscataway, NJ, 2006. IEEE Operations Center.

[42] B. W. Boehm. Guidelines for Verifying and Validating Software Requirements and Design Specifications. In P. A. Samet, editor, *Euro IFIP 79*, pages 711–719. North Holland, 1979.

[43] B. W. Boehm. A spiral model of software development and enhancement. *Computer*, 21(5):61–72, 1988.

[44] A. Bondavalli, H. Kopetz, and S. Bouchenak. *Cyber-Physical Systems of Systems: Foundations – A Conceptual Model and Some Derivations: The AMADEOS Legacy*. Springer, 2016.

[45] S. Boschi and G. Santomaggio. *RabbitMQ Cookbook*. Packt Open Source Community Experience Distilled. Packt Publishing, Birmingham and Mumbai, 2013.

[46] C. Brecher, D. B. Rawat, H. Song, and S. Jeschke, editors. *Cyber-Physical Systems: Foundations, Principles and Applications*. Academic Press, London, 2017.

[47] A. W. Colombo. *Industrial Cloud-Based Cyber-Physical Systems: The IMC-AESOP Approach*. Springer International Publishing AG, Cham, 1st edition, 2014.

[48] J. Dahmann, G. Rebovich, and J. Lane. Systems Engineering for Capabilities. *CrossTalk, The Journal of Defense Software Engineering*, 21:7, Nov 2008.

[49] J. S. Dahmann and K. J. Baldwin. Understanding the Current State of US Defense Systems of Systems and the Implications for Systems Engineering. In *2nd Annual IEEE Systems Conference, 2008*, pages 1–7, Piscataway, NJ, 2007. IEEE Operations Center.

[50] Department of Defense. Systems Engineering Guide for Systems of Systems, V 1.0.

[51] E. Dinc, M. Vondra, S. Hofmann, D. Schupke, M. Prytz, S. Bovelli, M. Frodigh, J. Zander, and C. Cavdar. In-Flight Broadband Connectivity: Architectures and Business Models for High Capacity Air-to-Ground Communications. *IEEE Communications Magazine*, 55(9):142–149, 2017.

[52] O. C. Eichmann, F. Giertzsch, and R. God. Digitale Transformation - Systeme in der Luftfahrt im Wandel. *Ingenieurspiegel*, (1):36–38, 2019.

[53] O. C. Eichmann, R. God, and S. Melzer. In T. Weilkiens, J. G. Lamm, S. Roth, and M. Walker, editors, *Model-based System Architecture*, Systems Engineering and Management, chapter Systems, Systems of Systems, and Cyber-Physical Systems, pages 17–30. Wiley, 2022.

[54] O. C. Eichmann, J. Lamm, S. Melzer, T. Weilkiens, and R. God. Development of functional architectures for cyber-physical systems using interconnectable models. *Systems Engineering*, pages 1–19, 2024.

[55] O. C. Eichmann, J. G. Lamm, S. Melzer, and R. God. Deriving Functional Architectures for Cyber-Physical Systems Using Interconnectable Models. In *Tag des Systems Engineering, TdSE 2020*, 2020.

[56] O. C. Eichmann, S. Melzer, F. Giertzsch, and R. God. Stakeholder Needs and Requirements Definition During Service Development in a System of Systems. In *2020 IEEE International Systems Conference (SysCon)*, pages 1–8. IEEE, 2020.

[57] O. C. Eichmann, S. Melzer, and R. God. Model-based Development of a System of Systems Using Unified Architecture Framework (UAF): A Case Study. In *2019 IEEE International Systems Conference (SysCon)*, pages 1–8. IEEE, 2019.

[58] O. C. Eichmann, S. Melzer, M. Hanna, R. God, and D. Krause. A Model-Based Approach for the Development of Modular Product Families Considering Different Life Phases. In *EMEASEC2018/TdSE2018*, 2018.

[59] O. C. Eichmann, S. Melzer, H. Wang, and R. God. Project report for the research project “4.0-Fähigkeit von KMU in der n-Tier- Lieferkette und bei MRO-Prozessen: Teilvorhaben: Entwicklung der Methoden und Werkzeuge zur Spezifikation und Auslegung Industrie-4.0-fähiger Prozesse bei kleinen und mittleren Unternehmen in der Luftfahrt”.

[60] O. C. Eichmann, C. Witte, J. P. Speichert, and R. God. Rapid Prototyping of a Graphical User Interface for Dispersed Services in Cyber-physical Systems of Systems. In *Tag des Systems Engineering (TdSE 2019)*, 2019.

[61] O. C. Eichmann, C. Witte, J. P. Speichert, and R. God. Rapid Prototyping of Graphical User Interfaces in a Message Broker Context. In *SAE International 2019 AeroTech Europe Conference, AEROTECHEUR 2019*, 2019.

[62] H. Eisner, J. Marciniak, and R. McMillan. Computer-Aided System of Systems (S2) Engineering. In *1991 IEEE International Conference on Systems, Man, and Cybernetics*, pages 531–537, New York and Piscataway, NJ, 1991. Institute of Electrical and Electronics Engineers and Available from IEEE Service Center.

[63] S. Engell. Cyber-physical Systems of Systems — Definition and core research and innovation areas. *European Roadmap on Research and Innovation in Engineering and Management of Cyber-Physical Systems of Systems*, 111, 2014.

[64] S. Engell, R. Paulen, M. A. Reniers, C. Sonntag, and H. Thompson. Core Research and Innovation Areas in Cyber-Physical Systems of Systems. In *International Workshop on Design, Modeling, and Evaluation of Cyber Physical Systems*, pages 40–55, 2015.

[65] J. A. Estefan et al. Survey of Model-Based Systems Engineering (MBSE) Methodologies. *Incose MBSE Focus Group*, 25(8):1–12, 2007.

[66] J. Feldhusen and K.-H. Grote. *Pahl/Beitz Konstruktionslehre: Methoden und Anwendung erfolgreicher Produktentwicklung*. Springer Vieweg, Berlin, Heidelberg, 8th edition, 2013.

[67] J. L. Fernandez and C. Hernandez. *Practical Model-Based Systems Engineering*. Artech House, Norwood, Massachusetts, 2019.

[68] K. Forsberg and H. Mooz. The Relationship of System Engineering to the Project Cycle. In *INCOSE International Symposium*, volume 1, pages 57–65. Wiley Online Library, 1991.

[69] S. Friedenthal, A. Moore, and R. Steiner. *A Practical Guide to SysML*. Elsevier Science, 2014.

[70] G. Friedmann. *La crise du progrès - esquisse d'histoire des idées, 1895-1935*. Gallimard, Paris, 1936.

[71] E. Geisberger and M. Broy. *agendaCPS: Integrierte Forschungsagenda Cyber-Physical Systems*, volume 1. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[72] F. Giertzsch, O. C. Eichmann, H. Hintze, and R. God. An Approach for a Simulation-Based Analysis of Business Processes Using the Systems Modeling Language (SysML). In *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, pages 331–340, 2022.

[73] F. M. Giertzsch, H. Hintze, B. Heinke, and R. God. Network Design Criteria to Introduce Data Analytics Within the Aircraft Cabin. In *7th International Workshop on Aircraft System Technologies,(AST 2019)*, pages 363–372. Springer, 2019.

[74] R. God. Project description for the research project “Entwicklung und Nutzung einer intelligenten digitalen Passagierflussmessung vom Flughafeneingang bis zum Erreichen des Fluggastsitzes”.

[75] F. Harashima, M. Tomizuka, and T. Fukuda. Mechatronics - "What Is It, Why, and How?" An editorial. *IEEE/ASME Transactions on Mechatronics*, 1(1):1–4, 1996.

[76] H. Hintze, F. Giertzsch, and R. God. Design Approach for Secure Networks to Introduce Data Analytics within the Aircraft Cabin. *SAE International Journal of Advances and Current Practices in Mobility*, 2(2019-01-1853):737–746, 2019.

[77] H. Hintze and R. God. Using Model-Based Security Engineering in the Development of Complex Aircraft Cabin Systems. *SAE International Journal of Aerospace*, 8(1):89–96, 2015.

[78] H. Hintze, J. P. Speichert, and R. God. The Risk Matrix as an integral part of a SysML-based Security Engineering Approach in the Development of Complex Aircraft Cabin Systems. In *2018 IEEE/AIAA 37th Digital Avionics Systems Conference (DASC)*, pages 1–9. IEEE, 2018.

[79] J. Holt and S. Perry. *SysML for Systems Engineering: A Model-Based Approach*, volume 10 of *Professional Applications of Computing Series*. Institution of Engineering and Technology, London, 2nd edition, 2014.

[80] J. Holt, S. Perry, M. Brownsword, D. Cancila, S. Hallerstede, and F. O. Hansen. Model-based requirements engineering for system of systems. In *2012 7th International Conference on System of Systems Engineering (SoSE)*, pages 561–566, 2012.

[81] J. Holt, S. A. Perry, and M. Brownsword. *Model-Based Requirements Engineering*, volume 9 of *IET Professional applications of computing series*. Institution of Engineering and Technology, Stevenage, 2012.

[82] International Organization for Standardization. Information technology – Message Queuing Telemetry Transport (MQTT) v3.1.1. Standard, 2016.

[83] M. Jackson and M. Wilkerson. MBSE-driven visualization of requirements allocation and traceability. In *2016 IEEE Aerospace Conference*, pages 1–17, 2016.

[84] M. Jamshidi. *System of Systems Engineering: Innovations for the 21st Century*. Wiley Series in Systems Engineering and Management. Wiley, Hoboken N.J., 2009.

[85] R. M. Jaradat, C. B. Keating, and J. M. Bradley. A histogram analysis for system of systems. *International Journal System of Systems Engineering*, 5(3):193–227, 2014.

[86] J. G. Lamm and T. Weilkiens. Funktionale Architekturen in SysML. In *Tag des Systems Engineering*, pages 109–118, 2010.

[87] J. G. Lamm and T. Weilkiens. Method for Deriving Functional Architectures from Use Cases. *Systems Engineering*, 17(2):225–236, 2014.

[88] G. A. Lewis, E. Morris, P. Place, S. Simanta, and D. B. Smith. Requirements Engineering for Systems of Systems. In *3rd Annual IEEE Systems Conference, 2009*, pages 247–252, Piscataway, NJ, 2009. IEEE.

[89] M. W. Maier. Architecting Principles for Systems-of-Systems. *Systems Engineering*, 1(4):267–284, 1998.

[90] S. Melzer, O. C. Eichmann, H. Wang, and R. God. Modeling and Simulation of Database Interactions. In *Tag des Systems Engineering, TdSE 2021*, 2021.

[91] S. Melzer, O. C. Eichmann, H. Wang, and R. God. Simulation of Database Interactions for Early Validation of Digitized Enterprise Processes. *Procedia Computer Science*, 219:658–665, 2023. CENTERIS – International Conference on ENTERprise Information Systems / ProjMAN – International Conference on Project MANagement / HCist – International Conference on Health and Social Care Information Systems and Technologies 2022.

[92] S. Melzer, J. P. Speichert, O. C. Eichmann, and R. God. Simulating Cyber-Physical Systems Using a Broker-Based SysML Toolbox. In *7th International Workshop on Aircraft System Technologies (AST 2019)*, pages 411–420, 2019.

[93] B. Meyer. Applying 'design by contract'. *Computer*, 25(10):40–51, 1992.

[94] C. Ncube and S. L. Lim. On Systems of Systems Engineering: A Requirements Engineering Perspective and Research Agenda. In *2018 IEEE 26th International Requirements Engineering Conference (RE)*, pages 112–123. IEEE, 2018.

[95] P. H. Nguyen, S. Ali, and T. Yue. Model-Based Security Engineering for Cyber-Physical Systems: A Systematic Mapping Study. *Information and Software Technology*, 83:116–135, 2017.

[96] S. Oppermann, J. Hoth, and R. God. Project report for the research project "Lernendes Galley-Catering-System (LGCS): Teilvorhaben: Entwurfsmethoden und Architekturen für lernende, künstlich intelligente Systeme in der Kabine".

[97] C. Pautasso, E. Wilde, and R. Alarcon. *REST: Advanced Research Topics and Practical Applications*. Springer, New York, 2013.

[98] F. Rahimi and A. Ipakchi. Demand Response as a Market Resource Under the Smart Grid Paradigm. *IEEE Transactions on Smart Grid*, 1(1):82–88, 2010.

[99] E. Rauch, D. T. Matt, C. A. Brown, W. Towner, A. Vickery, and S. Santiteerakul. Transfer of Industry 4.0 to Small and Medium Sized Enterprises. *Transdisciplinary Engineering Methods for Social Innovation of Industry*, 4:63–71, 2018.

[100] M. Roscia, M. Longo, and G. C. Lazaroiu. Smart City by multi-agent systems. In *2013 International Conference on Renewable Energy Research and Applications (ICRERA)*, pages 371–376, 2013.

[101] W. W. Royce. Managing the Development of Large Software Systems: Concepts and Techniques. In *Proceedings of the 9th International Conference on Software Engineering*, pages 328–338, 1987.

[102] A. P. Sage. *Handbook of Systems Engineering and Management*. John Wiley & Sons, Chicester, 2nd edition, 2011.

[103] B. Sauser, J. Boardman, and D. Verma. Systomics: Toward a Biology of System of Systems. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 40(4):803–814, 2010.

[104] G. Shea, editor. *NASA Systems Engineering Handbook*. 2nd edition, 2017.

[105] H. Song, D. B. Rawat, S. Jeschke, and C. Brecher. *Cyber-Physical Systems: Foundations, Principles and Applications*. Morgan Kaufmann, 2016.

[106] A. Taghiyar, O. C. Eichmann, S. Lammers, and R. God. Entwurf und Analyse eines Montageprozesses unter Verwendung einer MBSE-CAD/CAM Werkzeugkette am Beispiel einer Montagelinie mit Mensch-Roboter-Kollaboration. *Tag des Systems Engineering 2022: Tagungsband Paderborn, 16.-18. November 2022*, 20:45, 2022.

[107] A. Tukker. Eight Types of Product–Service System: Eight Ways to Sustainability? Experiences from SusProNet. *Business Strategy and the Environment*, 13(4):246–260, 2004.

[108] D. D. Walden, G. J. Roedler, K. Forsberg, R. D. Hamelin, and T. M. Shortell, editors. *Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities*. Wiley, Hoboken, NJ, 4th edition, 2015.

[109] T. Weilkiens. *Systems Engineering mit SysML/UML*. dpunkt, Heidelberg, 3rd edition, 2014.

[110] T. Weilkiens, J. G. Lamm, S. Roth, and M. Walker. *Model-based System Architecture*. Wiley Series in Systems Engineering and Management. John Wiley & Sons, Inc, Hoboken, New Jersey, 2016.

[111] L. Zhang. Modeling Large Scale Complex Cyber Physical Control Systems Based On System of Systems Engineering Approach. In *2014 20th International Conference on Automation and Computing*, pages 55–60. IEEE, 2014.

[112] D. Ziegler. *Die Industrielle Revolution*. Wissenschaftliche Buchgesellschaft in Herder, 2012.

List of Abbreviations

ACRE	Approach to Context-based Requirements Engineering
AMADEOS	Architecture for Multi-criticality Agile Dependable Evolutionary Open System-of-Systems
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
ARP	Aerospace Recommended Practice
CCA	Common Cause Analysis
CPS	Cyber-physical system
CPSoS	Cyber-physical System of Systems
CS	Constituent System
DoD	Department of Defense
DoDAF	Department of Defense Architecture Framework
EASA	European Union Aviation Safety Agency
ERP	Enterprise Resource Planning
eID	Electronic Identification
FAA	Federal Aviation Administration
FAS	Functional Architectures for Systems
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
INCOSE	International Council on Systems Engineering
ISE&PPOOA	Integrated Systems Engineering and Pipelines of Processes in Object-Oriented Architectures
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
IT	Information Technology
KPI	Key Performance Indicator
LCC	Low-Cost Carrier
LTE	Long Term Evolution

MODAF	Ministry of Defence Architecture Framework
MQTT	Message Queuing Telemetry Transport
MRO	Maintenance, Repair and Overhaul
OASIS	Organization for the Advancement of Structured Information Standards
OMG	Object Management Group
OOSEM	Object-Oriented Systems Engineering Method
PASA	Preliminary Aircraft Safety Assessment
PSS	Product-Service System
PSSA	Preliminary System Safety Assessment
REST	Representational State Transfer
RPC	Remote Procedure Call
SAE	Society of Automotive Engineers
SoI	System of Interest
SoS	System of Systems
SoSE	System of Systems Engineering
SYSMOD	Systems Modeling Toolbox
UAF	Unified Architecture Framework
UPDM	Unified Profile for Department of Defense Architecture Framework and Ministry of Defence Architecture Framework
VDI	Verein Deutscher Ingenieure (*engl.* Association of German Engineers)
VSM	Viable System Model
XML	Extensible Markup Language

List of Figures

1.1 Informatization in aeronautics: Digitally supported passenger journey and passenger connectivity in the aircraft cabin. Sources: Travel Planning by Rawpixel.com/Shutterstock.com; Boarding by Jaromir Chalabala/ Shutterstock.com; IFE and Connectivity by GaudiLab/Shutterstock.com 7
1.2 In today's aviation Cyber-physical Systems are merging into one extensive networking Cyber-Physical System of Systems (CPSoS) [51,63,64,84]. Sources: Travel Planning by Rawpixel.com/Shutterstock.com; Boarding by Jaromir Chalabala/Shutterstock.com; Automated Catering Inventory by g_dasha/ Shutterstock.com; IFE and Connectivity by GaudiLab/Shutterstock.com; Production Automation by Mykola Holyutyak/Shutterstock.com and Jaromir Chalabala/Shutterstock.com; Augmented Reality, Autonomous Logistics, and Predictive Maintenance by Gorodenkoff/Shutterstock.com; Background by Sunward Art/Shutterstock.com 8
1.3 Unified Architecture Framework for Enterprise Architecture [8] 10
1.4 The V model for the development of mechatronic systems [3] 11
1.5 A methodology is the combination of processes, methods, and tools [3] . . . 12

2.1 Industrial revolutions, illustration by [13] 15
2.2 Time periods in SoS research 17
2.3 SoS types, arranged according to increasing managerial and operational independence [94] 20
2.4 SysML block definition diagram with a concept for the relationships between SoS, CPS, and CPSoS types [53] 25
2.5 Common elements in SysML block definition diagrams [12] 25
2.6 PSS categories according to Tukker [107] 26
2.7 Analyzed system types and related process models that are illustrated schematically. The VDI 2221 is presented in more detail in Figure 2.8 on page 29, the V model in Figure 1.4 on page 11 and in more detail on page 31, the waterfall model in Figure 2.9 on page 30, the spiral model in Figure 2.12 on page 34, and the Unified Architecture Framework in Figure 1.3 on page 10. 28
2.8 Systematic engineering design of technical systems and products according to VDI 2221 [10] 29
2.9 Waterfall model according to Royce [101] 30
2.10 V model for verification and validation through the software life cycle according to Boehm [42] 31
2.11 V model for the development of mechatronic systems and CPSs according to the newest version of the VDI/VDE standard 2206 from the year 2021 [16] . . 33
2.12 Spiral model [43] 34
2.13 Development Life Cycle [6] 35
2.14 CPSoS Research Needs Identification 37

3.1 Document-based requirements analysis [34] 39
3.2 Relationships between standards for system life cycle processes and SoS considerations [23] 42

3.3 SYSMOD analysis process by Weilkiens [109], modeled in a SysML activity diagram . . . 45
3.4 Common elements in SysML activity diagrams [12] . . . 45
3.5 Process for stakeholder needs analysis by Friedenthal et al. [69], modeled in a SysML activity diagram . . . 47
3.6 Approach to Context-based Requirements Engineering . . . 49
3.7 COMPASS ontology [80], modeled in a SysML block definition diagram . . . 50
3.8 Subset of the MBSE ontology by [79], modeled in a SysML block definition diagram . . . 51
3.9 Process for Requirements Engineering for SoS by Lewis et al. [88], modeled in a SysML activity diagram . . . 53
3.10 UAF grid [19] . . . 54
3.11 Scope of approaches for functional analysis and architecture design . . . 58
3.12 ISO/IEC/IEEE 15288:2015 process for architecture definition [21], modeled in a SysML activity diagram . . . 59
3.13 FAS process [86], modeled in a SysML activity diagram . . . 62
3.14 Logical architecture definition according to OOSEM [69], modeled in a SysML activity diagram . . . 64
3.15 ISE&PPOOA process for architecture development [67], modeled in a SysML activity diagram . . . 66
3.16 Allocation of functional elements to CPSs in a logical CPSoS architecture draft 70
3.17 Allocation of a functional element to CPSs in an alternative logical CPSoS architecture draft . . . 71
3.18 Modeling logical architectures according to SYMOD [110], modeled in a SysML activity diagram . . . 71
3.19 Allocation of actions to logical elements according to SYSMOD [110], modeled in a SysML activity diagram . . . 72
3.20 Segment of OOSEM for the development of physical architecture alternatives [69], modeled in a SysML activity diagram . . . 73
3.21 Evaluation of architecture alternatives [69], modeled in a SysML activity diagram . . . 74
3.22 Derivation and evaluation of architecture alternatives [66], modeled in a SysML activity diagram . . . 75
3.23 Communication paths in a CPSoS . . . 78
3.24 REST API structure [97] . . . 79
3.25 AMQ Model components . . . 81
3.26 Broker Federation . . . 82
3.27 Life cycle phases according to the ARP 4754A [6], the ISO/IEC/IEEE standard 15288 [21], and the life cycle phases for commercial aircraft programs by Altfeld [31] . . . 83

4.1 Enhanced V model for CPSoS development and transition to CPS development . . . 85
4.2 SysML activity diagram of the process for CPSoS Mission and Operational Scenarios Definition . . . 87
4.3 Mission and business requirements definition for the intelligent catering CPSoS, modeled in a SysML requirements definition diagram . . . 88

4.4 Diagram segment of the stakeholder context for the intelligent catering CPSoS, modeled in a SysML block definition diagram. Common elements in block definition diagrams are explained in Figure 2.5 . . . 89
4.5 Diagram section of the system context for the intelligent catering CPSoS, modeled in a SysML block definition diagram. Common elements in block definition diagrams are explained in Figure 2.5 . . . 89
4.6 Relevant source elements for the intelligent catering CPSoS, modeled in a SysML block definition diagram . . . 91
4.7 Information network structure for the intelligent catering CPSoS, modeled in a SysML internal block diagram . . . 92
4.8 Common elements in SysML internal block diagrams [12] . . . 92
4.9 Operational scenario for passenger provisioning in the intelligent catering CPSoS, modeled in a SysML activity diagram . . . 93
4.10 Use cases for the intelligent catering CPSoS, modeled in a SysML use case diagram . . . 94
4.11 Traceability between missions, operational scenarios, and use cases, modeled in a SysML requirements diagram . . . 95
4.12 KPI definition for the intelligent catering CPSoS, modeled in a SysML block definition diagram . . . 96
4.13 SysML activity diagram of the process for CPSoS Function Identification and Architecture Development . . . 97
4.14 CPSoS use case activity, modeled in a SysML activity diagram . . . 98
4.15 Segment of functional grouping of CPSoS use case activities in a dependency matrix of Cameo Systems Modeler . . . 100
4.16 Segment of the functional CPSoS architecture of the intelligent catering CPSoS, modeled in a SysML internal block diagram . . . 101
4.17 Exemplary logical CPSoS architecture draft for the intelligent catering CPSoS, modeled in a SysML internal block diagram . . . 103
4.18 Segment of an alternative logical CPSoS architecture draft for the intelligent catering CPSoS, modeled in a SysML internal block diagram . . . 103
4.19 SysML activity diagram of the process for CPSoS Architecture Evaluation . . 104
4.20 SysML block definition diagram for evaluating logical architecture drafts . . . 105
4.21 Logical CPSoS architecture evaluation in a SysML parametric diagram 106
4.22 Logical CPSoS architecture evaluation results in a generic table of Cameo Systems Modeler . . . 108
4.23 SysML activity diagram of the process for CPS Function Identification and Functional Architecture Development . . . 109
4.24 CPS-human and CPS-machine use cases for functional CPS architecture development, modeled in a SysML use case diagram . . . 110
4.25 Use case activity for the cps-machine use case “identify non-consumed items,” modeled in a SysML activity diagram . . . 111
4.26 Functional grouping in a dependency matrix of Cameo Systems Modeler . . 112
4.27 Functional blocks of the smart trolley, modeled in a SysML block definition diagram . . . 113
4.28 Segment of the functional CPS architecture of the smart trolley, modeled in a SysML internal block diagram . . . 113

4.29 SysML activity diagram of the process for CPSoS Communication Architecture Implementation . . . 115
4.30 Components for communication architecture design using message broker communication, modeled in a SysML block definition diagram . . . 117
4.31 Communication architecture draft, modeled in a SysML block definition diagram . . . 120
4.32 Smart trolley definition in a SysML block definition diagram . . . 122
4.33 Simulation of smart trolley behavior using a SysML activity diagram . . . 123
4.34 Overview of communication architecture design and integration, modeled in SysML block definition diagrams and SysML activity diagrams . . . 124
4.35 Contribution of the methods to the design of CPSoS . . . 126

5.1 The methods proposed in this thesis are applicable to acknowledged and collaborative SoS according to the classification in the ISO/IEC/IEEE standard 21841:2019 [24] and the graphical representation by Ncube and Lim [94] . . . 128
5.2 Application of the CPSoS Mission and Operational Scenarios Definition method . . . 132
5.3 Initial concept for identifying passengers in the aircraft cabin . . . 133
5.4 Alternative concept for identifying passengers in the aircraft cabin . . . 133
5.5 Traceability between pain points, business requirements, and use cases in a relation map of Cameo Systems Modeler . . . 134
5.6 Application of the CPSoS Function Identification and Architecture Development method . . . 136
5.7 Logical Architecture for the improvement of passenger flow control and seamless travel experience, modeled in a SysML internal block diagram . . . 137
5.8 First draft of the logical CPSoS architecture for the data-driven catering CPSoS in a SysML internal block diagram . . . 139
5.9 Application of the CPSoS Architecture Evaluation method . . . 142
5.10 Parametric diagram for the technological evaluation of logical CPSoS architecture drafts for the data-driven catering CPSoS, modeled in a SysML parametric diagram . . . 143
5.11 An alternative logical CPSoS architecture draft for the data-driven catering CPSoS does not integrate a message broker into the galleys. Instead, galley services are connected to the message broker of the cabin server. . . . 144
5.12 Application of the CPS Function Identification and Functional Architecture Development method . . . 146
5.13 Segment of the functional CPS architecture for the visualization service for presentation of catering and beverage consumption, modeled in a SysML internal block diagram . . . 147
5.14 Illustrative implementation of a visualization service for presentation of catering and beverage consumption . . . 148
5.15 Application of the CPSoS Communication Architecture Implementation method . . . 150
5.16 Segment of the communication architecture for message brokers in the galley and the cabin server, modeled in a SysML block definition diagram. Another segment is presented in the diagram in the appendix A.1 on page 185. . . . 152

5.17 Segment of the communication architecture for inter-enterprise communication for the exchange of orders and order confirmations, modeled in a SysML block definition diagram 154

6.1 Further disciplines for the CPSoS design [52]. A more detailed graphic is presented in the appendix A.3 on page 188. 165

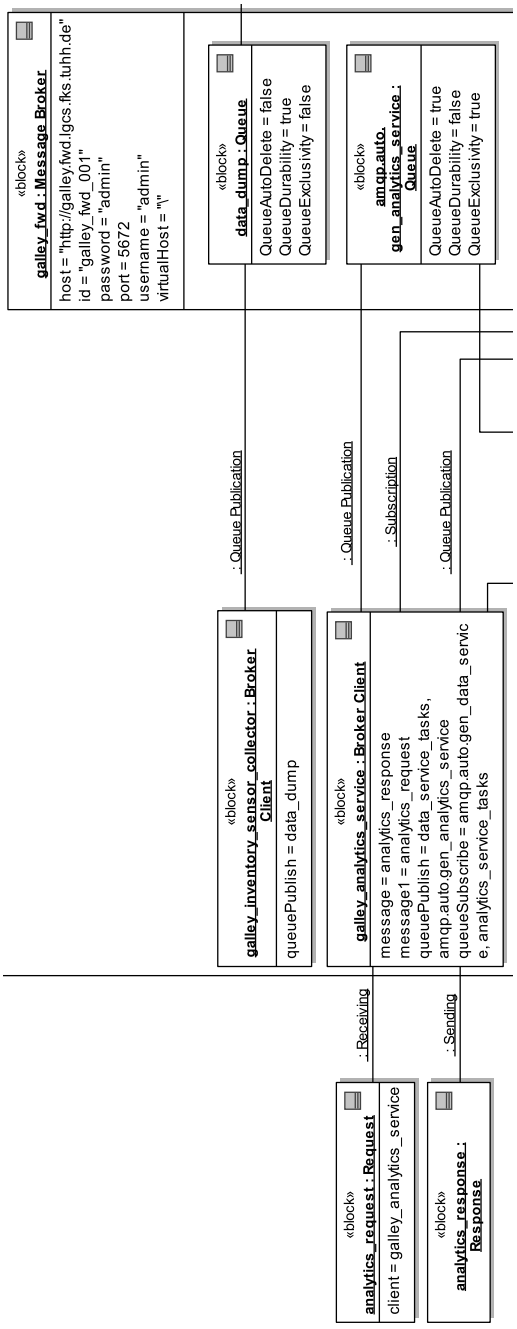
List of Tables

3.1 Evaluation of methods with respect to suitability for the concept phase in CPSoS development 55
3.2 Evaluation of methods with respect to architecture development in CPSoS . . 69
3.3 Evaluation of methods with respect to the design phase of CPSoS 77

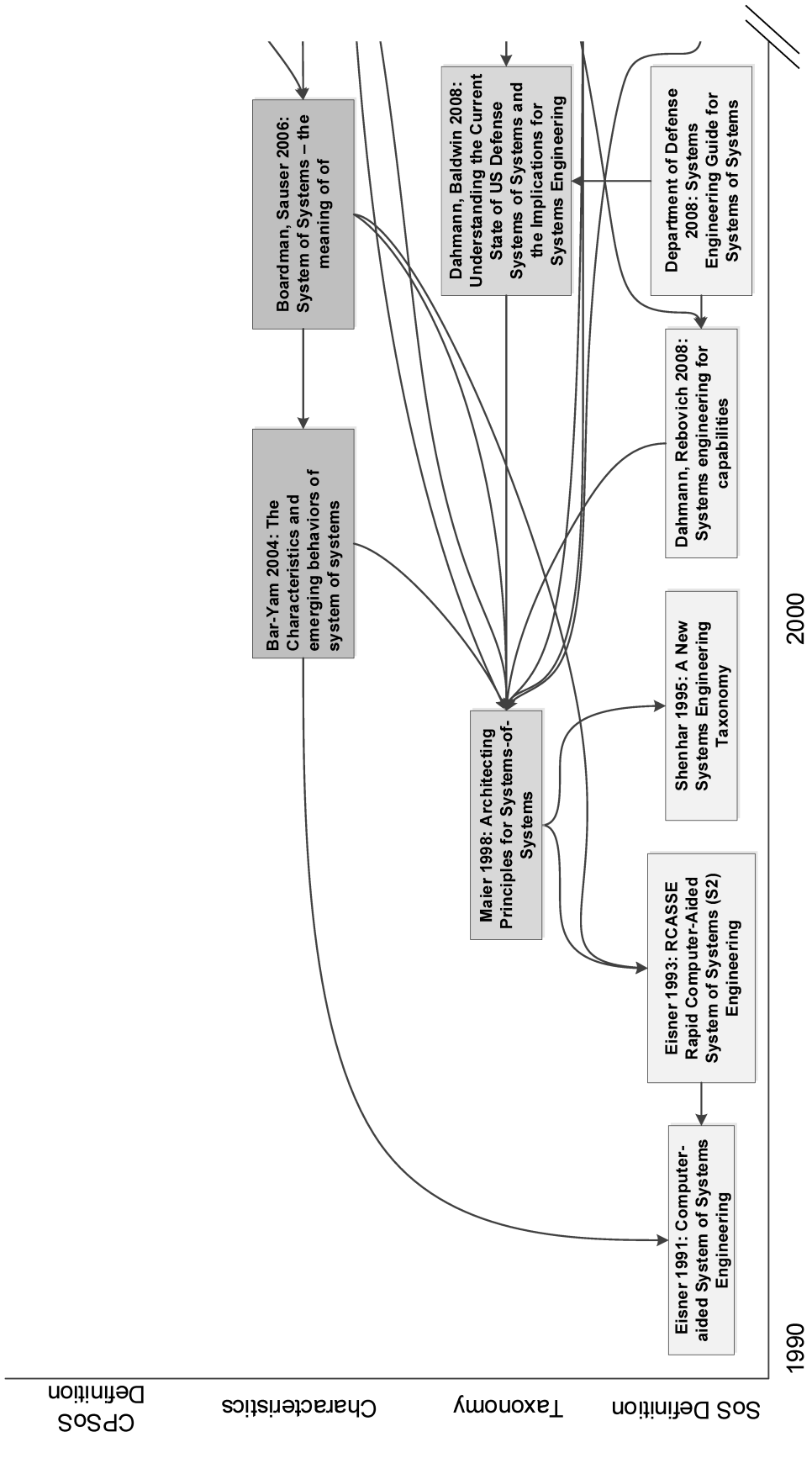
5.1 Validation results 156

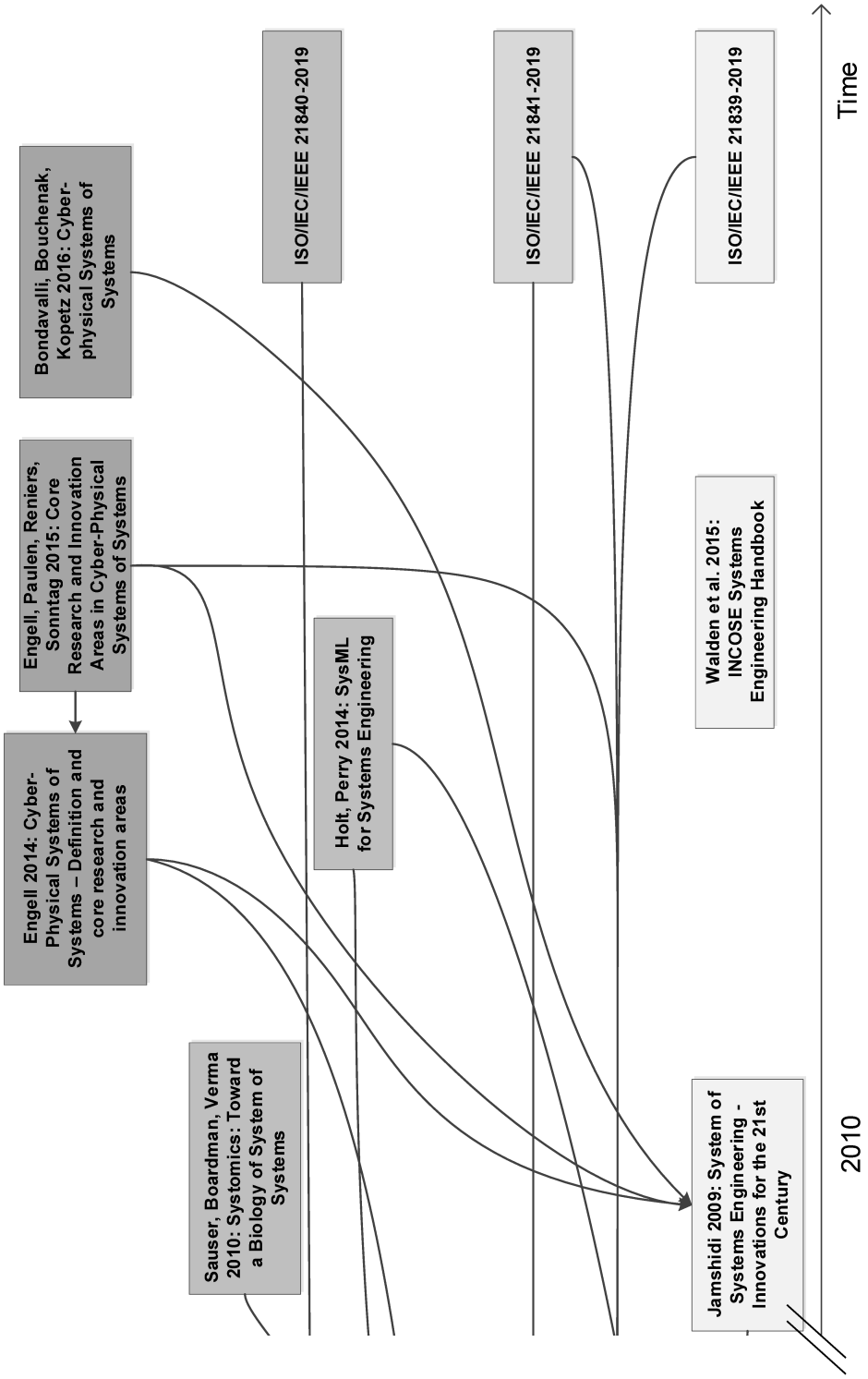
Appendix A

A.1 Communication Architecture for Message Brokers in the Galley and the Cabin Server



A.2 Chronology of SoS Definitions, Taxonomy, Characteristics, and CPSoS Definition in Literature




Time
2010
Bondavalli, Bouchenak, Kopetz 2016: Cyber-physical Systems of Systems
Engell, Paulen, Reniers, Sonntag 2015: Core Research and Innovation Areas in Cyber-Physical Systems of Systems
Engell 2014: Cyber-Physical Systems of Systems – Definition and core research and innovation areas
Sauser, Boardman, Verma 2010: Systomics: Toward a Biology of System of Systems
Holt, Perry 2014: SysML for Systems Engineering
ISO/IEC/IEEE 21840-2019
ISO/IEC/IEEE 21841-2019
ISO/IEC/IEEE 21839-2019
Walden et al. 2015: INCOSE Systems Engineering Handbook
Jamshidi 2009: System of Systems Engineering - Innovations for the 21st Century

A.3 Further Disciplines for the CPSoS Design