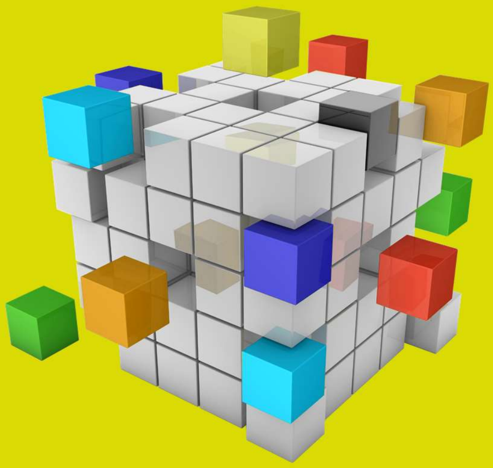


Whitepaper

ARCHITECTURE-CENTRIC, MODEL-BASED SYSTEMS ENGINEERING (ACMBSE)

VERSION	DATUM	AUTOR	ÄNDERUNG(EN)
1.2	21.08.2024	Enrico Seidel	Translation from 1.2 GE

INHALT

SUMMARY

In the past ten years, the automotive industry has significantly transformed from traditional document-based systems engineering to a more architecture-centered and model-based systems engineering (MBSE). This change is driven by the rapidly increasing complexity and the challenges of managing system descriptions with conventional document-based methods. This white paper explores how MBSE can support this transformation using the architecture framework developed by the author. The author proposes an architecture-centered approach to systems engineering based on various viewpoints. Focusing on architecture centrality and using models allows for more efficient and effective complex system analysis, development, and design. The author also uses an example from the automotive sector to demonstrate how this approach can be practically integrated into a development workflow.

1 INTRODUCTION

Like other industrial sectors, the automotive industry faces challenges in today's rapidly changing environment concerning productivity, profitability, and human capital (Nayak et al., 2022). The transition to a software-based ecosystem has required automotive manufacturers to achieve significant technological advancements in multiple functional areas (Burkacky et al., 2021). The growing demand for enhanced system capabilities and efficiency in development is driving the use of system models. Traditional document-based systems engineering is no longer sufficient to manage the complexity of software-driven functions in resilient and safety-critical systems. Instead, an MBSE approach is necessary to facilitate compliance with critical standards such as ISO 26262 (ISO 1 2018).

The author of this article takes it a step further by placing architecture at the forefront. Traditional document-based systems engineering has proven inadequate for modern complex systems such as automotive systems and components. Consequently, an architecture-centered model-based approach is gaining importance, referred to in this paper as Architecture-Centered Model-Based Systems Engineering (ACMBSE) (Brooks et al., 2007). ACMBSE provides a means to ensure the design and implementation of system behavior through rigorous models and simulations, making it an essential methodology for organizations like automotive suppliers to adapt to the changing demands of the automotive industry.

This publication examines the adoption of ACMBSE and demonstrates how this approach can be used for system analysis, functional development, the design of logical elements, and technical system design, extending to the implementation of electronic control units, including their physical realization.

2 ARCHITECTURE-CENTRIC MBSE

ACMBSE is a systems engineering approach that strongly emphasizes the central role of system architecture in the development of complex systems, as illustrated in Figure 1. ACMBSE is based on the principle that the architecture of a system and all its subsystems provides a coherent overview of the system, including its representations of the components, their interrelationships, and the overall system behavior, encompassing both software and hardware aspects. Therefore, ACMBSE acts as a bridge to heterogeneous, domain-specific engineering disciplines. This approach further requires that all other aspects of systems engineering, such as requirements, functional safety, security and privacy, system variability, reliability, verification, and validation, be aligned with the architecture and contribute to its viewpoints.

Compared to document-based development, ACMBSE offers several advantages: improved communication, information maintainability, enhanced ability to manage system complexity, increased product quality, and excellent knowledge retention.

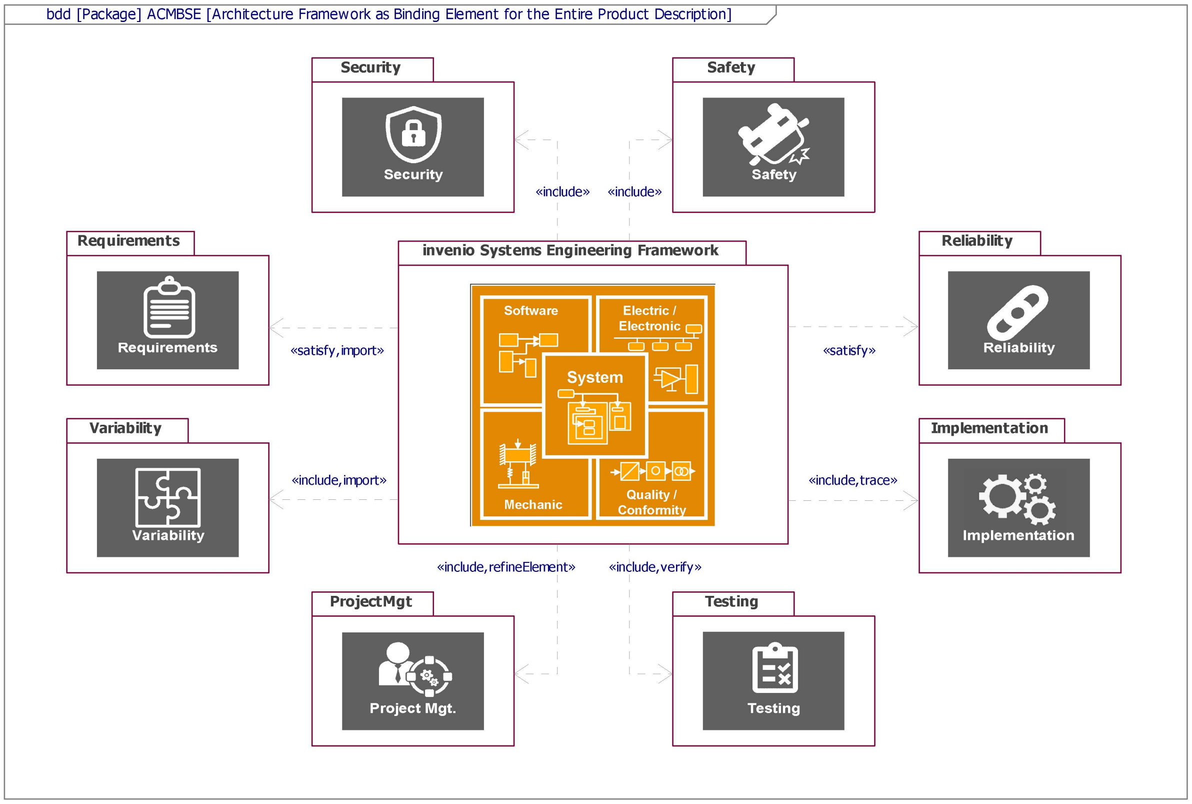


Figure 1: The Architecture Framework as a Link for the Overall Product Description

3 THE IMPORTANCE OF AN ARCHITECTURE FRAMEWORK

SysML (OMG 2018) is a widely used language in systems engineering that enables model-based development. However, it does not define rules and methods for building system architecture and design within a specific domain (ISO 2022). To achieve a cross-domain platform, it is crucial to have a standardized set of methods and tools for developing system products and their architecture artifacts.

The use of structured frameworks such as Harmony (Douglas 2017), OOSEM (Friedenthal et al., 2015), or MOFLT (Ducamp et al., 2022) is essential for the consistent implementation of MBSE on a large scale. Each framework has its strengths within domain-focused architecture frameworks, including UPDM (UPDM 2017), which combines DoDAF (Department of Defense Architecture Framework) and MoDAF (Ministry of Defense Architecture Framework). However, the author recognized that these frameworks, primarily developed for defense, did not fully meet the specific requirements of the automotive industry. Challenges such as reusability, automated model verification, systematic use of viewpoints on the model, system abstraction levels, and hierarchy depth as prerequisites for the ACMBSE approach were not entirely addressed. Even frameworks like TOGAF (TOGAF 10) and Zachman (Zachman 2023), which aim to capture enterprise perspectives, would require adjustments to meet the technical aspects of the automotive industry. In this context, only the SYSMOD (Weilkiens 2020) framework proved suitable and met the author's needs for tailored viewpoints and modeling techniques.

The author intended to use his framework primarily for the automotive sector and recognized the need to include the criteria mentioned above for holistic system design, including developing electronic control units and components.

For these reasons, the author previously developed an architecture framework (CAF) at Continental AG, which, with the transition to invenio AG, has now been significantly further developed and is to be published as the invenio Systems Engineering Framework (iSEF). Recently, this framework has evolved into a mature modeling methodology, including language and tool set, defining all the necessary methods and process steps to seamlessly develop architecture and design specifications for systems, software, and hardware engineers. In addition to the usual rules and methods that extend the SysML language, iSEF is intended to provide a comprehensive set of predefined architecture artifacts, pre-developed design elements, and reusable functions, supporting the development of standardized automotive functions and platforms with shorter time-to-market. Furthermore, iSEF ensures full compliance with all essential ISO standards, such as ISO 42010 (ISO 2022) and ISO 15288 (ISO 2015).

For implementing iSEF, IBM Rhapsody (IBM 2022) was chosen as the modeling tool. It is embedded in the IBM Engineering Lifecycle Management (ELM) platform and uses a fully customized domain-specific profile based on SysML.

4 THE 3 DIMENSIONS OF A SYSTEM ARCHITECTURE

As systems become increasingly complex (INCOSE 2022), it becomes difficult for a single system engineer to understand its functionality, properties, and performance comprehensively. Therefore, systems must be divided and organized into smaller units or subsystems. Each subsystem has a specific task, allowing architects and designers to focus on the details of that particular part. Examples of such complex systems include vehicles and airplanes, where many organizational units work on different parts of the system to create the overall system architecture.

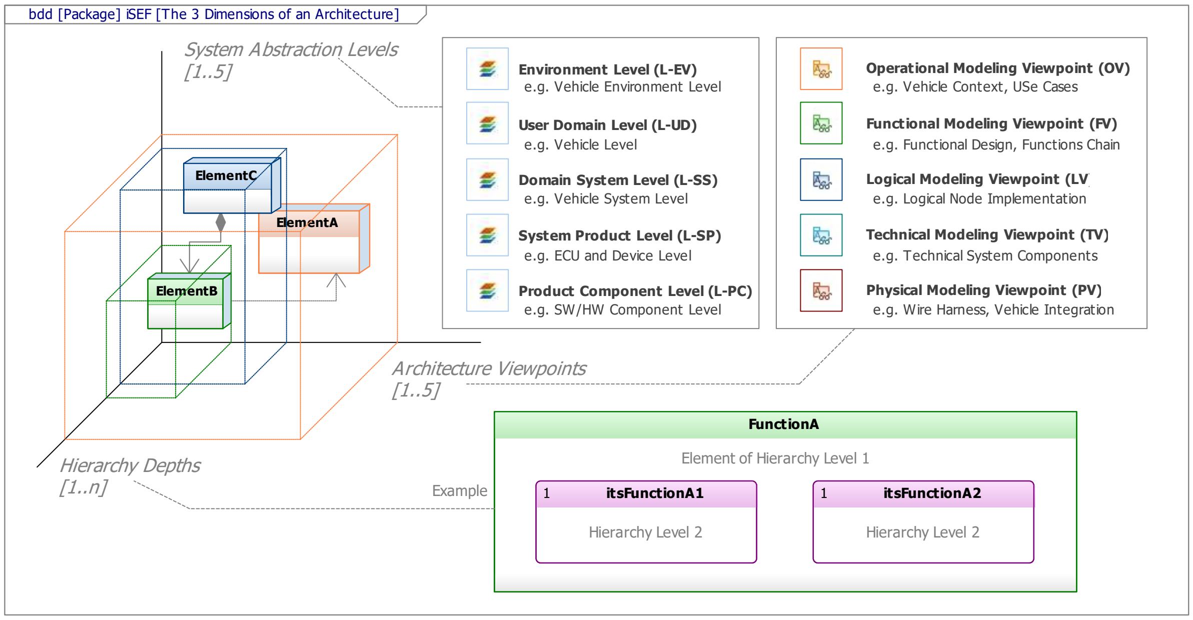
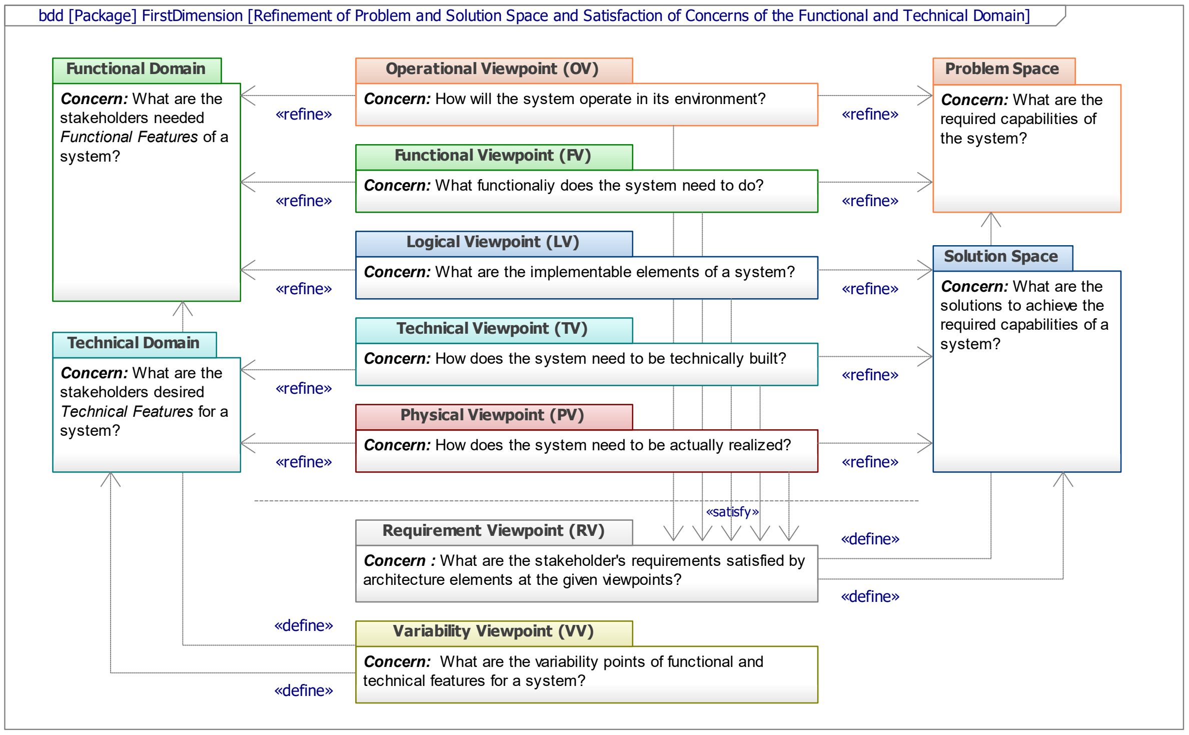

bdd [Package] iSEF [The 3 Dimensions of an Architecture]
System Abstraction Levels [1..5]
ElementC
ElementA
ElementB
Architecture Viewpoints [1..5]
Hierarchy Depths [1..n]
Environment Level (L-EV)
e.g. Vehicle Environment Level
User Domain Level (L-UD)
e.g. Vehicle Level
Domain System Level (L-SS)
e.g. Vehicle System Level
System Product Level (L-SP)
e.g. ECU and Device Level
Product Component Level (L-PC)
e.g. SW/HW Component Level
Operational Modeling Viewpoint (OV)
e.g. Vehicle Context, USe Cases
Functional Modeling Viewpoint (FV)
e.g. Functional Design, Functions Chain
Logical Modeling Viewpoint (LV
e.g. Logical Node Implementation
Technical Modeling Viewpoint (TV)
e.g. Technical System Components
Physical Modeling Viewpoint (PV)
e.g. Wire Harness, Vehicle Integration
FunctionA
Element of Hierarchy Level 1
Example
1 itsFunctionA1
Hierarchy Level 2
1 itsFunctionA2
Hierarchy Level 2

Figure 2: The Three Dimensions of System Architecture Description

With the iSEF, the author introduces three dimensions of system architecture: the model viewpoints on the system, the system abstraction levels, and the hierarchy depth. These three orthogonal dimensions, as applied to the automotive domain, as shown in Figure 2, enable a structured approach to coherent system architecture development.

5 THE MODEL VIEWPOINTS OF ISEF

As part of developing a framework to meet the needs of a development organization, the author has identified five essential architecture-relevant model viewpoints (as shown in Figure 3) to analyze, define, and create a system with a strong focus on reusability. The emphasis is on addressing the concerns of various stakeholders. The operational viewpoint, functional viewpoint, logical viewpoint, technical viewpoint, and physical viewpoint of the model are introduced. The directly architecture-relevant viewpoints are defined as follows:



bdd [Package] FirstDimension [Refinement of Problem and Solution Space and Satisfaction of Concerns of the Functional and Technical Domain]
Functional Domain
Concern: What are the stakeholders needed Functional Features of a system?
Technical Domain
Concern: What are the stakeholders desired Technical Features for a system?
Operational Viewpoint (OV)
Concern: How will the system operate in its environment?
Functional Viewpoint (FV)
Concern: What functionaliy does the system need to do?
Logical Viewpoint (LV)
Concern: What are the implementable elements of a system?
Technical Viewpoint (TV)
Concern: How does the system need to be technically built?
Physical Viewpoint (PV)
Concern: How does the system need to be actually realized?
Requirement Viewpoint (RV)
Concern : What are the stakeholder's requirements satisfied by architecture elements at the given viewpoints?
Variability Viewpoint (VV)
Concern: What are the variability points of functional and technical features for a system?
Problem Space
Concern: What are the required capabilities of the system?
Solution Space
Concern: What are the solutions to achieve the required capabilities of a system?
«refine»
«satisfy»
«define»

Figure 3: Relationships Between Viewpoints and the Problem/Solution Space

Operational Viewpoint (OV):

As part of the problem space, the Operational Viewpoint (OV) focuses on analyzing the system to be developed in alignment with the operational concept (ISO 2 2018). It defines the System of Interest (SoI) within its contextual environment and identifies its subsystems and interfaces through ports, representing communication's logical and technical aspects. The primary objective of the OV is to provide a visual representation of the system processes and use cases, detailing the intended operations and behaviors of the system and refining them into system activities. Additionally, the OV outlines the sequences of interactions between the system elements, clarifying how communication occurs within the SoI being analyzed.

Functional Viewpoint (FV):

The Functional Viewpoint (FV), which is still part of the problem space, focuses on functional design following the system analysis. It defines consolidated and reusable function blocks that encapsulate behavior models based on activities and state machines (resulting from the operational analysis in the OV). The FV facilitates the definition of structured and standardized function interfaces with their exposed operations, which ease the flow of information between functions. Additionally, the FV allows for the visualization of function blocks according to predefined decomposition strategies, enabling a comprehensive understanding of the entire functional behavior of the System of Interest (SoI).

Logical Viewpoint (LV):

The Logical Viewpoint (LV) marks the transition to the solution space and focuses on implementing a targeted system, considering given constraints and platforms. It groups the function blocks from the FV into implementable units, known as logical nodes. A primary objective is visually representing the system from an implementation perspective, including the connections between nodes and the definition or reuse of logical interfaces using standardized signals and services. Additionally, the LV outlines the detailed design of a logical unit, such as an application or driver within a specific system environment, while considering the given non-functional requirements.

Technical Viewpoint (TV):

As part of the solution space, the Technical Viewpoint (TV) focuses on the constraints and technologies necessary to construct the technical system. The main objective of the TV is to provide an execution environment for the logical nodes and their integrated functions. The TV takes the logical nodes from the LV and connects them with the required hardware (HW) and software (SW) interfaces, thereby linking executable functions, logical nodes, and technical components. The TV bridges the HW and SW elements, integrating the system design within the specified hardware-specific boundaries or technological constraints.

Physical Viewpoint (PV):

The Physical Viewpoint (PV) in the solution space ultimately focuses on realizing the desired end-user systems or subsystems using available physical products and components while ensuring that non-functional performance and quality requirements are met. This viewpoint enables the visualization of block diagrams of the hardware architecture concerning system integration using the developed or existing technical/physical components and elements. The PV also establishes a tightly coupled connection between the electrical/electronic (EE) architecture and the EE detailed design (e.g., hardware schematics), optimizing the E/E development process.

The modeling of requirements (the Requirements Viewpoint) as well as the creation of variability models (the Variability Viewpoint), as defined by ISO 26550 (ISO 1 2015) and ISO 26580 (ISO 2021), extend the model viewpoints with two additional viewpoints, as shown in Figure 3, which are also integral parts of the development process. The further, indirectly architecture-relevant viewpoints are defined as follows:

Requirements Viewpoint (RV):

The Requirements Viewpoint (RV) manages and represents textual requirements within the model by tracking them with other viewpoints using the zigzag method (Weilkiens 2020). It defines customer requirements in the specification document and system requirements in the requirements specification, controls the definition of system element requirements after system decomposition, and supports traceability between requirements and architecture elements. The RV is crucial in the iSEF architecture model and bridges classic systems engineering and ACMBSE. The RV becomes particularly important when models cannot be exchanged between suppliers and OEMs, and a system can only be aligned through a requirements-based design specification.

Variability Viewpoint (VV):

The Variability Viewpoint (VV) manages system variants based on a feature catalog, which describes a system's functional and technical characteristics (features) from the perspective of stakeholders or end users. The VV models follow ISO 26550 (ISO 1 2015), where dependencies are organized hierarchically and in terms of variability. The VV uses variation points on architecture elements to filter the variability of a system according to the given features. A variability model within the VV influences other architecture elements regarding their visibility to user-perceivable functions and technical solutions. The VV is crucial for managing platform and product lines, particularly useful for complex systems with many variants, to increase reusability within engineering organizations (Forlingieri 2022).

The underlying iSEF modeling profile (iSefML) is based on the SysML language and facilitates using all the modeling mentioned in earlier viewpoints, optimized for variable system development processes. The profile includes a customized model browser, uses drawing tools modified for the required element usage, and implements interactive rule sets that ensure the correctness of the composition and interaction between model elements during the modeling process.

6 THE ABSTRACTION LEVELS OF ISEF

The technical and functional scope of a system, as well as the depth to which the system should be examined, are defined by the System Abstraction Levels, which form an artificial boundary but not an inherent property of the system. They serve more as a category or classification for any possible system (with a focus on vehicle-related systems here), as shown in Figure 4. A system abstraction level can refer to the process of abstraction or the generalization of a system's physical, spatial, or temporal details, or the abstraction may instead focus on a user-perceived model or representation of information from the real world. In iSEF, the system abstraction levels are defined according to the framework shown in Figure 4, with five abstraction levels applied to a system's domain. These are:

Environment Abstraction Level (L-EV):

The Level of External View (L-EV) encompasses the context of a System of Interest (SoI), such as the vehicle environment or mobile communication system, which interacts with its immediate operational environment. The L-EV pertains to the interaction with the user and the corresponding environmental effects that the system can respond to or stimulate. Furthermore, this abstraction level includes modeling interactions between various independent systems and functionalities within a System of Systems (e.g., traffic management systems, fleet management systems, and cloud solutions).

User Domain Abstraction Level (L-UD):

The Level of User Domain (L-UD) encompasses a system, including its composite subsystems within a domain (e.g., vehicle, smartphone, airplane, etc.). The L-UD focuses exclusively on a system with which a user interacts or a system that can directly respond to environmental influences. The physical size or complexity of the systems is not a determining factor here. For example, an airplane has a significantly higher number of subsystems on the subsequent abstraction levels than a smartphone. At this level, the systems' physical size or inherent complexity is not the primary focus. Instead, the emphasis is on how the system functions as a whole, with attention to how it meets the user's needs and interacts with its environment.

Domain System Abstraction Level (L-DS):

The Level of Domain Subsystems (L-DS) encompasses subsystems (e.g., Advanced Driver Assistance Systems) of a user system within a domain that focus on a specific set of functions, communication, or technologies that are no longer directly visible to the end user. Integrative subsystems typically consist of at least two or more system products at this abstraction level. Examples include a vehicle door with all its mechanical, electromechanical, and electronic assemblies or a mobile reception system in a smartphone. The L-DS abstraction level is crucial for understanding how these subsystems interact internally and with other parts of the broader system, ensuring that they work seamlessly to support the overall functionality and reliability of the embracing user system.

System Product Abstraction Level (L-SP):

The Level of System Products (L-SP) encompasses all technical systems within a subsystem, such as within the vehicle domain. This subsystem is defined by integrating all discipline-specific components (software, electrical/electronic, mechanical) to create a physical system as a product. This typically includes control units (ECUs), technical actuators, sensors, and the functions to be integrated into these systems. At L-SP, the focus is on how these diverse components come together to form functional units that can be deployed in the field. The functions integrated into these systems at L-SP are designed to meet specific performance, safety, and reliability standards, making this level crucial for ensuring that the overall subsystem performs as expected.

Product Component Abstraction Level (L-PC):

The Level of Physical Components (L-PC) encompasses all system components described within a single discipline to realize a system product. This also includes interdisciplinary interfaces such as the hardware-software interface or communication protocols involving the components. The discipline-specific components then serve as the starting point for further decomposition into specific hardware and software architecture descriptions. At this level, each element's detailed design and specification are considered, and the discipline-specific components and their interfaces set the stage for further decomposition and development of the subsequent hardware and software architecture.

Every system and its subsystems can be categorized into one of the system abstraction levels. When a system is decomposed, a subsystem resulting from a decomposition step may be considered at a lower system abstraction level than the original parent system, provided no further hierarchy is applied. Development always begins at the system abstraction level of the system context, though the L-EV level does not necessarily need to be included. The System of Interest (SoI) determines which system abstraction level should be used for the context and thus serves as the starting point.

The use of system abstraction levels offers several advantages. At each abstraction level, a new system context can be derived, enabling clear interactions between system elements such as functions, nodes, and components. Communication (interaction) between elements is only permitted on the same abstraction level, which prevents "level-hopping" between systems and components on different abstraction levels.

Similarly, system abstraction levels enable decomposition models with precisely formulated rules that specify which element meta-classes may be used at each abstraction level. This allows for the implementation of machine-verifiable rules for all architecture elements in the architectural description through tool-supported mechanisms supported by the iSEF profile. This ensures the system modeling is valid and follows predefined decomposition and integration schemes. This practice facilitates communication among systems engineers from different fields and companies (such as suppliers) because they can focus on the relevant abstraction level. Clear handover rules govern the exchange of elements. As corresponding specifications and requirements must be at each abstraction level, associated specification and requirements models are also necessary.

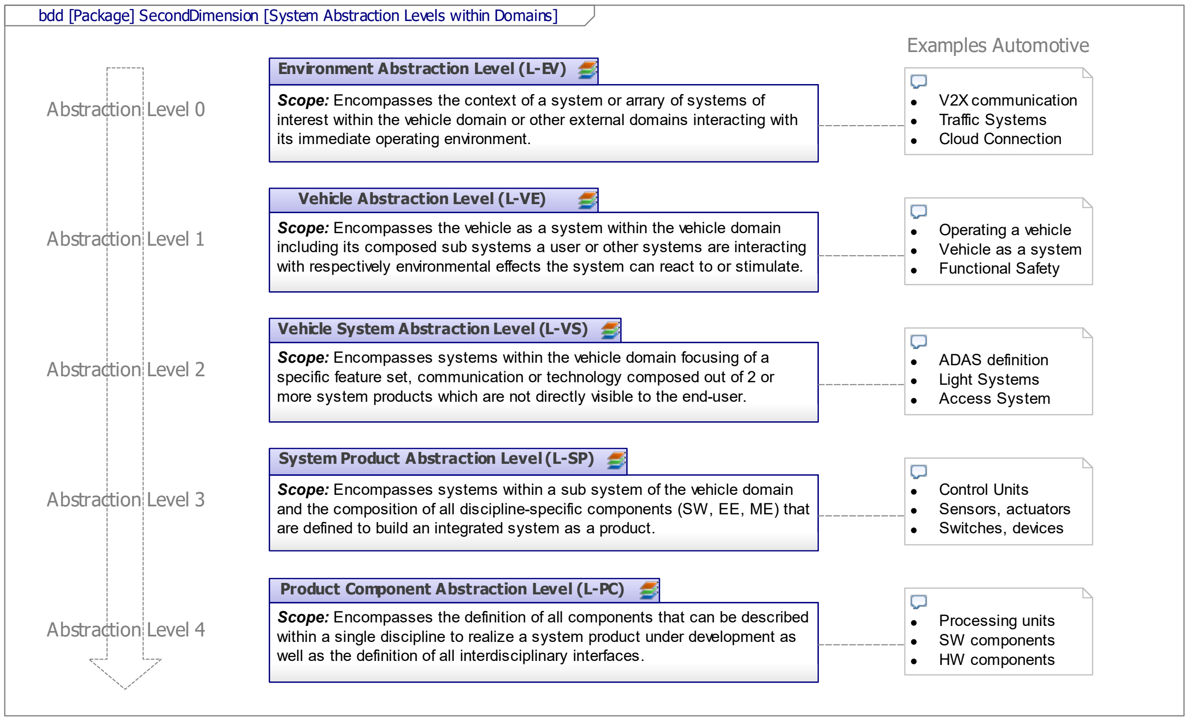

bdd [Package] SecondDimension [System Abstraction Levels within Domains]
Examples Automotive
Abstraction Level 0
Abstraction Level 1
Abstraction Level 2
Abstraction Level 3
Abstraction Level 4
Environment Abstraction Level (L-EV)
Scope: Encompasses the context of a system or array of systems of interest within the vehicle domain or other external domains interacting with its immediate operating environment.
Vehicle Abstraction Level (L-VE)
Scope: Encompasses the vehicle as a system within the vehicle domain including its composed sub systems a user or other systems are interacting with respectively environmental effects the system can react to or stimulate.
Vehicle System Abstraction Level (L-VS)
Scope: Encompasses systems within the vehicle domain focusing of a specific feature set, communication or technology composed out of 2 or more system products which are not directly visible to the end-user.
System Product Abstraction Level (L-SP)
Scope: Encompasses systems within a sub system of the vehicle domain and the composition of all discipline-specific components (SW, EE, ME) that are defined to build an integrated system as a product.
Product Component Abstraction Level (L-PC)
Scope: Encompasses the definition of all components that can be described within a single discipline to realize a system product under development as well as the definition of all interdisciplinary interfaces.
V2X communication
Traffic Systems
Cloud Connection
Operating a vehicle
Vehicle as a system
Functional Safety
ADAS definition
Light Systems
Access System
Control Units
Sensors, actuators
Switches, devices
Processing units
SW components
HW components

Figure 4: Example of the 5 Applied Abstraction Levels for the Vehicle Domain

7 HIERARCHY DEPTH

When a system is analyzed at an abstraction level and modeled across all viewpoints, this is referred to as a complete architecture step. Within this process, the system and function blocks may exhibit different levels of granularity across the various viewpoints.

Therefore, hierarchy depths define the granularity of a system's decomposition within a specific viewpoint's actual system abstraction level, as illustrated in Figure 5.

The lowest and preferred number of hierarchy depths is 1, which helps keep the system decomposition simple. However, it is essential to note that the hierarchy depth does not have to be consistent across the viewpoints within a given system abstraction level. This is because the granularity of the decomposition can vary depending on the decomposition scheme applied to each viewpoint, resulting in different hierarchy depths.

To ensure standardized system decomposition, the iSEF offers a set of reference decomposition models that guide architects in breaking down the system and functions in a way that produces compatible and reusable building blocks.

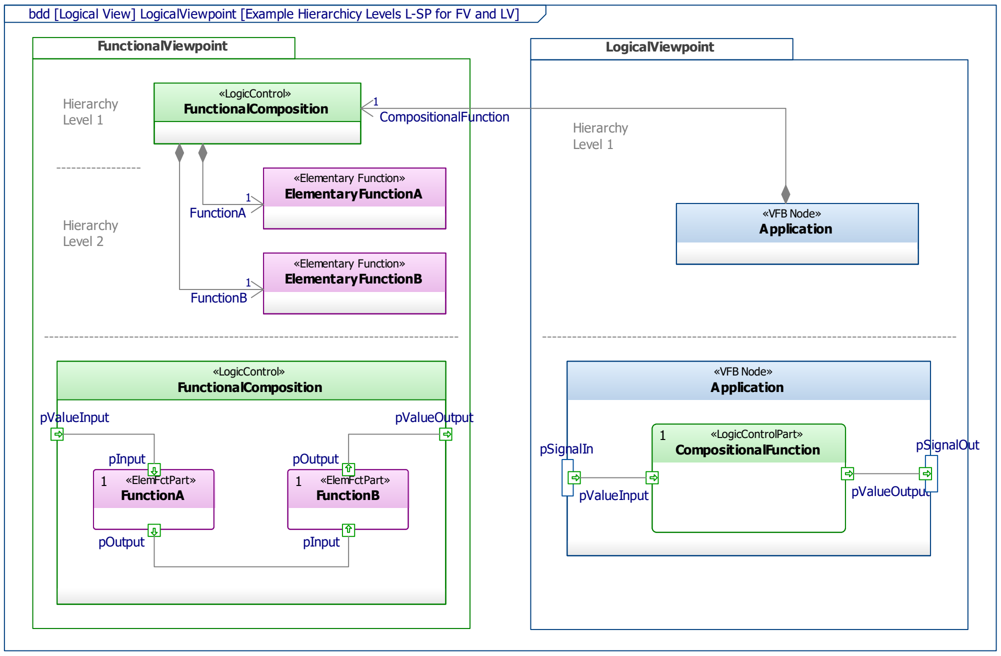

bdd [Logical View] LogicalViewpoint [Example Hierarchicy Levels L-SP for FV and LV]
FunctionalViewpoint
LogicalViewpoint
Hierarchy Level 1
Hierarchy Level 2
«LogicControl» FunctionalComposition
CompositionalFunction
«Elementary Function» ElementaryFunctionA
FunctionA
«Elementary Function» ElementaryFunctionB
FunctionB
«VFB Node» Application
pValueInput
pValueOutput
pInput
pOutput
«ElemFctPart» FunctionA
«ElemFctPart» FunctionB
pSignalIn
pSignalOut
«LogicControlPart» CompositionalFunction

Figure 5: Example of a Hierarchy Depth of "2" in the FV and "1" in the LV at the L-PC

8 LINKING ELEMENTS BETWEEN MODEL VIEWPOINTS

Ensuring a coherent architectural description across different viewpoints requires establishing linking patterns between the architectural elements within the same modeling viewpoint and between elements across different perspectives. The iSEF method addresses this need by introducing extended SysML semantics. Within the framework, elements from various perspectives are interconnected by integrating functional elements directly into logical elements, which are then composed into technical elements. These technical elements are further incorporated into physical components.

The linkage of all involved architecture elements, as illustrated in Figure 6, is strengthened through their interconnected interfaces, which are aligned and checked for consistency by a model checker. Consistency is maintained through nested port modeling, where *Proxy Ports* are embedded within *Full Ports* via an interface definition, and a logical signal encapsulates one or more functional values. This signal is transmitted via a technical interface, carried along a technical line, and connected to a physical interface. This physical interface represents the solution for achieving overall connection across all viewpoints.

The technical and physical constraints that come into play at this modeling step dictate that the physical architecture is based on technological decisions made from the technical viewpoint. Furthermore, the technical modeling depends on architectural choices made from a logical viewpoint. The logical modeling viewpoint, in turn, reflects an implementation based on a specified functional design created in the functional viewpoint. The functional design from the functional viewpoint is also seen as the outcome of an operational analysis that has evaluated the activities and use cases outlined in the operational viewpoint.

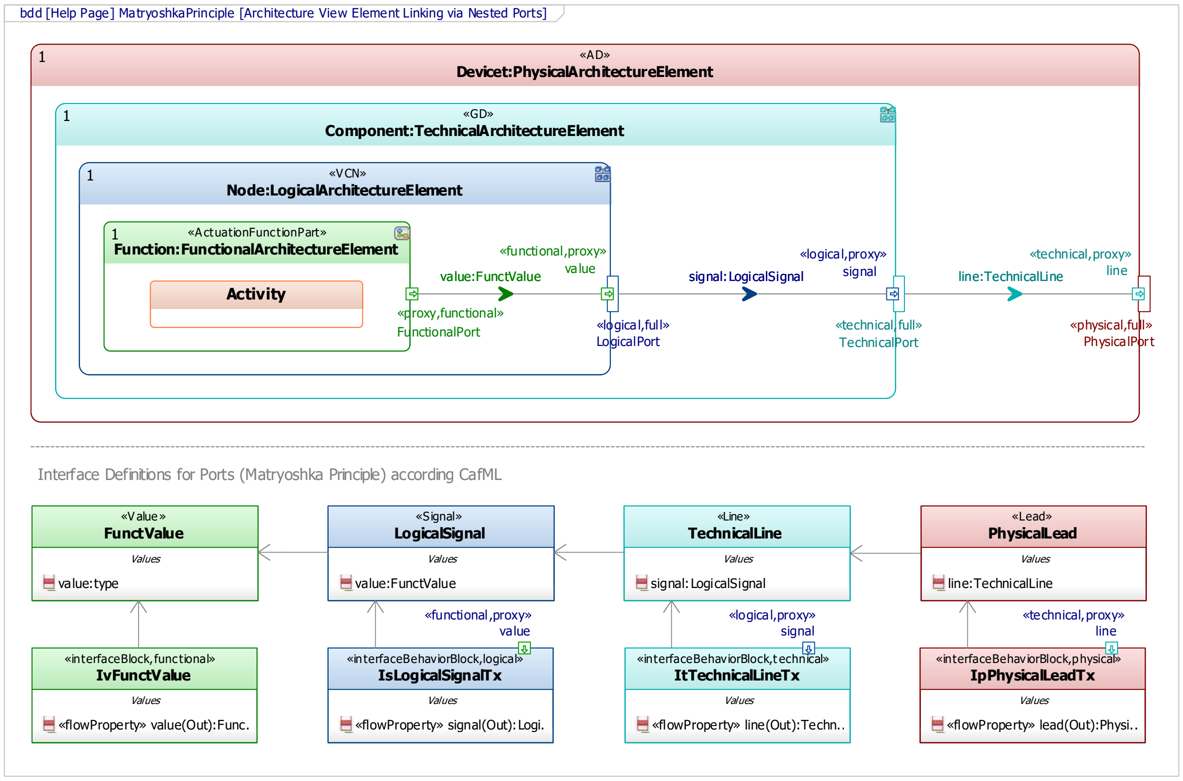

bdd [Help Page] MatryoshkaPrinciple [Architecture View Element Linking via Nested Ports]
1
«AD»
Devicet:PhysicalArchitectureElement
1
«GD»
Component:TechnicalArchitectureElement
1
«VCN»
Node:LogicalArchitectureElement
1
«ActuationFunctionPart»
Function:FunctionalArchitectureElement
Activity
value:FunctValue
«functional,proxy» value
«proxy,functional» FunctionalPort
signal:LogicalSignal
«logical,proxy» signal
«logical,full» LogicalPort
line:TechnicalLine
«technical,proxy» line
«technical,full» TechnicalPort
«physical,full» PhysicalPort
Interface Definitions for Ports (Matryoshka Principle) according CafML
«Value» FunctValue
Values
value:type
«Signal» LogicalSignal
Values
value:FunctValue
«Line» TechnicalLine
Values
signal:LogicalSignal
«Lead» PhysicalLead
Values
line:TechnicalLine
«functional,proxy» value
«logical,proxy» signal
«technical,proxy» line
«interfaceBlock,functional» IvFunctValue
Values
«flowProperty» value(Out):Func.
«interfaceBehaviorBlock,logical» IsLogicalSignalTx
Values
«flowProperty» signal(Out):Logi.
«interfaceBehaviorBlock,technical» ItTechnicalLineTx
Values
«flowProperty» line(Out):Techn..
«interfaceBehaviorBlock,physical» IpPhysicalLeadTx
Values
«flowProperty» lead(Out):Physi..

Figure 6: Linking Elements and Interfaces Across Successive Viewpoints

Figure 6 also illustrates the use of the iSefML stereotype *InterfaceBehaviorBlock*, which is introduced in addition to SysML. This specific interface block facilitates the connection of elements and the formal transfer of information across viewpoints, both statically and dynamically (through simulation). This approach enables the creation of semiformal architectures and designs that comply with the highest safety integrity levels (ISO 26262, 2018). Modeling tools like Rhapsody can apply these principles to automate notations and semantic verifications, achieving machine-based model verifiability. This principle is essential because it allows for a seamless connection of all architectural elements across different abstraction levels and viewpoints. On one side, iSEF enables the coherent linking of composed parts, ensuring that every element within the model is logically connected to its corresponding elements in other viewpoints. This maintains the integrity of the system architecture and facilitates a clear and structured decomposition of the system into its functional, logical, technical, and physical components. On the other side, this view-linking approach allows the architectural elements to fit together like puzzle pieces, where interfaces between elements in one viewpoint align perfectly with those in the next. This alignment ensures that as you move from one architectural view to the next, the elements are not just placed haphazardly but deliberately designed to integrate seamlessly.

Further insights into the connection between viewpoints and system decomposition can be found in a previous publication by Continental (Seidel & Forlingieri, 2023).

9 METHODICAL MODELING STEPS FOR SYSTEM DESIGN

After a brief introduction to the principles for constructing a system architecture, including the previously introduced seven model viewpoints, the author demonstrates system modeling using ACMBSE through a possible workflow, which, among other things, illustrates the development of a system model using iSEF. The example presented in the following section focuses on the design and implementation of the exterior lighting functions of a vehicle based on an actual development scope. For simplicity, only a tiny portion of the overall scope is depicted in this paper.

The author demonstrates eight fundamental modeling steps from analysis to design, which are outlined as follows:

1. **Feature Definition using the Variability Viewpoint (VV):** This involves defining functional and technical features and their relationships in terms of variability, managed through variation points.
2. **System Context and Definition of the System of Interest using the Operational Viewpoint (OV):** This includes the composition and interaction with external elements (actors).
3. **Requirements Derivation using the Requirements Viewpoint (RV):** This step includes traceability of system elements, design constraints, interfaces, and other design aspects.
4. **Behavioral Analysis using the Operational Viewpoint (OV):** This focuses on analyzing the expected system behavior, including a use case analysis and recursive system activity refinement.
5. **Development of the Functional Design using the Functional Viewpoint (FV):** This includes defining state machines, refining activities, and achieving complete functional composition.
6. **Logical Components Development using the Logical Viewpoint (LV):** This involves creating structural system compositions and interface definitions with port linking for a specific target system.
7. **Definition of the Technological Solution of the System using the Technical Viewpoint (TV):** This includes integrating logical architecture elements to bridge the problem space with the solution space.
8. **Realization of the Physical Architecture of the End-User System:** This step uses existing or newly defined implementable physical components and assemblies, including the definition of physical interfaces.

The following models will simplify the development of a fictional Body Control Unit (BCU). The process begins at the vehicle level by identifying the required system capabilities, designing the system functions and interfaces, and developing the functionalities and components at the product level. Finally, these components are integrated into a physical electronic control unit (ECU), which, among other things, implements a series of vehicle lighting functions. This serves as an illustrative example of applying the ACMBSE method.

9.1 STEP 1 - FEATURE DEFINITION OF THE SYSTEM

Before the developer begins defining a system architecture, it is crucial to have a clear understanding of the essential attributes that the system or system product line must possess to meet the end user's needs. This involves analyzing the required *Functional* and *Technical Features*, utilizing the previously introduced Variability Viewpoint (VV) according to the modeling approach described by Forlingieri and Weilkiens (2022).

In line with this understanding, *Functional Features* characterize a System of Interest (SoI) from a functional perspective that is understandable to end users or stakeholders who may not have a deep understanding of system design and behavior. *Functional Features* represent a synthesis of the needs for an SoI or its capabilities, serving as the foundation for deriving functional requirements, as also demonstrated by Beuche et al. (2004) and Forlingieri (2022).

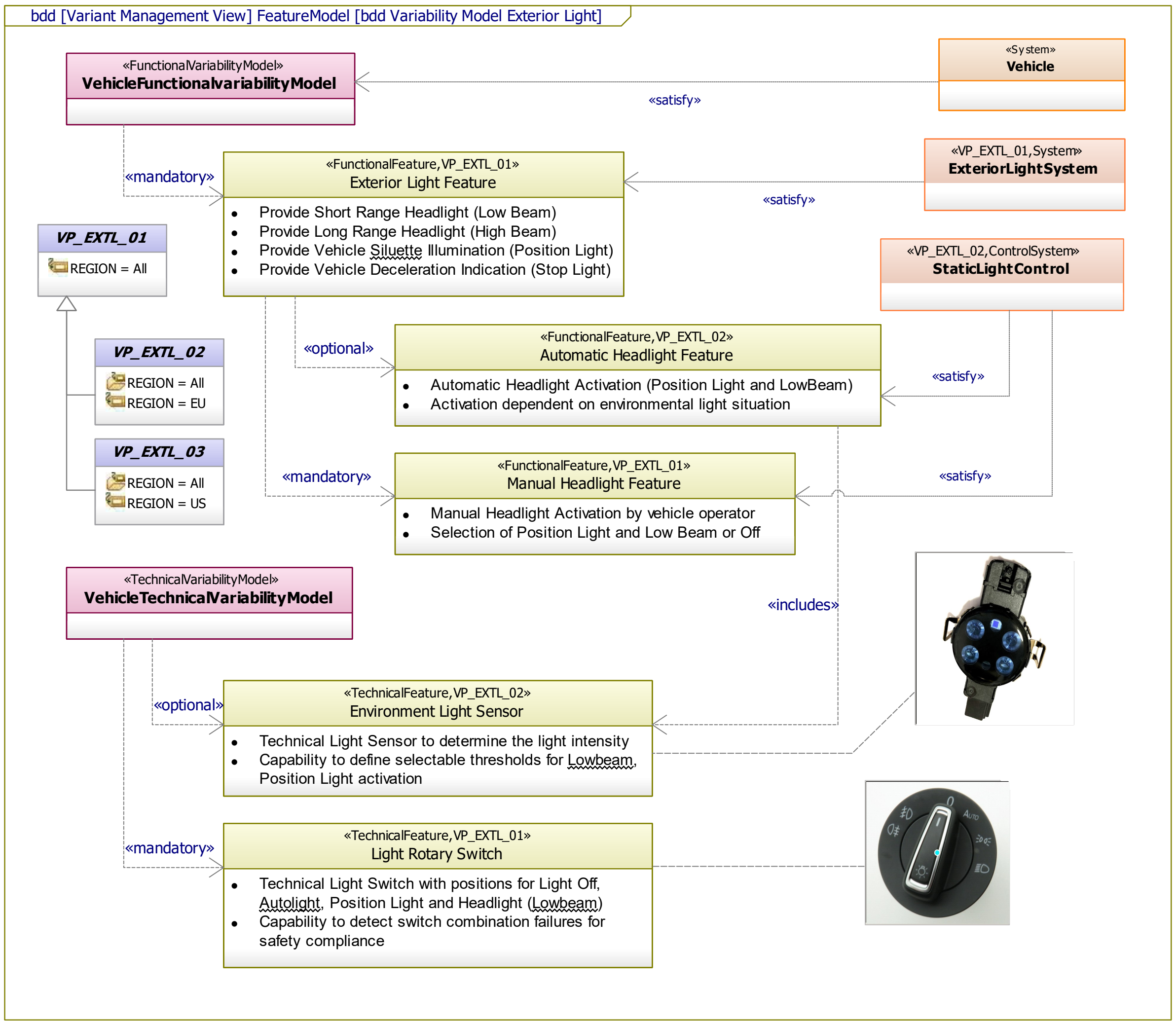

bdd [Variant Management View] FeatureModel [bdd Variability Model Exterior Light]
«FunctionalVariabilityModel» VehicleFunctionalvariabilityModel
«System» Vehicle
«satisfy»
«mandatory»
«FunctionalFeature,VP_EXTL_01» Exterior Light Feature
Provide Short Range Headlight (Low Beam)
Provide Long Range Headlight (High Beam)
Provide Vehicle Siluette Illumination (Position Light)
Provide Vehicle Deceleration Indication (Stop Light)
«VP_EXTL_01,System» ExteriorLightSystem
«satisfy»
VP_EXTL_01
REGION = All
«VP_EXTL_02,ControlSystem» StaticLightControl
VP_EXTL_02
REGION = All
REGION = EU
VP_EXTL_03
REGION = All
REGION = US
«optional»
«FunctionalFeature,VP_EXTL_02» Automatic Headlight Feature
Automatic Headlight Activation (Position Light and LowBeam)
Activation dependent on environmental light situation
«satisfy»
«mandatory»
«FunctionalFeature,VP_EXTL_01» Manual Headlight Feature
Manual Headlight Activation by vehicle operator
Selection of Position Light and Low Beam or Off
«satisfy»
«TechnicalVariabilityModel» VehicleTechnicalVariabilityModel
«includes»
«optional»
«TechnicalFeature,VP_EXTL_02» Environment Light Sensor
Technical Light Sensor to determine the light intensity
Capability to define selectable thresholds for Lowbeam, Position Light activation
«mandatory»
«TechnicalFeature,VP_EXTL_01» Light Rotary Switch
Technical Light Switch with positions for Light Off, Autolight, Position Light and Headlight (Lowbeam)
Capability to detect switch combination failures for safety compliance

Figure 7: Modeling the Variability Model with Functional and Technical Features

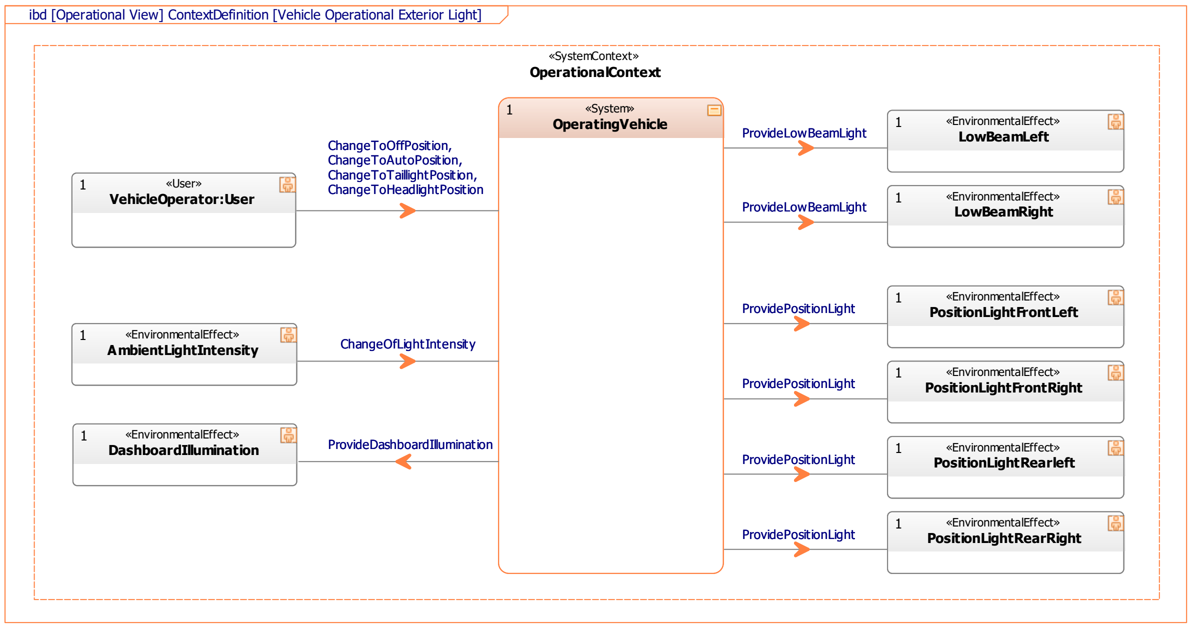
Technical Features characterize a System of Interest (SoI) from a technical perspective regarding performance and technology, making them understandable to end users or stakeholders who may not have a deep understanding of the underlying implementation. They represent a synthesis of technical constraints, performance capabilities, or technological limitations for the desired SoI and serve as the basis for deriving non-functional technical requirements. *Technical Features* often constrain the system's characteristics based on technical boundary conditions and their dependencies on functional features. This dependency (Beuche et al., 2004; Forlingieri 2022) is illustrated in Figure 7, for example, through the "includes" relationship.

Functional and *Technical Features* can also be modeled in terms of their variability, as shown in Figure 7, characterized by dependency relationships such as "mandatory" features, "optional" features, or "alternative" features, as well as relationships to other features like "exclusive" or "inclusive." For example, in Figure 7, the "mandatory" "Manual Headlight Feature" and the optional "Automatic Headlight Feature" are depicted, both realized through a user-perceivable function called "Static Light Control." Additionally, a technical feature that includes an "Environment Light Sensor" to measure light intensity is an example, demonstrating the "inclusive" dependency between the given functional and technical features.

Variation points (e.g., "VPA-EXTL-02"), defined at any variation point in the feature hierarchy, can address the variants. These variation points can subsequently be added to any architecture artifact to clarify the relevance of a variant of an architectural element, as illustrated in the "Static Light Control" function block. Together, the modeling tool and the provided profile enable a plausibility check of the model and a visualization of feature-specific content that can be applied to diagram views in a modeling tool like Rhapsody.

9.2 STEP 2 - SYSTEM & KONTEXT DEFINITION

In the iSEF method, the next step in developing a system architecture is defining the system context, including the System of Interest (SoI). This is accomplished by modeling the relevant system elements in an operational block definition diagram (OBDD), which ensures that all actors within the system context are identified.


ibd [Operational View] ContextDefinition [Vehicle Operational Exterior Light]
«SystemContext» OperationalContext
1 «System» OperatingVehicle
1 «User» VehicleOperator:User
ChangeToOffPosition, ChangeToAutoPosition, ChangeToTaillightPosition, ChangeToHeadlightPosition
1 «EnvironmentalEffect» AmbientLightIntensity
ChangeOfLightIntensity
1 «EnvironmentalEffect» DashboardIllumination
ProvideDashboardIllumination
ProvideLowBeamLight
1 «EnvironmentalEffect» LowBeamLeft
ProvideLowBeamLight
1 «EnvironmentalEffect» LowBeamRight
ProvidePositionLight
1 «EnvironmentalEffect» PositionLightFrontLeft
ProvidePositionLight
1 «EnvironmentalEffect» PositionLightFrontRight
ProvidePositionLight
1 «EnvironmentalEffect» PositionLightRearleft
ProvidePositionLight
1 «EnvironmentalEffect» PositionLightRearRight
Figure 8: Context Definition and Functional Light System Identification

The next step involves specifying interactions between the SoI and the context elements (actors) using one or more operational internal block diagrams (OIBDs), as shown in Figure 8, where the "Operating Vehicle" is the SoI. The gray elements represent the context elements in this definition, as depicted in Figure 8. Identifying the abstract operational events (at the environmental level) is crucial. Figure 8 illustrates a context definition that utilizes the operational events for a lighting system.

The Technical Features shown in Figure 7, which form the basis for constructing the SoI, are also utilized and are particularly important for understanding the SoI. They constrain the subsequent development steps for the system and subsystems and define the interfaces that must later be physically realized. Predefined operational events can also be used for the following operational analysis (see Figure 8) to identify user interactions between the SoI and the context elements, thereby facilitating the consistent definition of use cases. Suppose interfaces have emerged from a previous architecture step and are thus part of the current system abstraction level requirements. In that case, they can be incorporated into the architecture as technical and logical interfaces.

9.3 STEP 3 - RECURSIVE REQUIREMENTS ENGINEERING

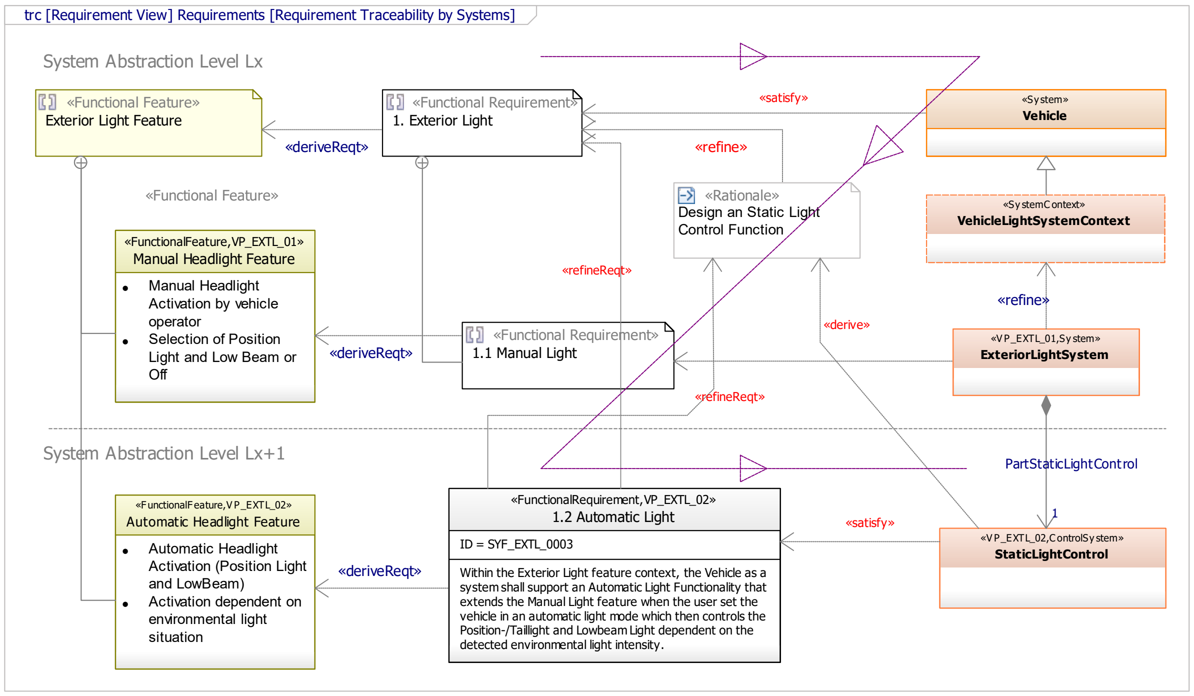

trc [Requirement View] Requirements [Requirement Traceability by Systems]
System Abstraction Level Lx
«Functional Feature» Exterior Light Feature
«Functional Requirement» 1. Exterior Light
«deriveReqt»
«satisfy»
«System» Vehicle
«refine»
«Rationale» Design an Static Light Control Function
«Functional Feature»
«SystemContext» VehicleLightSystemContext
«FunctionalFeature,VP_EXTL_01» Manual Headlight Feature
Manual Headlight Activation by vehicle operator
Selection of Position Light and Low Beam or Off
«refineReqt»
«refine»
«derive»
«Functional Requirement» 1.1 Manual Light
«deriveReqt»
«VP_EXTL_01,System» ExteriorLightSystem
«refineReqt»
System Abstraction Level Lx+1
PartStaticLightControl
«FunctionalFeature,VP_EXTL_02» Automatic Headlight Feature
Automatic Headlight Activation (Position Light and LowBeam)
Activation dependent on environmental light situation
«FunctionalRequirement,VP_EXTL_02» 1.2 Automatic Light
ID = SYF_EXTL_0003
Within the Exterior Light feature context, the Vehicle as a system shall support an Automatic Light Functionality that extends the Manual Light feature when the user set the vehicle in an automatic light mode which then controls the Position-/Taillight and Lowbeam Light dependent on the detected environmental light intensity.
«deriveReqt»
«satisfy»
1
«VP_EXTL_02,ControlSystem» StaticLightControl

Figure 9: Traceability Between Architecture Elements and Requirements

The iSEF method recursively uses the Zig-Zag method (Weilkiens 2008) to create system requirements combined with the principles of ACMBSE. Figure 8 illustrates the traceability between features, requirements, and architecture elements at the relevant abstraction level of the system (Seidel & Forlingieri 2023).

Applying the Zig-Zag method works by following the illustrated "Z" in Figure 9: At the initial system abstraction level (Lx), requirements are created that reflect the scope of the Black-Box system, such as the "Vehicle" system fulfilling the "Exterior Light" requirement in the example. Next, an architectural step is performed to decompose the "Vehicle" system into smaller parts or subsystems (as exemplified by "Static Light Control" in Figure 9). The architectural decisions are documented in rationales, which are also refined from the requirements of the "Vehicle" system. These rationales form the basis for deriving requirements at the next system abstraction level (Lx+1) for the subsystems (e.g., StaticLightControl), as illustrated in Figure 9. This alternation between requirement specification and architectural process is repeated across all system abstraction levels until a subsystem can be implemented within a single discipline (e.g., electrical/electronic or software).

9.4 STEP 4 - OPERATIONAL ANALYSIS OF THE SYSTEM

Conducting an operational analysis by modeling use cases is a crucial step in identifying the activities and state behaviors of the System of Interest (SoI). The Use Case Diagram aids in refining use cases derived from functional requirements and underlying system activities. These activities can be further refined at various abstraction levels within the operational analysis. The iSEF guide provides specialized meta-classes for this purpose, such as System Process, System Use Case, Secondary Use Case, and Continuous Use Case, as Wilkins (2008) described.

Use case analysis is particularly helpful when subsystems are not yet identified, and it is unclear how the system should be decomposed. However, if the system decomposition is already predefined due to sufficiently detailed requirements or technical constraints, use case analysis may not be the most effective approach to elaborate further information. In such cases, it is more practical to directly break down the SoI into already identified subsystems and conduct the operational analysis at the next system abstraction level, where no further structural decomposition is necessary.

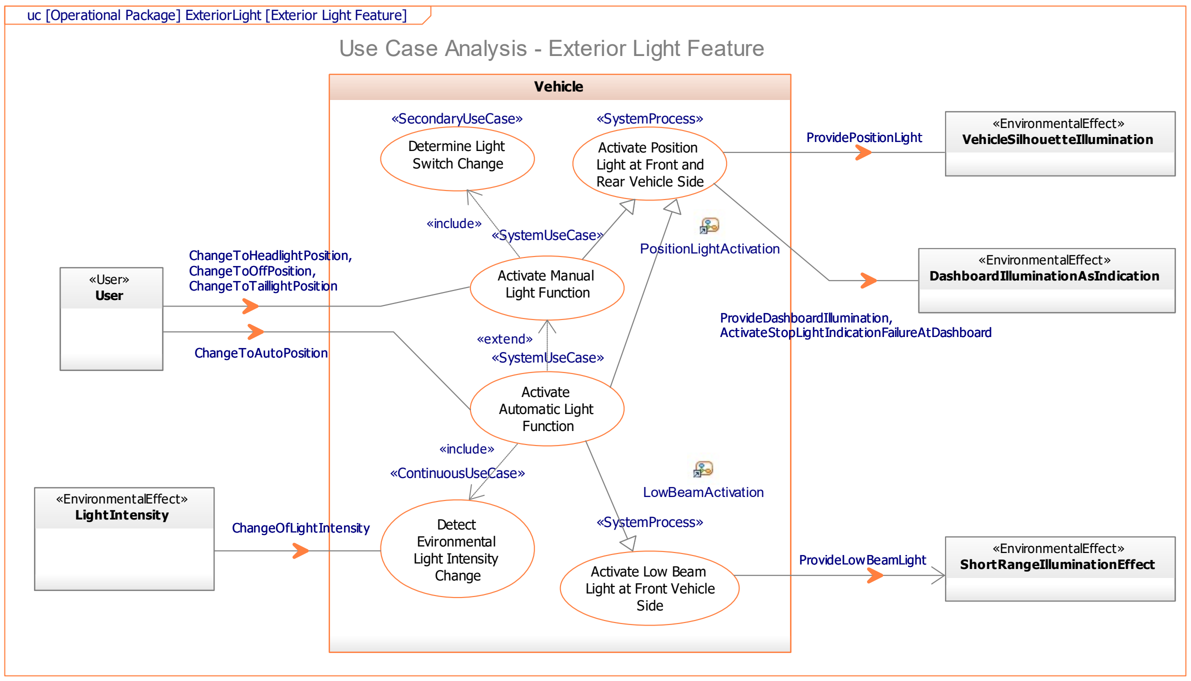

uc [Operational Package] ExteriorLight [Exterior Light Feature]
Use Case Analysis - Exterior Light Feature
Vehicle
«SecondaryUseCase»
Determine Light Switch Change
«SystemProcess»
Activate Position Light at Front and Rear Vehicle Side
ProvidePositionLight
«EnvironmentalEffect» VehicleSilhouetteIllumination
«include»
«SystemUseCase»
PositionLightActivation
«User» User
ChangeToHeadlightPosition, ChangeToOffPosition, ChangeToTaillightPosition
Activate Manual Light Function
«EnvironmentalEffect» DashboardIlluminationAsIndication
ProvideDashboardIllumination, ActivateStopLightIndicationFailureAtDashboard
ChangeToAutoPosition
«extend»
«SystemUseCase»
Activate Automatic Light Function
«include»
«ContinuousUseCase»
LowBeamActivation
«EnvironmentalEffect» LightIntensity
ChangeOfLightIntensity
Detect Evironmental Light Intensity Change
«SystemProcess»
Activate Low Beam Light at Front Vehicle Side
ProvideLowBeamLight
«EnvironmentalEffect» ShortRangeIlluminationEffect

Figure 10: Operational Analysis with Underlying System Activities

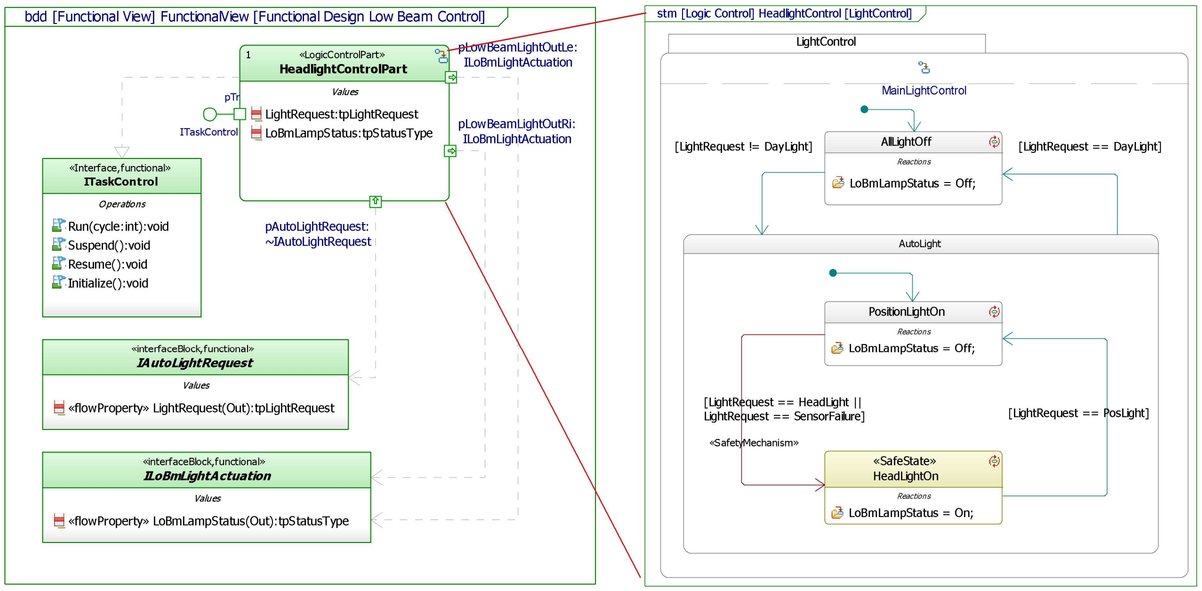
As a distinctive feature of the methodology, the example in Figure 10 shows a use case analysis that reuses the previously defined operational events to consistently model the events of the given context elements with the system use cases. The underlying system activity, such as "Low Beam Activation," then represents a further analysis step for detailed behavior refinement, enabling the creation of state machines.

By modeling with iSEF, it is particularly possible to simulate the intended, though simplified, behavior of activities and state machines during the analysis phase. This allows system details to be discussed with stakeholders early on, helping to avoid unnecessary development cycles.

9.5 STEP 5 - DETAILED FUNCTIONAL DESIGN

Following the Harmony principles used as a guide (Douglas 2017) and additional iSEF extensions, system activities formulated in the previous step of the operational analysis are refined into *Essential Activities* or *Node Activities* and placed into functional block elements (e.g., a *Logic Control* function block) on the same abstraction level or one level below. (Details on use case analysis will be further elaborated in a later whitepaper.) Swimlanes and function blocks modeled in the calling *System Activity* facilitate the linkage between operational viewpoint elements and functional elements.

When refining *System Activities* during the functional design step into executable *Node Activities* or *State Machines*, the operational events from the operational viewpoint are typically transformed into an information flow using SysML proxy ports. Such interface definitions, as exemplified in Figure 11 (e.g., "IAutoLightRequest"), are then implemented through a port of a function block (e.g., "Headlight Control"). Since most embedded systems contain state machines, the example in Figure 11 will illustrate this concept. The state machines are detailed by refining previously defined activities and events to utilize and simulate the targeted data elements, operations, and interfaces. Similarly, the state machine evaluates the "LightRequest" from Figure 8. Once all data elements and behavior are precisely designed, the system can be simulated to verify the intended behavior of the function block.



bdd [Functional View] FunctionalView [Functional Design Low Beam Control]
1 «LogicControlPart» HeadlightControlPart
Values
LightRequest:tpLightRequest
LoBmLampStatus:tpStatusType
pTr
ITaskControl
pLowBeamLightOutLe: ILoBmLightActuation
pLowBeamLightOutRi: ILoBmLightActuation
pAutoLightRequest: ~IAutoLightRequest
«Interface,functional» ITaskControl
Operations
Run(cycle:int):void
Suspend():void
Resume():void
Initialize():void
«interfaceBlock,functional» IAutoLightRequest
Values
«flowProperty» LightRequest(Out):tpLightRequest
«interfaceBlock,functional» ILoBmLightActuation
Values
«flowProperty» LoBmLampStatus(Out):tpStatusType
stm [Logic Control] HeadlightControl [LightControl]
LightControl
MainLightControl
AllLightOff
Reactions
LoBmLampStatus = Off;
[LightRequest != DayLight]
[LightRequest == DayLight]
AutoLight
PositionLightOn
Reactions
LoBmLampStatus = Off;
[LightRequest == HeadLight || LightRequest == SensorFailure]
«SafetyMechanism»
[LightRequest == PosLight]
«SafeState» HeadLightOn
Reactions
LoBmLampStatus = On;

Figure 11: Functional Design of the Low Beam Light Control

From a functional perspective, it is essential to note that many of the functions developed in iSEF are designed for high reusability. These functions can be achieved by using ports for communication between functional elements to ensure the independence of function blocks. In addition to the aspects illustrated in Figure 11 for the "ITask Control" interface, the functional viewpoint emphasizes predefined interfaces for exchanging functional values and implementing function calls to support operations (APIs). These APIs are later integrated into service-oriented interfaces from a logical viewpoint.

9.6 STEP 6 - DESIGN OF LOGICAL COMPONENTS

The primary goal of the logical modeling viewpoint is to organize functional elements into logical units, known as logical nodes, and integrate these nodes into larger logical systems. These nodes do not contain any additional explicitly implemented behavior, as all behavioral models have already been developed within the functional elements. Figure 12 illustrates an example of an integrated "Application Management" component, which controls other function blocks but does not contribute to additional functional user behavior.

Functional elements can typically communicate within the node with other functionalities using ports. While this approach is not mandatory, it allows for flexible reorganization of functional elements that can be developed independently, thereby maximizing the reuse and standardization of the design.

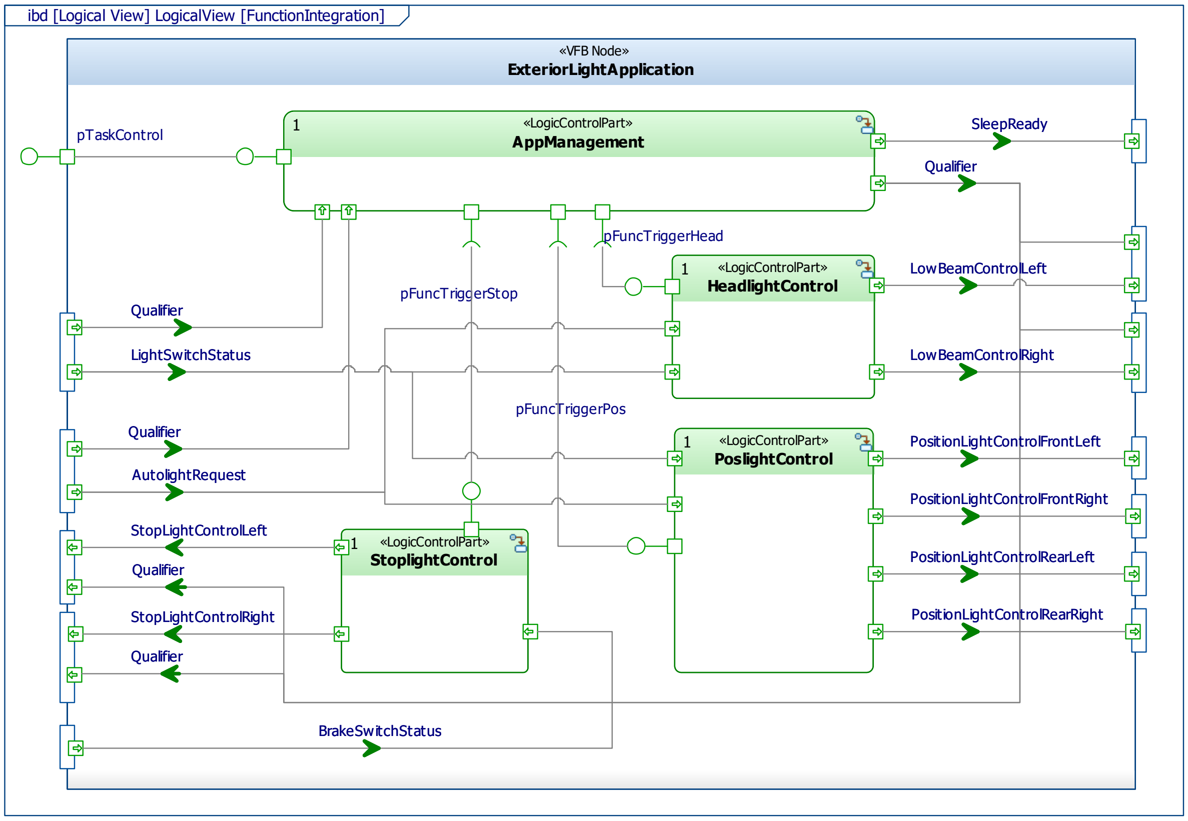


Figure 12: Integration of Function Elements into a Virtual Function Bus Node

Functions within an electronic control unit (ECU) must additionally implement constraints when operating on different operating systems and software frameworks, such as AUTOSAR Classic (2022). However, these functions should be designed to apply to various target systems to maximize reusability. The adaptation is accordingly handled within the logical nodes using logical ports, which apply the principles discussed in the chapter "Linking Elements Between Model Viewpoints."

Full-Ports, typified by *Interface Behavior Blocks*, enable the internal transformation of functional values to and from external, standardized, or custom signal and service interface definitions. This principle also allows for the flexible integration of Simulink blocks from Matlab models (IBM Docu 8.3) or third-party function designs, which might not directly align with logical interfaces, into logical nodes using the presented linking method. While logical nodes and their interfaces appear identical from the outside, their internal functional implementations can vary. Figure 12 illustrates how a logical signal port containing information for “light switch status” and “qualifier” is functionally linked to the “headlight control,” which in turn connects to the logical output port through functional values for “left/right low beam control.”

Allocations are typically avoided when linking elements from the functional to the logical view in ACMBSE. Like object-oriented programming, functional blocks are instantiated within logical nodes, with functional interfaces connected to signal ports. The internal functional ports reveal nested interfaces to the functional elements, as illustrated in Figure 12.

The modeling approach demonstrated leads to a highly consistent and fully executable architecture model, which is crucial for safety-critical applications requiring compliance with the ISO 26262 standard (ISO 1 2018). When iSEF guidelines are correctly applied, element linking across different model views remains confined to elements within the same abstraction level, preventing incorrect and non-implementable integrations due to "level hopping." Predefined decomposition models and specialized stereotypes within the iSefML profile are provided to facilitate real-time model validation during the design of architectural elements.

9.7 STEP 7 - TECHNICAL SYSTEM DEVELOPMENT

The technical view of a system of interest (SoI) is modeled by integrating components from both software and hardware disciplines. The Technical Viewpoint primarily provides the interaction between the system's logical nodes (such as applications) and technical elements like microcontrollers.

It is crucial to examine how the logical nodes communicate with external elements through the processing unit's technical interfaces (ports) (e.g., driver nodes), as shown in Figure 13.

It should also be noted that each technical interface of the microcontroller represents a Hardware-Software Interface (HSI), which includes analog or digital inputs and even data interfaces for communication over the Controller Area Network (CAN) bus. Instead of directly integrating with existing controllers, this model view focuses on identifying the technically significant interfaces for constructing the controller. The actual implementation of the existing controller will be carried out in later phases on the physical model view.

For the technical model view, it should be noted that the logical nodes (software components), simplified here, are integrated through composition, as the focus is on the elements' interaction. In reality, the code generation and compilation process must be carried out. However, these model views are then represented through deployment diagrams using the UML profile, where the focus shifts to the software-specific technical and physical artifacts, which are irrelevant in the system representation here.

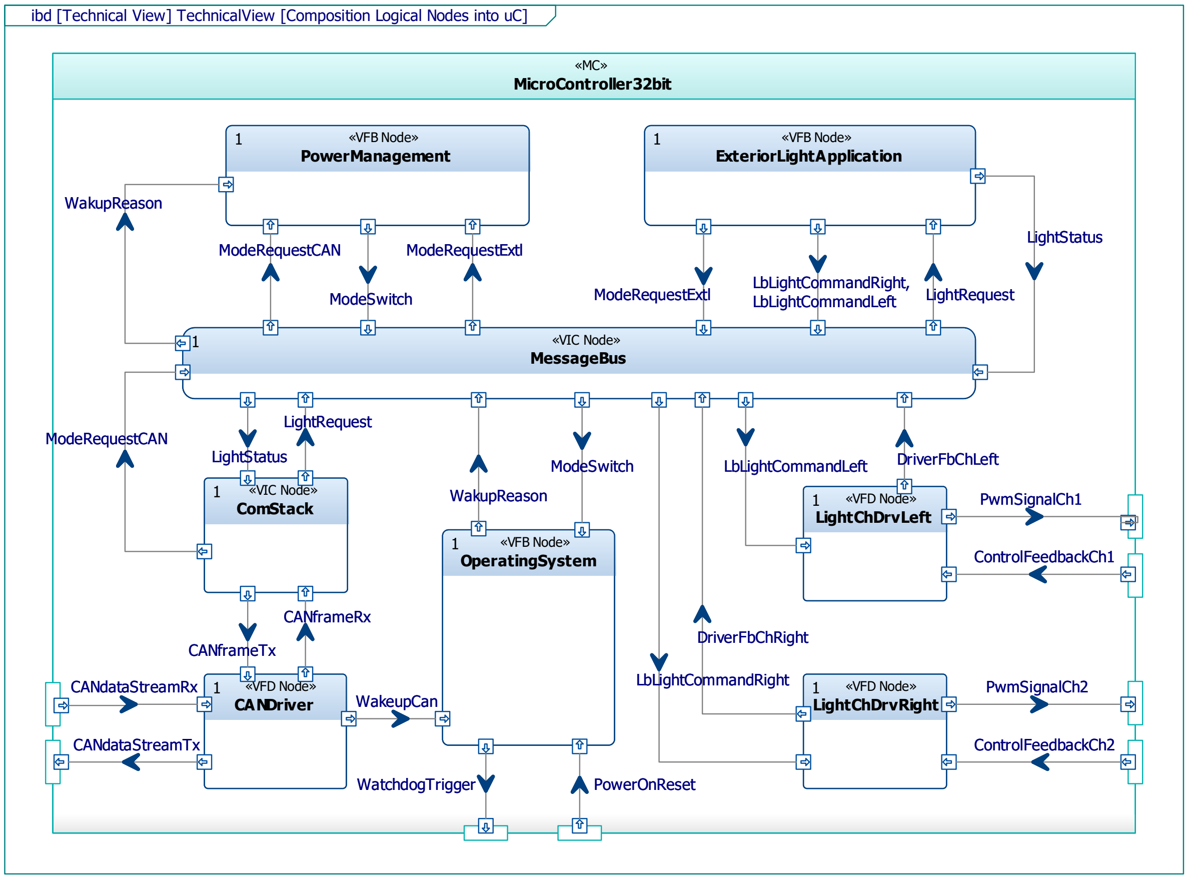


Figure 13: Implementation of Logical Nodes in a Technical Microcontroller

Modeling the technical architecture according to the iSEF method, particularly with the HSI elements, enables an executable simulation of logical elements within a processing unit before the software is designed. This allows for the early verification of technical constraints such as scheduling, resources, and timing in the development phase. Figure 13 illustrates how the “Exterior Light Application” from Figure 12 is integrated into the microcontroller architecture and interacts with other logical nodes such as “Energy Management” or “Communication Stack.” Additionally, a message bus is integrated to route all logical signals.

This approach facilitates the simulation of hardware-dependent safety mechanisms between hardware and software to verify safety-critical system developments up to ASIL D (ISO 26262, 2018).

The iSEF methods enable the creation of system prototypes even before the actual hardware is available, which can significantly reduce development costs. However, such detailed modeling challenges the architect's knowledge and requires a disciplined approach to develop all functional nodes and interface elements fully. Therefore, a compromise in the modeling scope is often necessary, focusing on the parts of the system that are essential for understanding and simulation while other parts remain static and not fully executable.

However, it needs to be considered that it is crucial that a model-based modeling approach also requires appropriate training for the system engineer to learn how to achieve their goals. Therefore, the invenio Systems Engineering GmbH conducts the necessary training to work effectively with the framework and libraries.

9.8 STEP 8 - PHYSICAL REALIZATION OF THE ARCHITECTURE

Physical modeling assists system and hardware engineers in the architecture of end products, primarily electronic control units (ECUs). However, physical modeling could also be applied to design semiconductor chipsets, document their architecture, or model entire vehicle E/E systems. The iSEF supports this goal in various ways. It provides model elements to develop microcontrollers based on predefined templates and port configurations. It also sets up the device for the software and the fundamental system. Furthermore, it helps precisely define the end-user system, such as the vehicle. For instance, an entire wiring harness is integrated with models of all control units.

This enables highly complex systems to be integrated and verified in a model by an OEM using the provided model components from suppliers without the need to build a single prototype. This can lead to significant development cost savings, as fewer correction loops are typically needed.

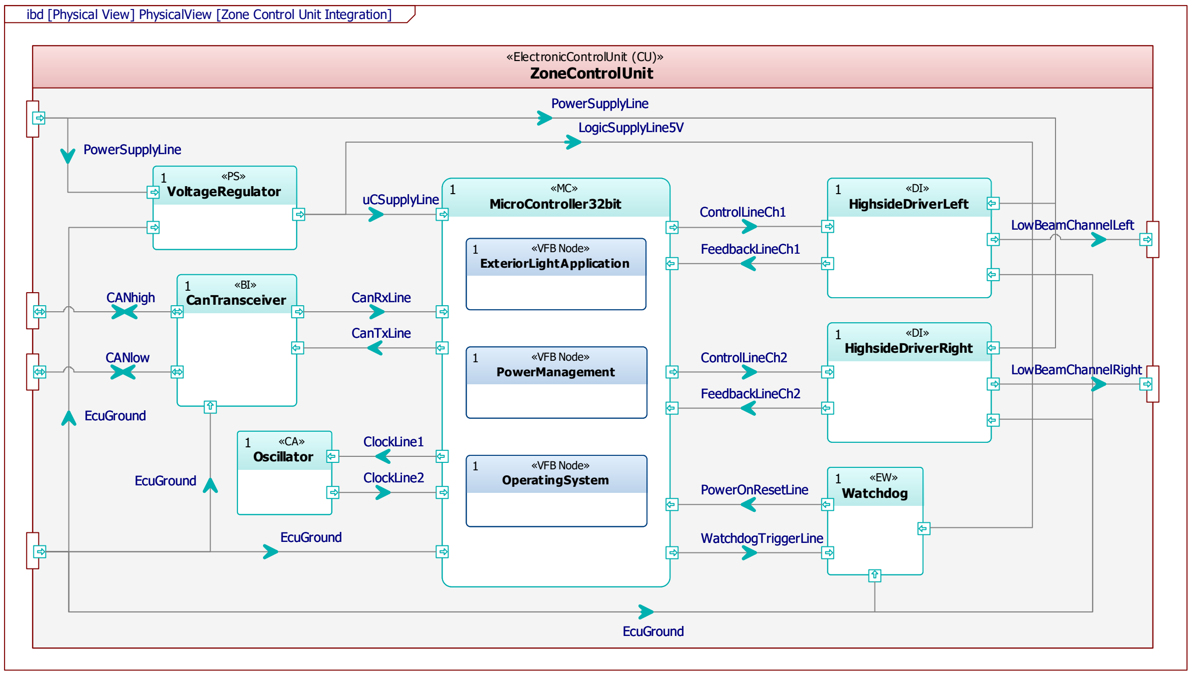


Figure 14: Composition of Technical Elements in the Physical Control Unit

Figure 14 illustrates an approach, among other possibilities, for building the hardware architecture of an ECU based on technical solutions defined by technical elements, as typically determined by a systems engineer. In this example, the previously designed technical elements (e.g., the technical microcontroller from Figure 13) are integrated into a physically to-be-developed ECU and connected to the ECU's required ports. With this architecture, the systems engineer can pass all technical solutions and necessary physical constraints to a hardware engineer, who designs the physical system (ECU) while considering these constraints.

The iSEF aims to provide many predefined physical standard components in the future to support these modeling approaches and accelerate the development process.

Another approach not covered in this article involves modeling a complete physical architecture for the final product. For such a use case, the ECU integrates all relevant physical elements into a fully optimized hardware architecture in this approach. This optimization concerns performance and cost based on technical features and given technical constraints. The result is a specific hardware design. This type of architecture, also known as a physical block diagram, enables the development of reusable, standardized physical subcomponents. This can significantly reduce development time when solutions are known and can be architecturally assembled from modular elements. This approach is particularly advantageous for large platforms in automotive manufacturing, as it can significantly streamline development. However, a significant initial investment is required to create these modular elements.

10 CONCLUSION AND OUTLOOK

In the coming years, significant changes in vehicle electronics will intensify the need for scalable, flexible platforms for system development, including the transition to the next generation of E/E architecture (Seidel & Forlingieri 2023). To meet this new and challenging context, a system architecture development approach that goes beyond current rules and traditional document-based methods. The ACMBSE approach presented in this article, implemented in the invenio Systems Engineering Framework, emphasizes the central role of system architecture in model-based development. The author has highlighted the importance of building and implementing this architectural approach in this article by introducing several key guiding principles:

(1) Developing a modeling framework like iSEF is essential for coherent and efficient development across various engineering departments. This practice can also be applied in other industries, as seen with Airbus adopting MOFLT (Ducamp et al., 2022). Applying ACMBSE principles can overcome challenges such as improved architectural understanding, component definition, and sustainable communication within the team and with stakeholders.

(2) The white paper emphasizes the importance of establishing various yet interconnected model viewpoints during system development. The iSEF framework provides seven fundamental model viewpoints, similar to analogous frameworks (Roques 2016; Ducamp et al., 2022), allowing system design from different perspectives and seamless integration from system to software architecture. In the automotive sector, this method extends from vehicle integration to ECU development, covering all essential architectural aspects and serving as a bridge for the overall product description.

(3) The author systematically demonstrated one of the many potential applications of iSEF and the introduced viewpoints. A fundamental principle of MBSE for establishing modeling can be achieved in iSEF by applying the principles of linking views presented in this article. While not all viewpoints and system abstraction levels are always required simultaneously, different aspects of the same viewpoint can be utilized during the development of complex systems, such as vehicles.

Although this white paper primarily focused on the analysis and design phase of system development using ACMBSE, it illustrated the modeling of a system architecture in 8 simple steps. However, one critical aspect remains model-based verification and validation. Ensuring the precision and coherence of a complex system architecture requires the verification of the model's semantic and syntactic accuracy and the consistency between the interacting model components. Therefore, future tools, including the iSEF, should incorporate technical procedures such as Just-in-Time model checks, which implement the framework's underlying methods and validation mechanisms and make them easily applicable to the user.

Another vital point briefly mentioned by the author is modeling the variability of functional and technical features, which considers the desired characteristics in product line development from the beginning. However, the article's goal was not to show how these features can be systematically modeled and used to develop multiple product variants. Following the example of Forlingieri and Weilkiens (2022), a future article could demonstrate how variant modeling can be realized across various viewpoints and abstraction levels using iSEF.

In conclusion, this article highlights the architecture-focused model-based aspect of system development, represented exclusively within the modeling tool. However, it is essential to emphasize the importance of using the system model as a bridge to connect with other domain-specific disciplines within the engineering lifecycle. MBSE does not replace requirements management, test management, software development, or the mechanical discipline. Instead, iSEF enables the seamless integration of these disciplines through an overarching model, facilitating a seamless and coherent system development process. The invenio Systems Engineering Framework (iSEF) has taken the first steps toward achieving this goal through architecture-centric model-based systems engineering (ACMBSE).

11 REFERENCES

Autosar Classic 2022, AUTOSAR Classic Platform, release November 2022 <https://www.autosar.org/standards/classic-platform>

Beuche, D, Papajewski, P, Schröder-Preikschat, W. Variability management with feature models, Science of Computer Programming, Volume 53, Issue 3, 2004, Pages 333-352.

Brooks M. D., Wheeler T.M. 2007. Experiences in Applying Architecture-Centric Model-Based System Engineering to Large-Scale, Distributed, Real-Time Systems <https://www.mitre.org/sites/default/files/pdf/07_0838.pdf > Pub. 2007 Computer Science

Burkacky, O, Kellner, M, Deichmann, J, Keuntje, P and Werra, J. 2021, Rewiring car electronics and software architecture for the 'Roaring 2020s'. McKinsey & Co

Douglas, B, P 2017. Harmony a MBSE Deskbook Version 1.00 Agile Model-Based Systems Engineering Best Practices with IBM Rhapsody. IBM Corporation

Ducamp, C, Bouffaron, F, Ernadote, D, Wirtz, J and Darbin, A. 2022. MBSE approach for complex industrial organization program. INCOSE International Symposium, pp. 839-856.

Forlingieri, M 2022. 'The four dimensions of Variability and their impact on MBPLE: How to approach variability in the development of aircraft product lines at Airbus'. Proceedings of the 16th International Working Conference on Variability Modeling of Software-Intensive Systems (VAMOS '22), February 23–25, 2022, Florence, Italy. ACM, Vamos.

Forlingieri, M. Weilkiens, T. 2022. 'Two Variant Modeling Methods for MBPLE at Airbus'. INCOSE International Symposium, vol. 32, no. 1, pp. 1097-1113.

Friedenthal, S. Moore, A, Steiner, R. 2015. A Practical Guide to SysML: The Systems Modeling Language. The MK/OMG Press, 3rd Edition

IBM 2022, IBM Engineering Systems Design Rhapsody, viewed 13 December 2022 <https://www.ibm.com/products/systems-design-rhapsody>

IBM Docu 8.3, Integrating Rational Rhapsody and the MathWorks Simulink
< https://www.ibm.com/docs/en/elms/esdr/8.3?topic=tools-integrating-rational-rhapsody-mathworks-simulink>

ISO 26262-3:2018. Road vehicles — Functional safety — Part 3:
Concept phase, release 2018

ISO/IEC 26550:2015. Software and systems engineering — Reference model for product line engineering and management.

ISO/IEC 26580:2021. Software and systems engineering — Methods and tools for the feature-based approach to software and systems product line engineering.

ISO/IEC/IEEE 15288:2015, Systems and software engineering — System life cycle processes

ISO/IEC/IEEE 42010:2022. Software, systems and enterprise — Architecture description

Nayak, J, Mishra M, Naik, B, Swapnarekha, H, Cengiz, K, Shanmuganathan, V. 2022. An impact study of COVID-19 on six different industries: Automobile, energy and power, agriculture, education, travel and tourism and consumer electronics. Expert Systems, vol. 39, no.3

OMG 2018, OMG Systems Modeling Language (OMG SysML), Version 1.6.

Seidel, E. Forlingieri, M. 2023, Moving towards Server-Zone Architecture with MBSE at Continental, INCOSE International Symposium.

TOGAF 10, The TOGAF Standard, 10th Edition, Version C220
<https://publications.opengroup.org/standards/togaf/specifications/c220>

UDPM 2017, UPDM Unified Profile for DoDAF and MODAF, Version 2.1.1
<https://www.omg.org/spec/UPDM/2.1.1/PDF>

Weilkiens, T. 2008, Systems Engineering with SysML/UML: Modeling, Analysis, Design, The MK/OMG Press

Weilkiens, T. (2020). SYSMOD - The Systems Modeling Toolbox: Pragmatic MBSE with SysML. MBSE 4U, 3rd Edition

Zachman 2023, The Zachman Framework, Version 16.1
<https://sparxsystems.com/resources/user-guides/16.1/model-domains/frameworks/zachman.pdf>

12 BIOGRAPHY

Enrico Seidel is a Senior Consultant at invenio Systems Engineering GmbH and is currently responsible as an expert for model-based development. As a technical architect, he leads the development of the iSEF. Previously, he was a Senior Technical Expert for Continental Automotive Singapore in Systems Engineering (SE). Within the Business Area (BA) Architecture and Networking, he was responsible for defining the SE working principles in the Engineering Excellence group. With over 17 years of experience in Systems Engineering, he co-developed the Capability Architecture Framework (CAF) with his team and is now advancing the ideas of MBSE with the successor product, iSEF, at invenio AG.