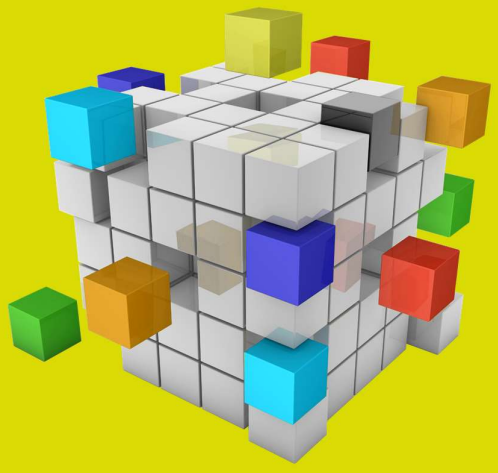
invenio



Whitepaper

ARCHITEKTURZENTRIERTES, MODELLBASIERTES SYSTEMS ENGINEERING (ACMBSE)

VERSION	DATUM	AUTOR	ÄNDERUNG(EN)
1.0	02.07.2024	Enrico Seidel	Erstausgabe
1.1	13.07.2024	Enrico Seidel	Kleinere Korrekturen
1.2	21.08.2024	Enrico Seidel	Kleinere Korrekturen

INHALT

ZUSAMMENFASSUNG

In den vergangenen zehn Jahren hat die Automobilindustrie eine bedeutende Transformation durchlaufen, indem sie vom traditionellen dokumentenbasierten Systems Engineering zu einem stärker architekturzentrierten und modellbasierten Systems Engineering (MBSE) übergegangen ist. Diese Veränderung wird durch die rapide zunehmende Komplexität und die Schwierigkeiten bei der Verwaltung von Systembeschreibungen mit konventionellen dokumentenbasierten Methoden angetrieben. Diese Veröffentlichung untersucht, wie MBSE diese Transformation mithilfe des durch den Autor entwickelten Architektur-Frameworks unterstützen kann. Der Autor schlägt einen architekturzentrierten Ansatz für Systems Engineering, basierend auf verschiedenen Sichten (Viewpoints) vor. Der Fokus auf Architekturzentrierung und die Nutzung von Modellen ermöglicht eine effizientere und effektivere Analyse, Entwicklung und Gestaltung komplexer Systeme. Der Autor verwendet zusätzlich ein Beispiel aus dem Automotive-Bereich, um zu demonstrieren, wie dieser Ansatz praktisch in einen Entwicklungsworkflow integriert werden kann.

1 EINLEITUNG

Die Automobilindustrie, wie auch andere Industriesektoren, stehen in der heutigen sich schnell verändernden Umgebung vor Herausforderungen in Bezug auf Produktivität, Rentabilität und Humankapital (Nayak et al., 2022). Der Übergang zu einem softwarebasierten Ökosystem hat dazu geführt, dass Automobilhersteller erhebliche technologische Fortschritte in mehreren Funktionsbereichen erzielen müssen (Burkacky et al., 2021). Die wachsende Nachfrage nach verbesserten Systemfähigkeiten und Effizienz in der Entwicklung treibt die Nutzung von Systemmodellen voran. Traditionelles dokumentenbasiertes Systems Engineering reicht nicht mehr aus, um die Komplexität softwaregetriebener Funktionen in widerstandsfähigen und sicherheitskritischen Systemen zu bewältigen. Stattdessen ist ein MBSE-Ansatz erforderlich, um die Einhaltung kritischer Standards wie ISO 26262 (ISO 1 2018) zu erleichtern.

Der Autor in diesem Artikel geht einen Schritt weiter, indem er die Architektur in den Vordergrund stellt. Traditionelles dokumentenbasiertes Systems Engineering wurde für moderne komplexe Systeme wie Automobilsysteme und -Komponenten unzureichend. Daher gewinnt ein architekturzentrierter modellbasierter Ansatz an Bedeutung, der in diesem Papier als Architekturzentriertes Modellbasiertes Systems Engineering (ACMBSE) bezeichnet wird (Brooks et al., 2007). ACMBSE bietet eine Möglichkeit, das Design und die Implementierung des Systemverhaltens durch rigorose Modelle und Simulationen zu gewährleisten, was es zu einer wesentlichen Methodik für Organisationen wie Automobilzulieferer macht, um sich an die sich ändernden Anforderungen der Automobilindustrie anzupassen.

Diese Veröffentlichung untersucht die Einführung von ACMBSE und zeigt auf, wie dieser Ansatz für Systemanalyse, Funktionsentwicklung, das Design logischer Elemente und technisches Systemdesign bis hin zur Implementierung elektronischer Steuergeräte einschließlich der physischen Umsetzung genutzt werden kann.

2 ARCHITEKTURZENTRIERTES MBSE

ACMBSE ist ein Systems-Engineering-Ansatz, der die zentrale Rolle der Systemarchitektur bei der Entwicklung komplexer Systeme stark betont, wie in Abbildung 1 dargestellt. ACMBSE basiert auf dem Prinzip, dass die Architektur eines Systems und aller Subsysteme einen kohärenten Überblick über das System bietet, einschließlich seiner Darstellungen der Komponenten, ihrer Wechselbeziehungen und des gesamten Systemverhaltens, das sowohl Software- als auch Hardware-Aspekte umfasst. Daher fungiert ACMBSE als Brücke zu heterogenen, domänenspezifischen Ingenieursdisziplinen.

Dieser Ansatz erfordert weiterhin, dass alle anderen Aspekte des Systems Engineering, wie Anforderungen, funktionale Sicherheit, Sicherheit und Datenschutz, Systemvariabilität, Zuverlässigkeit, Verifikation und Validierung, mit der Architektur in Einklang gebracht werden und zu ihren Perspektiven (Viewpoints) beitragen. Im Vergleich zur dokumentenbasierten Entwicklung bietet ACMBSE mehrere Vorteile, wie verbesserte Kommunikation, Wartbarkeit von Informationen, erweiterte Fähigkeit zur Verwaltung von Systemkomplexität, erhöhte Produktqualität und gesteigerte Wissensaufnahme.

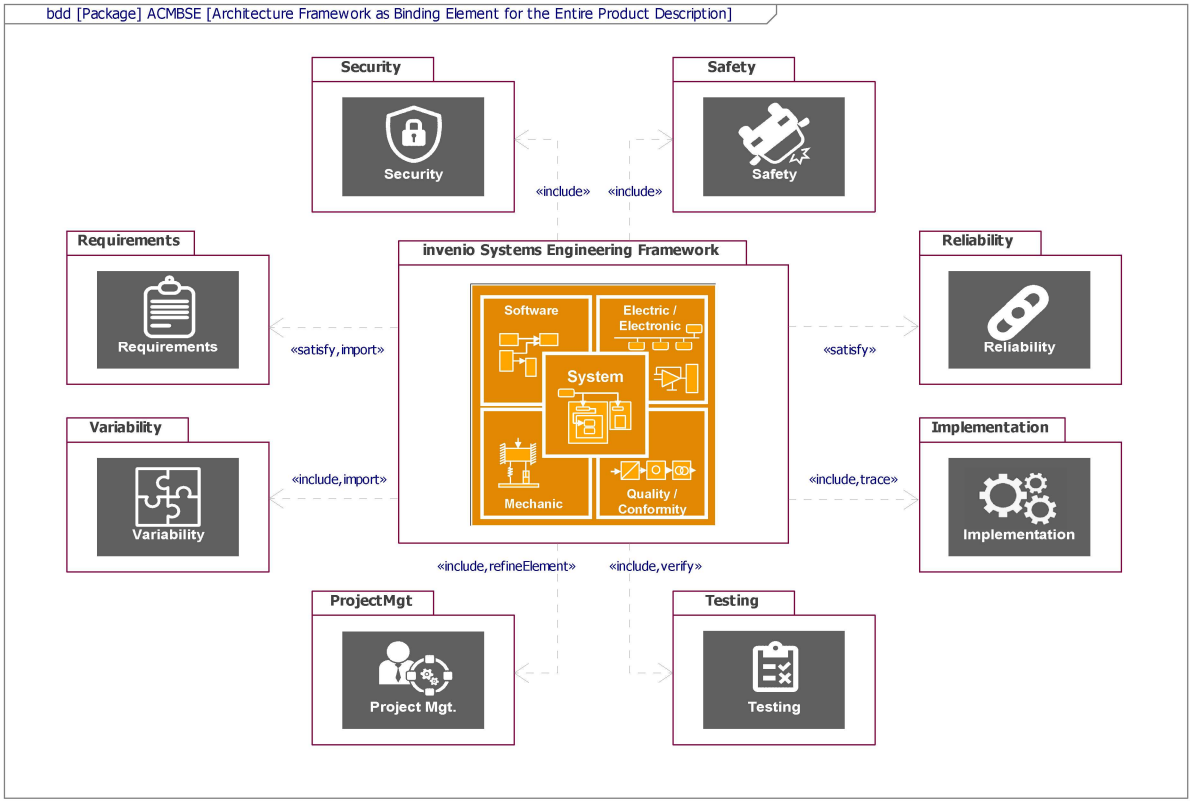


Abbildung 1: Das Architektur-Framework als Bindeglied für die Gesamtproduktbeschreibung

3 DIE BEDEUTUNG EINES ARCHITEKTUR-FRAMEWORKS

SysML (OMG 2018) ist eine weitverbreitete Sprache im Systems Engineering, die eine modellbasierte Entwicklung ermöglicht. Sie definiert jedoch keine Regeln und Methoden für den Aufbau einer Systemarchitektur und -gestaltung in einer bestimmten Domäne (ISO 2022). Um eine domänenübergreifende Plattform zu erreichen, ist es entscheidend, einen standardisierten Satz von Methoden und Werkzeugen zur Entwicklung von Systemprodukten und ihren Architekturartefakten zu haben.

Die Nutzung strukturierter Frameworks wie Harmony (Douglas 2017), OOSEM (Friedenthal et al., 2015) oder MOFLT (Ducamp et al., 2022) ist entscheidend für die konsistente Implementierung von MBSE in großem Maßstab. Jedes Framework hat seine Stärken innerhalb domänenfokussierter Architekturframeworks, einschließlich UPDM (UDPM 2017), welches das DoDAF (Department of Defence Architecture (Framework) und MoDAF (Ministry of Defence Architecture Framework) kombiniert. Der Autor hat jedoch erkannt, dass diese, hauptsächlich für die Verteidigung entwickelten Frameworks, den spezifischen Anforderungen der Automobilindustrie nicht gerecht wurden.

Herausforderungen wie Wiederverwendbarkeit, automatisierte Modellverifikation, systematische Nutzung von Sichten auf das Modell, Systemabstraktionsebenen und Hierarchietiefe als Voraussetzung für den ACMBSE-Ansatz wurden nicht vollständig erfüllt. Selbst TOGAF (TOGAF 10) und Zachman (Zachman 2023) Frameworks, die darauf abzielen, Unternehmensperspektiven zu erfassen, würden Anpassungen erfordern, um den technischen Aspekten der Automobilindustrie gerecht zu werden. In diesem Kontext erwies sich nur das SYSMOD (Weilkiens 2020) Framework als geeignet und entsprach den Bedürfnissen des Autors nach maßgeschneiderten Viewpoints und Modelliertechniken. Der Autor wollte sein Framework hauptsächlich für den Automobilsektor einsetzen, und erkannte die Notwendigkeit, die zuvor genannten Kriterien für das ganzheitliche Systemdesign einschließlich der Entwicklung elektronischer Steuergeräte und Komponenten einbeziehen zu müssen.

Aufgrund der genannten Gründe entwickelte der Autor in der Vergangenheit bei der Continental AG ein Architektur-Framework (CAF), welches mit dem Wechsel in die invenio AG nun signifikant weiterentwickelt und als das invenio Systems Engineering Framework (iSEF) publiziert werden soll. Seit kurzer Zeit entwickelte sich dieses Framework nun zu einer ausgereiften Modellierungsmethodik inklusive Sprache und Tool-Set, das alle notwendigen Methoden und Prozessschritte definiert, um nahtlos eine Architektur- und Designspezifikation sowohl für Systeme und Software als auch für Hardware-Ingenieure zu entwickeln. Neben den üblichen Regeln und Methoden, die die SysML-Sprache erweitern, soll das iSEF ein umfassendes Set vordefinierter Architektur-Artefakte, vorentwickelter Designelemente und wiederverwendbarer Funktionen bieten, womit die Entwicklung standardisierter Automobilfunktionen und -Plattformen in kürzeren Markteinführungszeiten unterstützt werden kann. Überdies wird mit iSEF die Voraussetzung geschaffen, alle wichtigen ISO-Standards wie ISO 42010 (ISO 2022) und ISO 15288 (ISO 2 2015) vollständig einzuhalten. Zur Implementierung von iSEF wurde IBM Rhapsody (IBM 2022) als Modellierungswerkzeug gewählt, das in die IBM Engineering Lifecycle Management (ELM) Plattform eingebettet ist und ein vollständig angepasstes domänenspezifisches Profil basierend auf SysML verwendet.

4 DIE 3 DIMENSIONEN EINER SYSTEMARCHITEKTUR

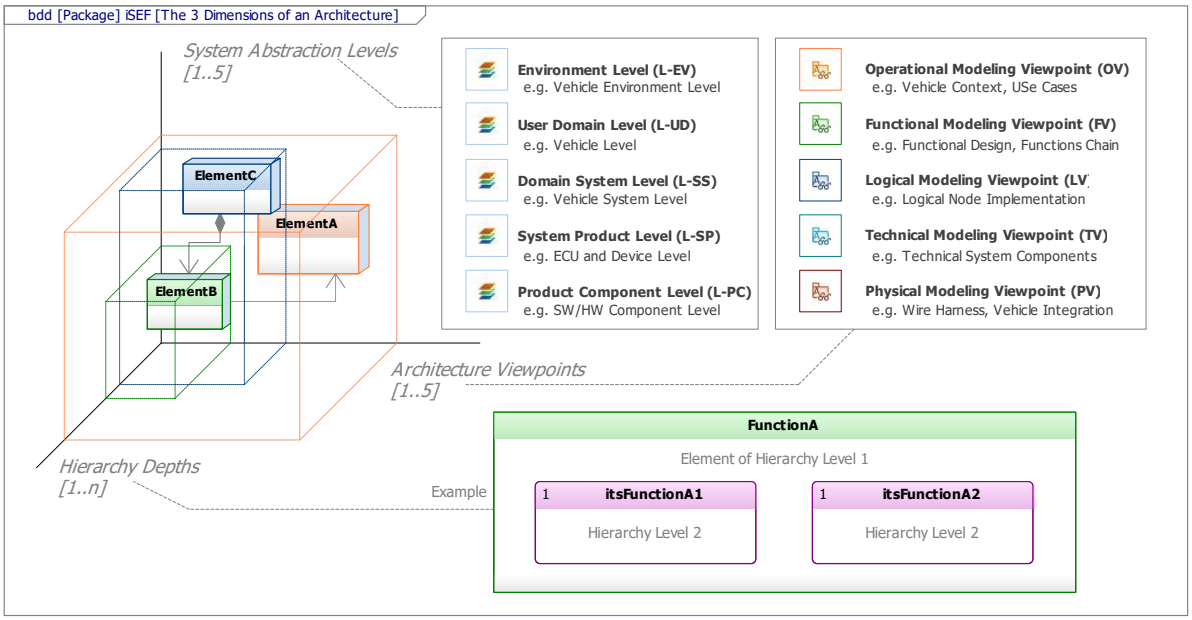


Abbildung 2: Die drei Dimensionen der Systemarchitekturbeschreibung

Mit zunehmender Komplexität von Systemen (INCOSE 2022) wird es für einen einzelnen Systemingenieur schwierig, ein umfassendes Verständnis von Funktionalität, Eigenschaften und Leistung des Systems zu überblicken. Daher müssen Systeme in kleinere Einheiten oder Subsysteme unterteilt und organisiert werden. Jedes Subsystem hat dann eine spezifische Aufgabe, die es Architekten und Designern ermöglicht, sich auf die Details dieses spezifischen Teils zu konzentrieren. Beispiele für solche komplexen Systeme sind Fahrzeuge und Flugzeuge, bei denen viele organisatorische Einheiten an verschiedenen Teilen des Systems arbeiten, um die gesamte Systemarchitektur zu schaffen.

Mit dem iSEF führt der Autor dazu drei Dimensionen der Systemarchitektur ein: die Modellsichten auf das System *(Viewpoints)*, die Abstraktionsebenen *(System Abstraction Levels)* und die Modellhierarchie Tiefe *(Hierarchy Depth)*. Diese drei orthogonalen Dimensionen z. B. bezogen auf die Fahrzeugdomäne wie in Abbildung 2 dargestellt, ermöglichen einen strukturierten Ansatz zur kohärenten Systemarchitekturentwicklung.

5 DIE MODELLSICHTEN DES iSEF

Im Rahmen der Entwicklung eines Frameworks zur Erfüllung der Bedürfnisse einer Entwicklungsorganisation hat der Autor entsprechend Abbildung 3 fünf wesentliche architekturrelevante Modellsichten (Viewpoints) identifiziert, um ein System mit starkem Fokus auf Wiederverwendbarkeit zu analysieren, zu definieren und zu erstellen. Im Fokus stehen die Belange der verschiedenen Stakeholder. Die operative Sicht (*Operational Viewpoint*), die funktionale Sicht (*Functional Viewpoint*), die logische Sicht (*Logical Viewpoint*), die technische Sicht (*Technical Viewpoint*) und die physische Sicht (*Physical Viewpoint*) auf das Modell werden eingeführt. Die direkt architekturrelevanten Viewpoints sind wie folgt definiert:

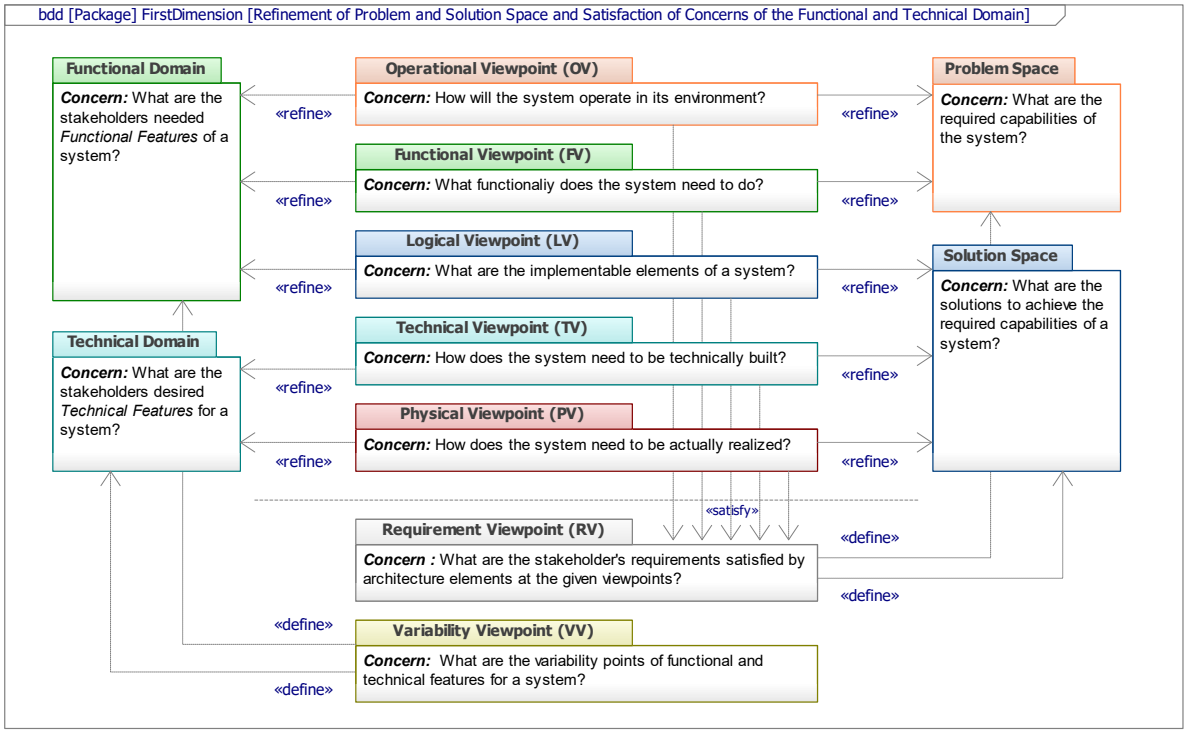


Abbildung 3: Relationen zwischen Viewpoints und dem Problem/Solution Space

Operational Viewpoint (OV):

Als Teil des Problemraums konzentriert sich der OV auf die Analyse des zu entwickelnden Systems im Einklang mit dem operativen Konzept (ISO 2 2018). Er definiert das System of Interest (SoI) innerhalb seines kontextuellen Umfelds und identifiziert seine Teilsysteme und Schnittstellen durch Ports, die die logischen und technischen Aspekte der Kommunikation darstellen. Das Hauptziel des OV ist es, eine visuelle Darstellung der Systemprozesse und Anwendungsfälle zu bieten, die die vorgesehenen Operationen und Verhaltensweisen des Systems detailliert und in Systemaktivitäten verfeinert. Weiterhin skizziert der OV die Abfolgen der Interaktionen zwischen den Systemelementen und schafft Klarheit darüber, wie die Kommunikation innerhalb des SoI erfolgt, das analysiert wird.

Functional Viewpoint (FV):

Der FV, welcher noch Teil des Problemraums ist, konzentriert sich nach der Systemanalyse auf das funktionale Design. Er definiert konsolidierte und wiederverwendbare Funktionsblöcke, die Verhaltensmodelle basierend auf Aktivitäten und Zustandsmaschinen (resultierend aus der operativen Analyse bei der OV) kapseln. Der FV ermöglicht die Definition strukturierter und standardisierter Funktionsschnittstellen mit ihren exponierten Operationen, die den Informationsfluss zwischen Funktionen erleichtern. Zudem erlaubt der FV die Visualisierung von Funktionsblöcken nach vorgegebenen Zerlegungsstrategien, um das gesamte funktionale Verhalten des interessierenden Systems (SoI) zu verstehen.

Logical Viewpoint (LV):

Der LV, bildet den Übergang zum Lösungsraum und konzentriert sich auf die Implementierung eines zielgerichteten Systems unter Berücksichtigung gegebener Einschränkungen und Plattformen. Er gruppiert die Funktionsblöcke aus dem FV in implementierbare Einheiten, bekannt als logische Knoten. Ein Hauptziel ist es, das System aus einer Implementierungsperspektive visuell darzustellen, einschließlich der Verbindungen der Knoten und der Definition oder Wiederverwendung logischer Schnittstellen mithilfe standardisierter Signale und Dienste. Zudem skizziert der LV das detaillierte Design einer logischen Einheit, wie einer Anwendung oder eines Treibers innerhalb einer spezifischen Systemumgebung, unter Berücksichtigung der gegebenen nicht-funktionalen Anforderungen.

Technical Viewpoint (TV):

Als Teil des Lösungsraums konzentriert sich der TV nun auf die Einschränkungen und Technologien, um das technische System zu konstruieren. Hauptziel des TVs ist es, eine Ausführungsumgebung für die logischen Knoten und deren integrierte Funktionen bereitzustellen. Der TV zieht die logischen Knoten aus der LV heran und verbindet sie mit den erforderlichen Hardware- (HW) und Software-Schnittstellen (SW), wodurch ausführbare Funktionen, logische Knoten und technische Komponenten miteinander verknüpft werden. Der TV fungiert als Brücke zwischen den HW- und SW-Elementen und integriert das Systemdesign innerhalb vorgegebener Hardware-spezifischen Grenzen oder technologischer Einschränkungen.

Physical Viewpoint (PV):

Der PV im Lösungsraum konzentriert sich schließlich auf die Realisierung der gewünschten Endnutzersysteme oder -subsysteme unter Verwendung verfügbarer physischer Produkte und Komponenten, wobei die nicht-funktionalen Leistungs- und Qualitätsanforderungen erfüllt werden sollen. Dieser Viewpoint ermöglicht die Visualisierung der Blockdiagramme der HW-Architektur hinsichtlich der Systemintegration mithilfe der entwickelten oder bereits vorhandenen technischen/physischen Komponenten und Elemente. Weiterhin schafft der PV eine eng gekoppelte

Verbindung zwischen der EE-Architektur und dem EE-Detailed-Design (z. B. HW-Schaltplänen), wodurch der E/E-Entwicklungsprozess optimiert wird.

Das Modellieren von Anforderung (der *Requirements Viewpoint*) als auch die Erstellung von Variabilitätsmodellen (der *Variability Viewpoint*), wie von ISO 26550 (ISO 1 2015) und durch ISO 26580 (ISO 2021) definiert, erweitern die Modellsichten entsprechend Abbildung 3 mit zwei weiteren Viewpoints welche ebenfalls integraler Bestandteil des Entwicklungsprozesses sind. Die zusätzlichen, indirekt architektur-relevanten Viewpoints sind wie folgt definiert:

Requirements Viewpoint (RV):

Der RV verwaltet und repräsentiert textuelle Anforderungen innerhalb des Modells, indem er sie mit anderen Standpunkten unter Verwendung der Zick-Zack-Methode (Weilkiens 2020) verfolgt. Er definiert Kundenanforderungen im Lastenheft als auch Systemanforderungen im Pflichtenheft, steuert die Definition von Systemelementanforderungen nach einer Systemzerlegung und unterstützt die Rückverfolgbarkeit zwischen Anforderungen und Architekturelementen. Der RV ist im iSEF-Architekturmodell von entscheidender Bedeutung und bildet eine Brücke zwischen klassischem Systems Engineering und dem ACMBSE. Der RV wird besonders wichtig, wenn Modelle nicht zwischen Lieferanten und OEMs ausgetauscht werden können und ein System nur durch eine auf Anforderungen basierende Designspezifikation ausgerichtet werden kann.

Variability Viewpoint (VV):

Der VV verwaltet Systemvarianten basierend auf einem Featurekatalog, welcher die funktionalen und technischen Merkmale (Features) eines Systems aus der Perspektive der Stakeholder oder Endnutzer beschreibt. Die VV-Modelle folgen ISO 26550 (ISO 1 2015), wobei die Abhängigkeiten hierarchisch und als auch hinsichtlich ihrer Variabilität organisiert sind. Der VV nutzt Variationspunkte an Architekturelementen, um die Variabilität eines Systems gemäß den gegebenen Features zu filtern. Ein Variabilitätsmodell innerhalb des VV beeinflusst weitere Architekturelemente hinsichtlich ihrer Sichtbarkeit für Benutzer-wahrnehmbare Funktionen und technische Lösungen. Der VV ist entscheidend für das Management von Plattform- und Produktlinien, was besonders nützlich für komplexe Systeme mit vielen Varianten ist, um die Wiederverwendbarkeit innerhalb von Ingenieursorganisationen zu steigern (Forlingieri 2022).

Das zugrunde gelegte iSEF-Modellierungsprofil (iSefML) basiert auf der SysML Sprache und erleichtert die Nutzung aller oben genannten Modelliersichten, optimierbar für variable Systementwicklungsprozesse. Das Profil umfasst einen angepassten Modellbrowser, nutzt Zeichnungswerkzeuge, die für die erforderliche Elementnutzung modifiziert sind, und implementiert interaktive Regelwerke, die die Richtigkeit der Zusammensetzung und Interaktion zwischen Modellelementen bereits während der Modellierung sicherstellen.

6 DIE ABSTRAKTIONSEBENEN DES iSEF

Der technische und funktionale Umfang eines Systems und wie tief in das System hineingeschaut werden soll, wird durch die System Abstraction Levels definiert, welche eine künstliche Abgrenzung, aber keine inhärente Eigenschaft des Systems bilden. Sie dienen viel mehr als Kategorie oder Klassifizierung jedes möglichen Systems (hier mit Fokus auf fahrzeugbezogene Systeme), wie in Abbildung 4 dargestellt. Eine Systemabstraktionsebene kann sich auf den Prozess der Abstraktion oder auf die Verallgemeinerung physischer, räumlicher oder zeitlicher Details eines Systems beziehen oder die Abstraktion konzentriert sich stattdessen auf ein benutzerwahrgenommenes Modell oder eine Darstellung von Informationen aus der realen Welt. Bei dem iSEF werden die Systemabstraktionsebenen gemäß dem in Abbildung 4 gezeigten Frameworks definiert, wobei 5 Abstraktionsebenen bezogen auf eine Domäne des Systems herangezogen werden. Diese sind:

Environment Abstraction Level (L-EV):

Der L-EV umfasst den Kontext eines System Of Interest (SoI) z. B. Fahrzeugumgebung, Mobilfunksystem, welche mit ihrer unmittelbaren Betriebsumgebung interagieren. Der L-EV bezieht sich dabei auf die Interaktion mit dem Benutzer und die entsprechenden Umwelteffekte, auf die das System reagieren oder die es stimulieren kann. Ferner beinhaltet diese Abstraktionsebene die Modellierung der Interaktionen verschiedener unabhängiger Systeme im Zusammenschluss der Funktionalität eines System of Systems (z. B. Verkehrsleitsysteme, Flottenmanagementsystems und Cloud-Lösungen).

User Domain Abstraction Level (L-UD):

Der L-UD umfasst ein System, einschließlich seiner zusammengesetzten Teilsysteme innerhalb einer Domäne (z. B. Fahrzeug, Smartphone, Flugzeug etc.). Der L-DU bezieht sich dabei auf ausschließlich auf ein System, mit dem ein Benutzer interagieren oder ein System, welches direkt auf Umwelteinflüsse reagieren kann. Die physische Größe oder die Komplexität der Systeme sind hierbei nicht entscheidend. Ein Flugzeug z. B. hat gegenüber einem Smartphone nur eine wesentlich höhere Anzahl von Subsystemen auf den nachfolgenden Abstraktionsebenen.

Domain System Abstraction Level (L-DS):

Der L-DS umfasst Teilsysteme (z. B. Advanced Driver Assistance Systems), eines Benutzersystems innerhalb einer Domäne, die sich auf einen spezifischen Funktionssatz, auf die Kommunikation oder Technologien konzentrieren, die für den Endbenutzer nicht mehr direkt sichtbar sind. Auf diesem Abstraktionslevel werden typischerweise integrative Teilsysteme definiert, die mindestens aus 2 oder mehr Systemprodukten bestehen. Beispiele sind unter anderem eine Fahrzeugtür mit allen mechanischen, elektromechanischen und elektronischen Baugruppen oder auch ein Mobilfunkempfangssystem in einem Smartphone.

System Product Abstraction Level (L-SP):

Der L-SP umfasst aller technischen Systeme innerhalb eines Teilsystems, z. B. der Fahrzeugdomäne, welche durch ihre Zusammensetzung (Integration) aller disziplinspezifischen Komponenten (SW, EE, ME) definiert sind, um ein physisches System als Produkt zu bauen. Dazu zählen typischerweise Steuergeräte (ECUs) und technische Aktoren sowie Sensoren, aber auch Funktionen, die in diese Systeme integriert werden sollen.

Product Component Abstraction Level (L-PC):

Der L-PC umfasst alle Komponenten, die innerhalb einer einzigen Disziplin beschrieben werden können, um ein Systemprodukt zu realisieren. Dazu gehören auch die interdisziplinären Schnittstellen wie das Hardware-Software-Interface oder Kommunikationsprotokolle, an denen die Komponenten beteiligt sind. Die Disziplin-spezifischen Komponenten bilden dann den Startpunkt für eine weitere Zerlegung der Komponenten in den spezifischen HW- und SW-Architekturbeschreibungen.

Jedes System und deren Subsysteme kann grundsätzlich in eine der Systemabstraktionsebenen eingeordnet werden. Wenn das System zerlegt wird, kann ein aus einem Zerlegungsschritt resultierendes Teilsystem in eine niedrigere Systemabstraktionsebene als das ursprüngliche übergeordnete System gelten, wenn keine weitere Hierarchie angewendet wurde. Eine Entwicklung beginnt dabei immer auf der Systemabstraktionsebene des Systemkontexts, wobei die L-EV-Ebene nicht unbedingt eingeschlossen sein muss. Hier entscheidet das System of Interest, welche Systemabstraktionsebene für den Kontext und damit als Startpunkt herangezogen werden soll.

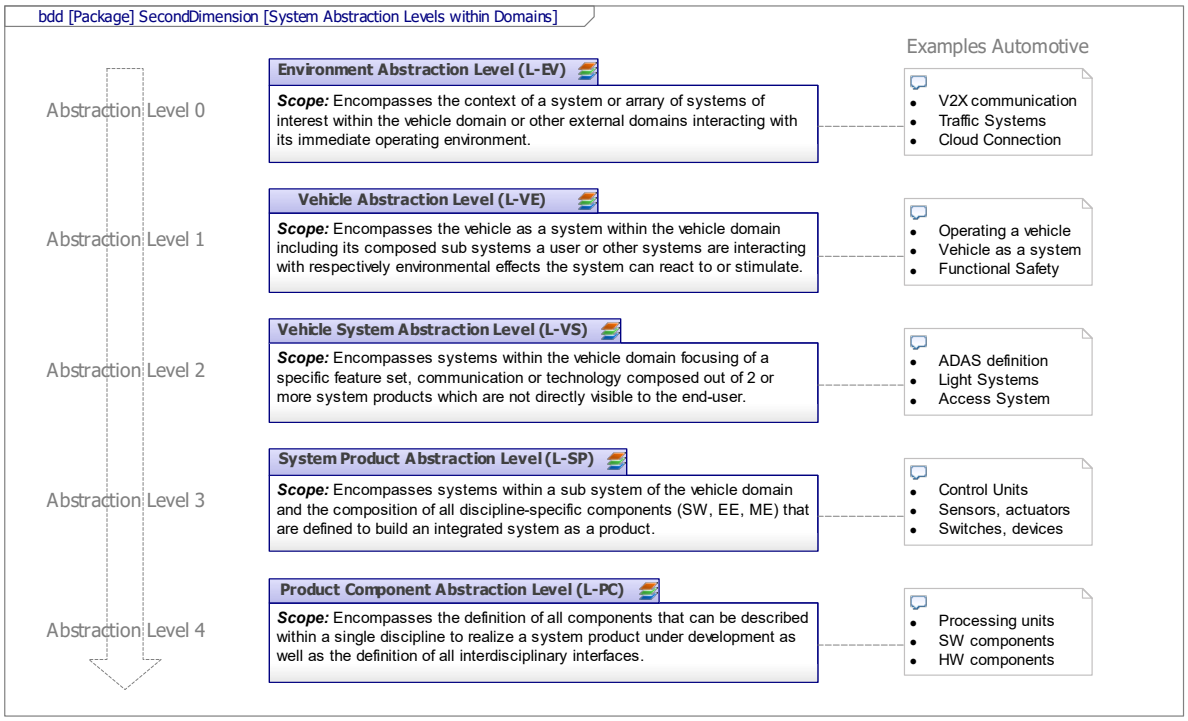


Abbildung 4: Exemplarisch die 5 applizierten Abstraktionsebenen für die Fahrzeugdomäne

Die Verwendung von Systemabstraktionsebenen bietet mehrere Vorteile. Auf jeder Systemabstraktionsebene kann ein neuer Systemkontext abgeleitet werden, der eine klare Interaktionen zwischen Systemelementen wie Funktionen, Knoten und Komponenten ermöglicht. Dabei ist eine Kommunikation (Interaktion) der Elemente nur auf derselben Abstraktionsebene zulässig, was ein "Level-Hopping" zwischen Systemen und Komponenten auf verschiedenen Abstraktionsebenen verhindert.

Ebenso ermöglicht die Verwendung von Systemabstraktionsebenen Dekompositionsmodelle mit präzisen formulierten Regeln, die festlegen, welche Elementmetaklassen auf welchem Abstraktionslevel verwendet werden dürfen. Das erlaubt dann die Implementierung von maschinell überprüfbaren Regeln aller Architekturelemente in der Architekturbeschreibung durch werkzeuggestützte Mechanismen, die vom iSEF-Profil unterstützt werden. Dies stellt letztlich sicher, dass die Systemmodellierung gültig ist und vordefinierten Zerlegungs- und Integrationsschemata folgt. Die Anwendung dieser Praxis erleichtert die Kommunikation von Systemingenieuren aus verschiedenen Bereichen und Unternehmen (wie Lieferanten), da sie sich auf die entsprechende Abstraktionsebene konzentrieren können. Klare Übergaberegeln regeln dabei den Austausch der Elemente. So wie es auf jeder Abstraktionsebene entsprechende Lasten- und Pflichtenheft geben muss, werden auch zugehörige Lasten- und Pflichtenmodelle notwendig.

7 HIERARCHY DEPTH

Wird ein System auf einer Abstraktionsebene analysiert und über alle Viewpoints hinweg modelliert, wird von einem vollen Architekturschritt gesprochen. Innerhalb dieses Prozesses können die System- und Funktionsbausteine eine unterschiedliche Granularität auf den verschiedenen Viewpoints aufweisen.

Hierarchietiefen definieren also die Granularität einer Zerlegung eines Systems innerhalb der tatsächlichen Systemabstraktionsebene eines bestimmten Viewpoints, wie in Abbildung 5 dargestellt.

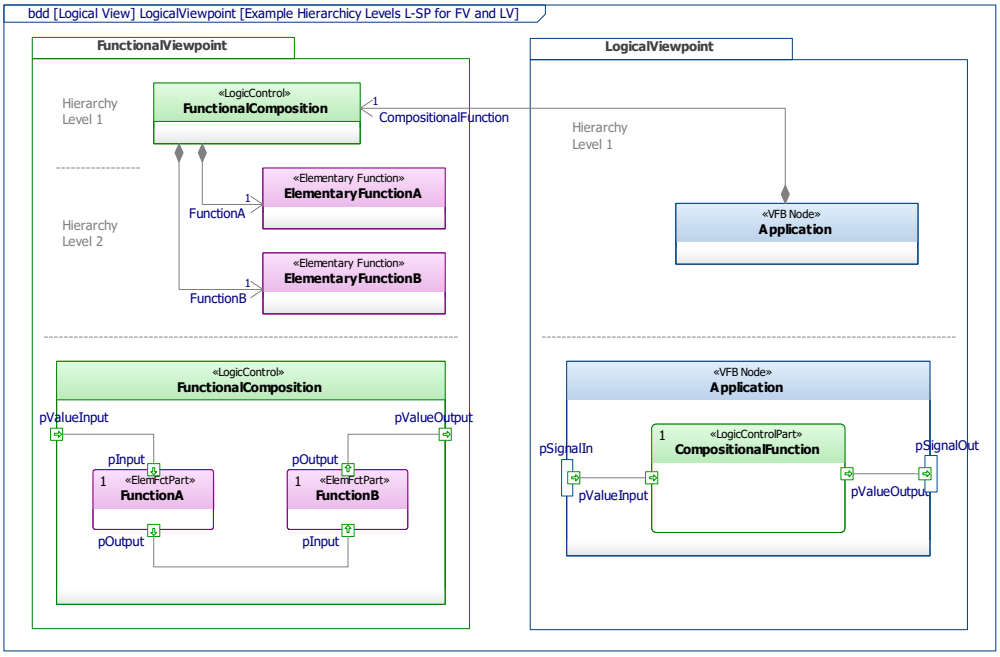


Abbildung 5: Beispielhafte Hierarchietiefe von „2“ des FV und „1“ des LV auf dem L-PC

Die niedrigste und bevorzugte Anzahl an Hierarchietiefen ist 1, da dies dazu beiträgt, die Systemzerlegung einfach zu halten. Es ist jedoch wichtig zu beachten, dass die Hierarchietiefe zwischen den Viewpoints innerhalb einer gegebenen Systemabstraktionsebene nicht konstant sein muss. Dies liegt daran, dass die Granularität der Zerlegung je nach angewendetem Zerlegungsschema für jeden Viewpoint variieren kann und sich dadurch verschiedene Hierarchietiefen ergeben.

Um eine standardisierte Systemzerlegung sicherzustellen, bietet das iSEF eine Reihe von Referenzzerlegungsmodellen an, mit denen die Architekten angeleitet werden, das System und die Funktionen so zu zerlegen, dass kompatible und wiederverwendbare Bausteine entstehen.

8 VERKNÜPFUNG VON ELEMENTEN ZWISCHEN DEN MODELLSICHTEN

Um eine kohärente Architekturbeschreibung über verschiedene Standpunkte hinweg sicherzustellen, ist es notwendig, Verknüpfungsmuster zwischen den architektonischen Elementen derselben Modellierungssicht als auch zwischen den architektonischen Elementen einer anderen Modelliersicht herzustellen. Angesichts dessen führt die iSEF-Methode erweiterte SysML-Semantiken ein. Innerhalb des Frameworks sind Elemente aus verschiedenen Viewpoints miteinander verknüpft, indem funktionale Elemente direkt in logische Elemente und diese anschließend in technische Elemente durch Komposition integriert werden. Diese technischen Elemente werden weiter in physische Komponenten eingesetzt.

Die Verknüpfung aller beteiligten Architekturelemente, wie in Abbildung 6 dargestellt, wird durch ihre miteinander verbundenen Schnittstellen verstärkt, ausgerichtet und durch einen Modelchecker auf Konsistenz überprüft. Die Konsistenz wird durch eine verschachtelte Portmodellierung erreicht, indem wie in Abbildung 6 Proxy-Ports über eine Interface-Definition in Full-Ports eingebettet sind und ein oder mehrere funktionale Werte durch ein logisches Signal gekapselt werden. Dieses Signal wird anschließend über eine technische Schnittstelle übertragen, die das Signal entlang einer technischen Leitung weiterleitet, welche wiederum an eine physische Schnittstelle gebunden ist, welches die Lösung zur Gesamtverbindung über alle Standpunkte hinweg repräsentiert.

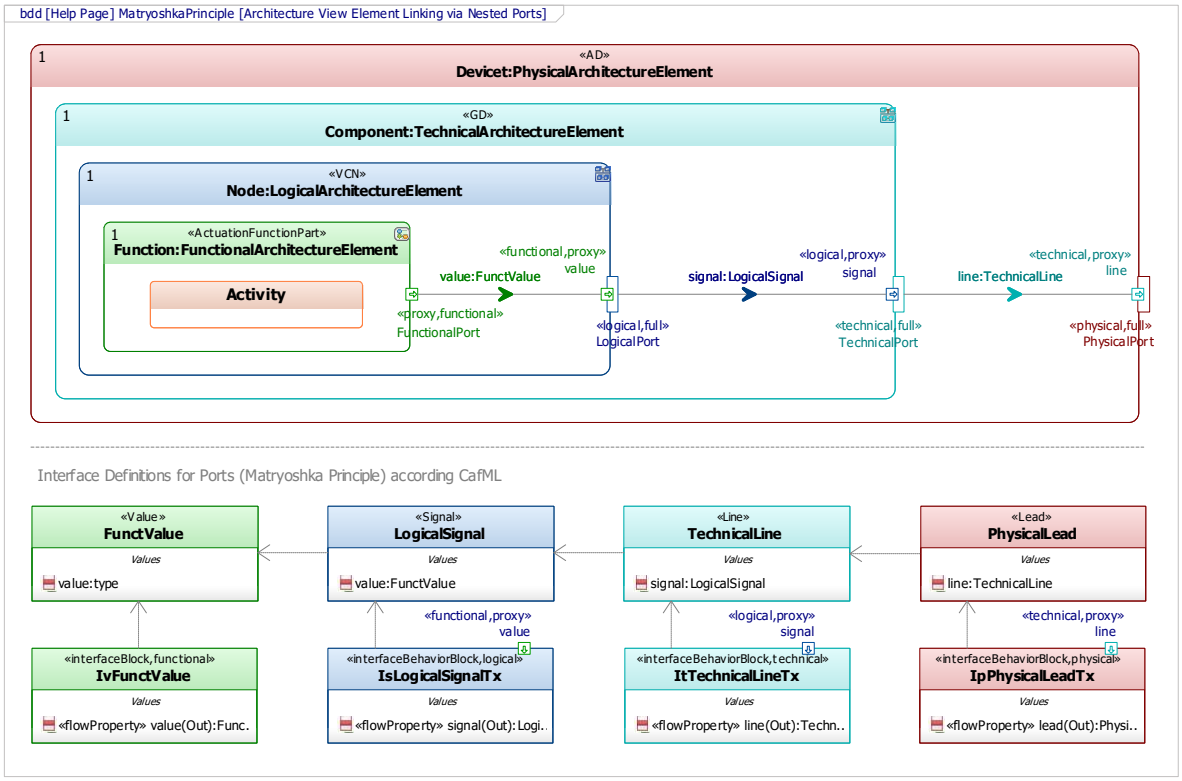


Abbildung 6: Linken von Elementen und Interfaces aufeinanderfolgender Viewpoints

Die technischen und physikalischen Einschränkungen, die an diesem Modellierschritt eine Rolle spielen, bestimmen nun, dass die physische Architektur auf technologischen Entscheidungen aus dem technischen Viewpoint beruht. Des Weiteren hängt die technische Modellierung von architektonischen Entscheidungen ab, die vom logischen Viewpoint getroffen wurden. Die logische Modelliersicht wiederum spiegelt eine Implementierung basierend auf einem spezifizierten funktionalen Design wider, das vom funktionalen Viewpoint erstellt wurde. Das funktionale Design aus dem funktionalen Viewpoint wird ebenfalls als Ergebnis einer operativen Analyse angesehen, welche die Aktivitäten und Anwendungsfälle bewertet hat, die im operativen Standpunkt skizziert wurden.

Abbildung 6 veranschaulicht ebenfalls die Verwendung des zur SysML zusätzlich eingeführten iSefML-Stereotyps *InterfaceBehaviorBlock*. Dieser spezifische Interfaceblock ermöglicht die Verbindung der Elemente als auch die formale Übertragung von Information über Viewpoints hinweg, statisch als auch dynamisch (Simulation), welches die Erstellung semiformaler Architekturen und Designs ermöglicht, die den höchsten Sicherheitsintegritätsstufen entsprechen (ISO 26262, 2018). Modellierungswerkzeuge wie Rhapsody können diese Prinzipien applizieren, um Notationen und semantische Verifizierungen zu automatisieren, wodurch zusätzlich eine maschinelle Modellverifizierbarkeit erreicht wird.

Weitere Einblicke in die Verbindung zwischen Ansichten und Systemzerlegung finden sich in einer früheren Veröffentlichung von Continental (Seidel & Forlingieri, 2023).

9 METHODISCHE MODELLIERUNGSSCHRITTE ZUM SYSTEMDESIGN

Nach einer kurzen Einführung in die Prinzipien für den Aufbau einer Systemarchitektur, einschließlich der zuvor eingeführten sieben Modellsichten, zeigen der Autor die Systemmodellierung unter Verwendung des ACMBSE einen möglichen Arbeitsablauf, der unter anderem die Entwicklung eines Systemmodells mithilfe des iSEF zeigt. Das im folgenden Abschnitt dargestellte Beispiel konzentriert sich auf die Gestaltung und Implementierung der äußeren Beleuchtungsfunktionen eines Fahrzeugs, basierend auf einem tatsächlichen Entwicklungsumfang. Zur Vereinfachung wird in diesem Papier nur ein kleiner Teil des gesamten Umfangs dargestellt.

Der Autor zeigt in einer Demonstration acht grundlegende Modellierschritte von der Analyse zum Design, die wie folgt aufgeführt sind:

1. Feature-Definition unter Verwendung des *Variability Viewpoints* (VV) durch Definition von funktionalen und technischen Features und deren Beziehung hinsichtlich der Variabilität, gesteuert durch Variationspunkte

2. Systemkontext und Definition des System of Interests unter Verwendung der *Operational Viewpoints* (OV), einschließlich der Komposition und Interaktion mit externen Elementen (Akteuren)

3. Anforderungsableitung unter Verwendung des *Requirements Viewpoints* (RV) einschließlich der Rückverfolgbarkeit von Systemelementen, Designeinschränkungen, Schnittstellen und anderen Designaspekten

4. Verhaltensanalyse mittels des *Operational Viewpoints* (OV) aus der Perspektive des erwarteten Systemverhaltens, einschließlich einer Anwendungsfallanalyse und rekursiver Systemaktivitätsverfeinerung

5. Entwicklung des funktionalen Designs unter Nutzung der *Functional Viewpoints* (FV) einschließlich Zustandsmaschinendefinitionen, Aktivitätsverfeinerungen und vollständiger funktionaler Komposition

6. Logische Komponentenentwicklung unter Anwendung der *Logical Viewpoints* (LV) zur Erstellung von strukturellen Systemkompositionen, Schnittstellendefinitionen mit Portverknüpfung für ein konkretes Zielsystem

7. Definition der technologischen Lösung des Systems mittels seiner Subkomponenten unter Nutzung der *Technical Viewpoints* (TV), inklusive der Integration logischer Architekturelemente für den Link des Problemraums zum Lösungsraum

8. Realisierung der physischen Architektur des Endnutzersystems unter Verwendung real vorhandener oder neudefinierter implementierbarer physischer Komponenten und Baugruppen inklusive der Definition physischer Schnittstellen

Die folgenden Modelle werden die Entwicklung einer fiktiven Body Control Unit (BCU) vereinfacht aufzeigen. Es beginnt auf Fahrzeugebene, indem die benötigten Fähigkeiten des Systems identifiziert, die Systemfunktionen und Schnittstellen gestaltet und die Funktionalitäten und Komponenten auf Produktebene entwickelt werden. Schließlich werden diese Komponenten in eine physische elektronische Steuereinheit (ECU) integriert, die unter anderem eine Reihe von Fahrzeuglichtfunktionen implementiert, die als illustratives Beispiel für die Anwendung der ACMBSE-Methode dienen.

9.1 SCHRITT 1 - FEATURE DEFINITION DES SYSTEMS

Bevor der Entwickler mit der Definition einer Systemarchitektur beginnt, ist es entscheidend, ein klares Verständnis der wesentlichen Attribute zu haben, die das System oder die Systemproduktlinie besitzen muss, um den Bedürfnissen des Endnutzers gerecht zu werden. Dies beinhaltet eine Analyse der erforderlichen *funktionalen* und *technischen Features*, die unter Nutzung des eingeführten *Variability Viewpoints* (VV) gemäß dem von Forlingieri und Weilkiens (2022) beschriebenen Modellierungsansatz.

In Übereinstimmung mit dem Verständnis charakterisieren *Functional Features* ein System of Interest (SoI) aus einer funktionalen Perspektive, die für Endnutzer oder Stakeholder ohne tiefes Verständnis des Systemdesigns und -verhaltens verständlich ist. *Functional Features* stellen eine Synthese der Needs an ein SoI bzw. Fähigkeiten eines SoI dar und dienen als Grundlage für abzuleitende funktionale Anforderungen, wie auch in Beuche et al. (2004) und Forlingieri (2022) gezeigt.

Technical Features charakterisieren ein SoI aus einer technischen Perspektive hinsichtlich Leistung und Technologie, die für Endnutzer oder Stakeholder ohne tiefes Verständnis der zugrunde liegenden Implementierung verständlich ist. Sie sind eine Synthese technischer Beschränkungen, Leistungsmöglichkeiten oder technologischer Einschränkungen für das gewünschte SoI und dienen als Grundlage für abzuleitende nicht-funktionale technische Anforderungen.

Technical Features beschränken oft basierend auf technischen Randbedingungen in Abhängigkeiten zu funktionalen Features die Systemmerkmale. Diese Abhängigkeit (Beuche et al., 2004; Forlingieri 2022), ist in Abbildung 7 z. B. durch die *includes* Beziehung veranschaulicht.

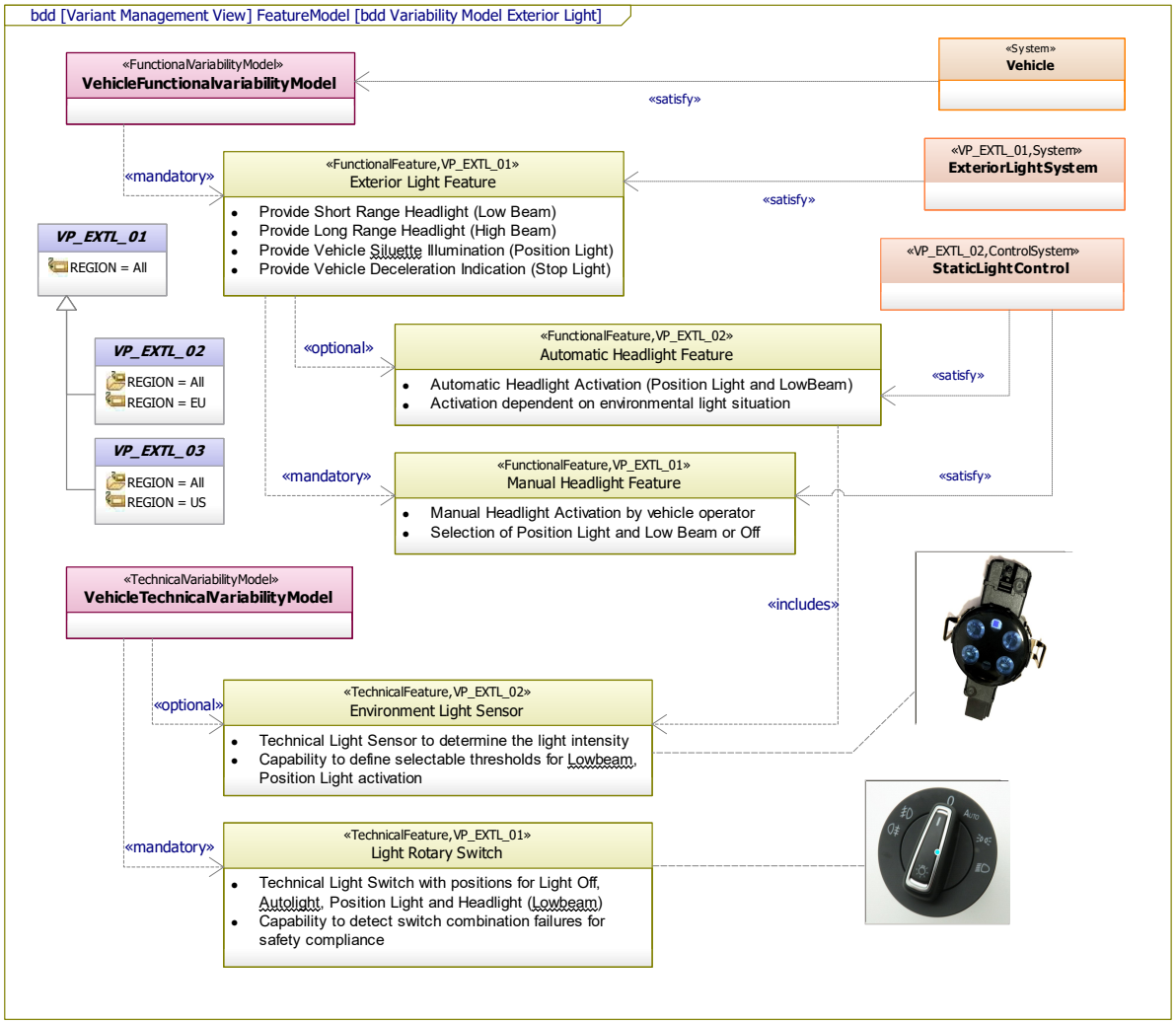


Abbildung 7: Modellierung des Variability Models mit Functional und Technical Features

Functional und *Technical Features* können ebenfalls wie in Abbildung 7 hinsichtlich ihrer Variabilität modelliert werden, die durch eine *Dependency*-Beziehung wie „mandatory“ Merkmale, optional“ oder „alternative“ Merkmale sowie Beziehungen zu anderen Merkmalen wie „ausschließend“ oder „einschließend“ gekennzeichnet sind. Beispielhaft dafür sind in Abbildung 7 „mandatory“ „Manual Headlight Feature“ als auch das optionale „Automatic Headlight Feature“, dargestellt, die durch eine *User Perceivable Function* „Static Light Control“ realisiert werden. Zusätzlich wird ein technisches Feature, das einen „Environment Light Sensor“ zur Messung der Lichtintensität umfasst, beispielhaft gezeigt, dass die „einschließende“-Abhängigkeit zwischen den gegebenen funktionalen und technischen Features darstellt. Variationspunkte (z. B. „VPA-EXTL-02“), die an jedem Variationspunkt in der Hierarchie der Features definiert sind, können dann verwendet werden, um die Varianten zu adressieren.

Danach können diese Variationspunkte zu jedem Architekturartefakt hinzugefügt werden, um die Relevanz einer Variante eines Architekturelements zu klären, wie im „Static Light Control“-Funktionsblock dargestellt. Zusammen ermöglichen das Modellierungswerkzeug und das bereitgestellte Profil eine Plausibilitätsmodellprüfung und eine Visualisierung von Feature-spezifischen Inhalten, die auf Diagrammsichten in Rhapsody angewendet werden können.

9.2 SCHRITT 2 - SYSTEM & KONTEXT DEFINITION

In der iSEF-Methode ist der nächste Schritt zur Entwicklung einer Systemarchitektur die Definition des Systemkontexts mit dem darin enthaltenen SoI. Dies wird durch das Modellieren der entsprechenden Systemelemente in einem operationalen Blockdefinitionsdiagramm (OBDD) durchgeführt. Damit sollten dann alle Akteure im Systemkontext identifiziert sein.

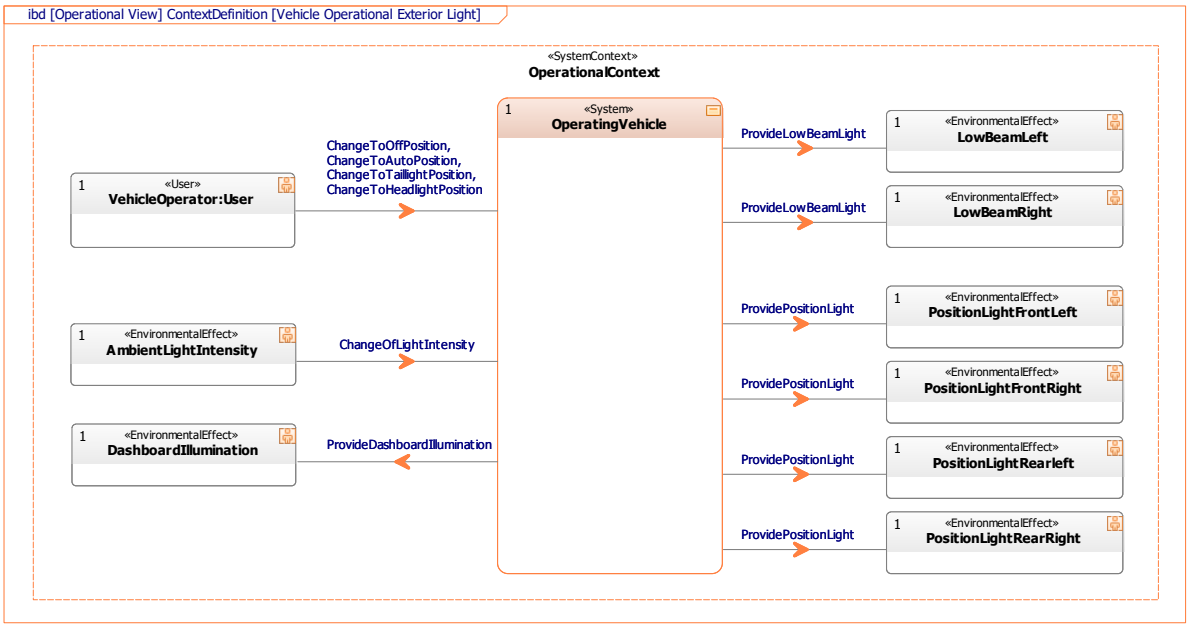


Abbildung 8: Kontextdefinition and Functional Light System Identifikation

Der nächste Schritt umfasst die Spezifikation von Interaktionen zwischen dem SoI und den Kontextelementen (Akteuren) unter Verwendung eines oder mehrerer operationaler interner Blockdiagramme (OIBD), wie in Abbildung 8 dargestellt, wobei das „Operating Vehicle“ das SoI ist. Die grauen Elemente sind die Kontextelemente in dieser Definition, wie in Abbildung 8 gezeigt. Die Identifizierung der abstrakten operativen Ereignisse (auf der Umweltebene) ist entscheidend. Abbildung 9 stellt eine Kontextdefinition dar, die die operativen Ereignisse für ein Lichtsystem verwendet.

Die in Abbildung 7 gezeigten *Technical Features*, welche die Grundlage für den Aufbau des SoI bilden, werden ebenfalls herangezogen und sind besonders wichtig, um das SoI zu verstehen, da sie die weiteren Entwicklungsschritte für das System und die Subsysteme einschränken und die Schnittstellen bilden, die später physisch realisiert werden müssen. Vorgegebene operationale Events können auch für die folgende operative Analyse (siehe Abbildung 8) genutzt werden, um Benutzerinteraktionen zwischen dem SoI und den Kontextelementen zu identifizieren, wodurch die Definition von Anwendungsfällen konsistent erleichtert wird.

Sind bereits Schnittstellen aus einem vorhergehenden Architekturschritt hervorgegangen und damit Teil der Anforderungen für die gegenwärtige System-Abstraktionsebene, können diese schon in die Architektur in Form von technischen und logischen Interfaces eingebaut werden.

9.3 SCHRITT 3 - REKURSIVES REQUIREMENTS ENGINEERING

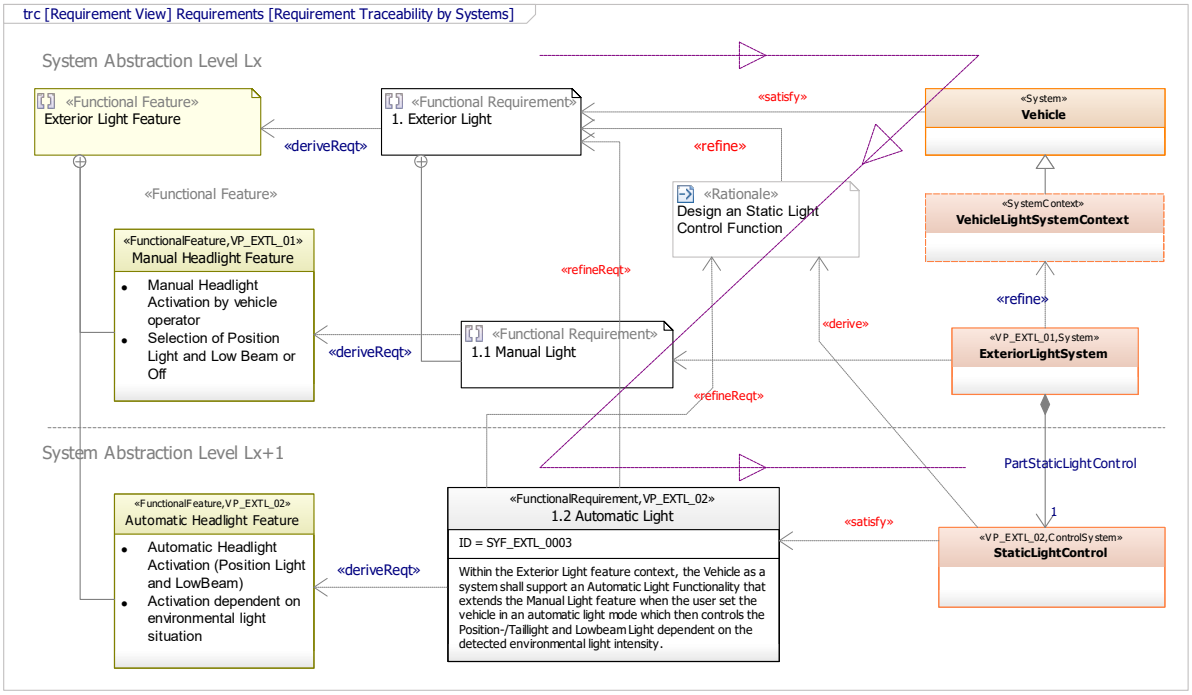


Abbildung 9: Nachverfolgbarkeit zwischen Architekturelementen und Anforderungen

Das iSEF verwendet rekursiv die Zig-Zag-Methode (Weilkiens 2008) zur Erstellung von Systemanforderungen, kombiniert mit den Prinzipien des ACMBSE. Abbildung 9 veranschaulicht entsprechend die Rückverfolgbarkeit zwischen Features, Anforderungen und Architekturelementen auf der relevanten Abstraktionsebene des Systems (Seidel & Forlingieri 2023).

Das Anwenden der Zig-Zag-Methode funktioniert im Prinzip, indem das illustrierte „Z“ in Abbildung 9 verfolgt wird: Auf der initialen Systemabstraktionsebene (Lx) werden dazu Anforderungen erstellt, welche den Umfang des Black-Box-Systems „Vehicle“ widerspiegeln (im Beispiel erfüllt das „Vehicle“-System die „Exterieur Light“-Anforderung). Als nächstes wird ein Architektur-Schritt durchgeführt, um das „Vehicle“-System in kleinere Teile oder Teilsysteme zu zerlegen (beispielhaft in Abbildung 9 durch das „Static Light Control“ gezeigt). Die Architekturentscheidungen werden dazu in *Rationales* dokumentiert, welche ebenfalls aus den Anforderungen des „Vehicle“-Systems verfeinert (*refined*) werden. Diese ‚Rationales‘ bilden dann die Grundlage für Ableitung von Anforderungen auf der nächsten Systemabstraktionsebene (Lx+1) für die Subsysteme (StaticLightControl), wie in Abbildung 9 illustriert. Dieser Wechsel zwischen Anforderungsbeschreibung und Architekturprozess wird rekursiv über alle Systemabstraktionsebenen hinweg wiederholt, bis ein Teilsystem innerhalb einer einzigen Disziplin (EE oder SW) implementiert werden kann.

9.4 SCHRITT 4 - OPERATIONALE ANALYSE DES SYSTEMS

Die Durchführung einer operationalen Analyse durch Modellieren von Anwendungsfällen ist ein wesentlicher Schritt zur Identifizierung der Aktivitäten und Zustandsverhalten des SoI. *Das Use Case Diagram* hilft dabei, Anwendungsfälle aus funktionalen Anforderungen zu verfeinern und zugrunde liegende Systemaktivitäten abzuleiten. Diese Aktivitäten können auf verschiedenen Abstraktionsebenen innerhalb der operativen Analyse weiter verfeinert werden. Der iSEF-Leitfaden bietet dazu spezialisierte Metaklassen wie *System Process*, *System Use Case*, *Secondary Use Case* und *Continuous Use Case* an, wie in Weilkiens (2008) beschrieben.

Die Anwendungsfallanalyse ist hilfreich, wenn Teilsysteme noch unbekannt sind und unklar ist, wie die Systemzerlegung aussehen könnte. Wenn jedoch die Systemzerlegung bereits aufgrund ausreichend detaillierter Anforderungen oder technischer Einschränkungen vordefiniert ist, ist eine Anwendungsfallanalyse nicht zielführend, um weitere Informationen zu elaborieren. In solchen Fällen ist es besser, das SoI direkt in bereits identifizierte Teilsysteme aufzubrechen und die operative Analyse auf der nächsten Systemabstraktionsebene durchzuführen, wo keine weitere strukturelle Zerlegung mehr erfolgt.

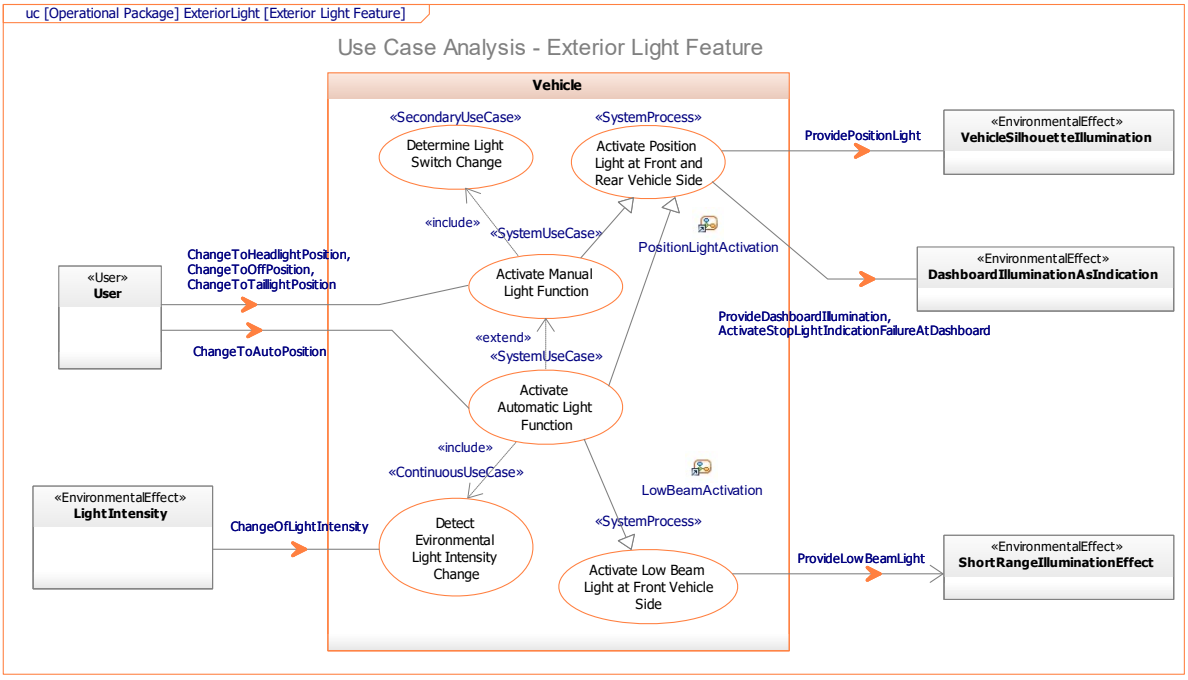


Abbildung 10: Operationale Analyse mit darunterliegenden Systemaktivitäten

Als Besonderheit der Methodik zeigt das Beispiel in Abbildung 10 eine Anwendungsfallanalyse, welche die zuvor definierten operativen Ereignisse wiederverwendet, um die Events der gegebenen Kontextelemente mit den Systemanwendungsfällen konsistent zu modellieren. Die dahinter liegende Systemaktivität z. B. „Low Beam Activation“ bildet dann einen weiteren Analyseschritt für eine detaillierte Verhaltensverfeinerung, die eine Zustandsmaschinenerstellung ermöglicht.

Durch die Modellierung mit iSEF ist es insbesondere möglich, das beabsichtigte, wenn auch hier vereinfachte Verhalten von Aktivitäten und Zustandsmaschinen während der Analysephase zu simulieren. Es können dann bereits, Systemdetails mit Stakeholdern diskutiert werden, um unnötige Entwicklungszyklen zu vermeiden.

9.5 SCHRITT 5 - DETAILLIERTES FUNKTIONALES DESIGN

Entsprechend der als Leitfaden verwendeten Harmony-Prinzipien (Douglas 2017) und zusätzlichen iSEF-Erweiterungen werden Systemaktivitäten, die im vorherigen Schritt der operationalen Analyse formuliert wurden, zu *Essential Activities* bzw. *Node Activities* verfeinert und in funktionale Blockelemente (z. B. in einen *Logic Control* Funktionsblock) auf derselben Abstraktionsebene oder eine Ebene darunter platziert (Details zur Use Case Analyse werden in einem späteren Whitepaper noch genauer erläutert werden. Swimlanes und Funktionsblöcke, die in der aufrufenden *System Activity* modelliert sind, erleichtern die Verknüpfung operativer Ansichtselemente mit funktionalen Elementen.

Beim Verfeinern von Systemaktivitäten, während des funktionalen Designschritts in ausführbare *Node Activities* bzw. *State Machines,* werden die operationalen Events aus dem operationalen Viewpoint typischerweise in einen Informationsfluss unter Verwendung der SysML-Proxy-Ports transformiert. Solche Schnittstellendefinitionen, wie beispielhaft in Abbildung 11 gezeigt (z.B. „IAutoLightRequest“), werden dann durch einen Port eines Funktionsblocks (z.B. „Headlight Control“) implementiert. Da die meisten eingebetteten Systeme Zustandsmaschinen enthalten, wird das Beispiel in Abbildung 11 den Punkt veranschaulichen. Die Zustandsmaschinen werden durch Verfeinerung zuvor bestimmter Aktivitäten und Ereignisse detailliert modelliert, um die gezielten Datenelemente, Operationen und Schnittstellen zu nutzen und sogar zu simulieren. Ebenso bewertet die Zustandsmaschine die „LightRequest“ aus Abbildung 8. Wenn alle Datenelemente und das Verhalten präzise gestaltet sind, kann das System simuliert werden, um das beabsichtigte Verhalten des Funktionsblocks zu überprüfen.

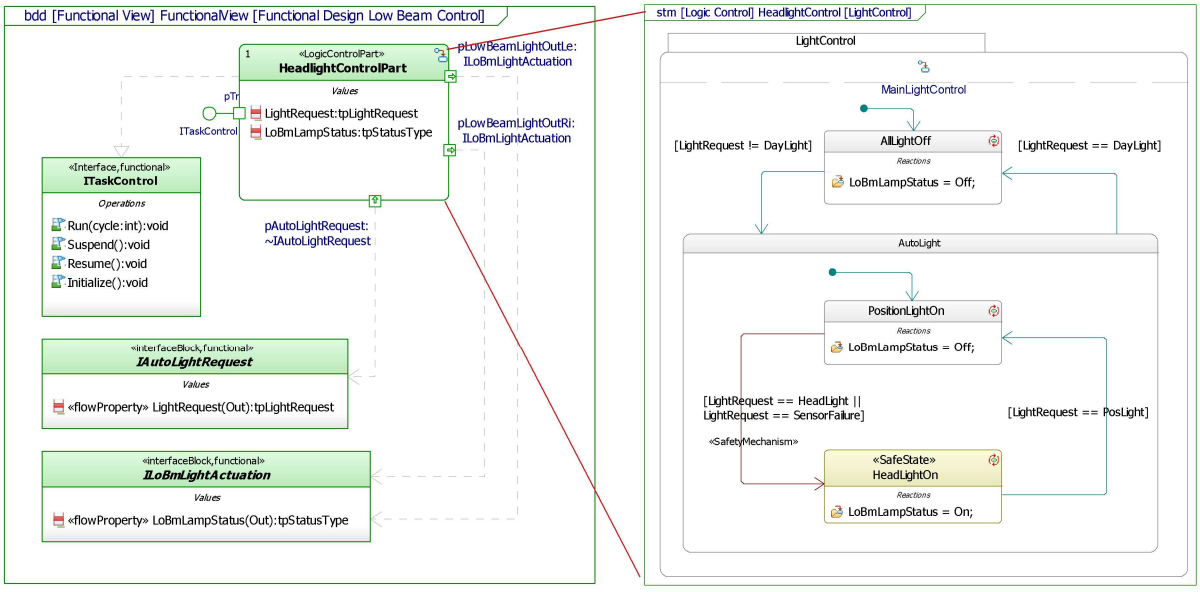


Abbildung 11: Funktionales Design des Low Beam Light Control

Aus der funktionalen Sicht ist es wichtig zu beachten, dass viele der im iSEF entwickelten Funktionen für eine hohe Wiederverwendbarkeit vorgesehen sind, die durch die Verwendung von Ports zur Kommunikation zwischen funktionalen Elementen erreicht werden kann, um eine Unabhängigkeit der Funktionsblöcke zu gewährleisten. Zusätzlich zu den in Abbildung 11 illustrierten Aspekten für das „ITask Control“ Interface betont die funktionale Sicht auch die Nutzung vordefinierter Schnittstellen zum Austausch funktionaler Werte und die Implementierung von Funktionsaufrufen zur Unterstützung von Operationen (APIs), die dann in serviceorientierte Schnittstellen auf der logischen Sicht später integriert werden.

9.6 SCHRITT 6 - DESIGN LOGISCHER KOMPONENTEN

Das primäre Ziel der logischen Modellsicht ist es, funktionale Elemente in logische Einheiten, benannt als logische Knoten, zu organisieren und diese Knoten in größere logische Systeme zu integrieren. Diese Knoten enthalten kein zusätzlich explizit implementiertes Verhalten, da alle Verhaltensmodelle bereits in funktionalen Elementen entwickelt wurden. Abbildung 12 veranschaulicht beispielhaft ein zusätzlich integriertes „Anwendungsmanagement“, das andere Funktionsblöcke steuert, aber nicht zu einem zusätzlichen funktionalen Benutzerverhalten beiträgt.

Funktionale Elemente können typischerweise innerhalb des Knotens mit anderen Funktionalitäten ebenfalls unter Verwendung von Ports kommunizieren. Dieser Ansatz ist nicht obligatorisch, ermöglicht jedoch eine flexible Umgestaltung von funktionalen Elementen, die unabhängig entwickelt werden können, was die Wiederverwendung und Standardisierung des Designs maximiert.

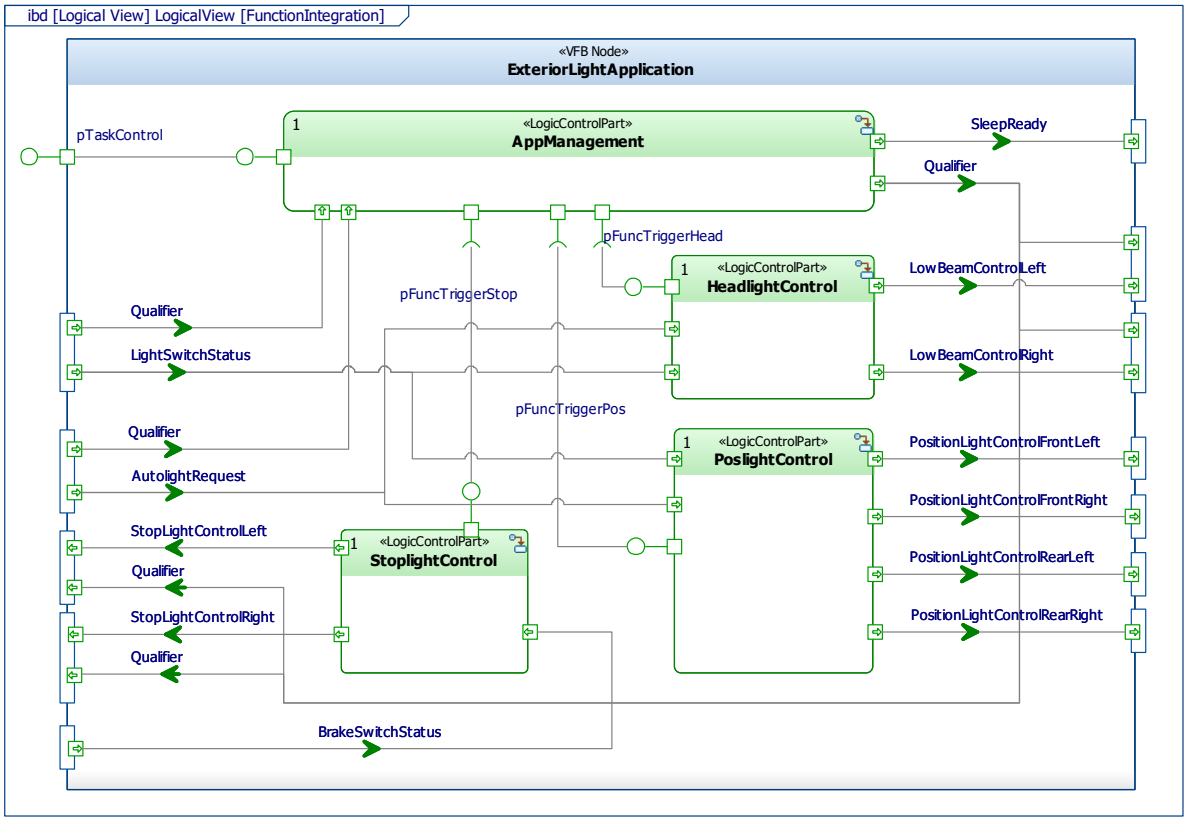


Abbildung 12: Integration der Funktionselemente in einen Virtual Function Bus Node

Funktionen in einer elektronischen Steuereinheit (ECU) müssen zusätzlich Implementierungs-Einschränkungen umsetzen, wenn sie für verschiedene Betriebssysteme und SW-Frameworks betrieben werden, wie z. B. (AUTOSAR Classic 2022). Dennoch sollten die Funktionen so gestaltet sein, dass sie auf verschiedene Zielsysteme angewendet werden können, um die Wiederverwendbarkeit zu maximieren. Die Anpassung erfolgt dementsprechend in den logischen Knoten durch logische Ports, die die Prinzipien vom Kapitel „Verknüpfung von Elementen zwischen den Modellsichten“ anwenden.

„Full-Ports“, typisiert auf *Interface Behavior Blocks*, ermöglichen es nun, funktionale Werte intern auf externe, standardisierte oder kundenspezifische Signal- und Service-Schnittstellendefinitionen und umgekehrt zu transformieren. Weiterhin können mit diesem Prinzip auch Simulink-Blöcke von Matlab Modellen (IBM Docu 8.3) oder Drittanbieter-Funktionsdesigns, die nicht direkt mit logischen Schnittstellen kompatibel sind, flexibel in logische Knoten integriert werden, indem die vorgestellte Verknüpfungsmethode angewendet wird. Logische Knoten und ihre Schnittstellen erscheinen von außen identisch, aber die innere Funktions-Implementierung kann unterschiedlich sein. Abbildung 12 zeigt beispielhaft, wie ein logischer Signal Port, der Informationen für den „Lichtschalterzustand“ und den „Qualifier“ enthält, funktional mit der „Scheinwerfersteuerung“ verknüpft ist, die wiederum mit dem logischen Ausgangsport durch die funktionalen Werte für „Abblendlichtsteuerung Links/Rechts“ gekoppelt ist.

Im Rahmen von ACMBSE werden Allokationen typischerweise vermieden, wenn Elemente von der funktionalen zur logischen Sicht verknüpft werden. Es werden, ähnlich wie bei der objektorientierten Programmierung, die funktionalen Blöcke in den logischen Knoten instanziiert und funktionale Schnittstellen mit Signal Ports verbunden, wobei die inneren funktionalen Ports die verschachtelten Schnittstellen zu den funktionalen Elementen freilegen, wie in Abbildung 12 zu sehen ist.

Der gezeigte Modellierungsansatz führt zu einem sehr konsistenten und durchgängig ausführbaren Architekturmodell, das für sicherheitskritische Anwendungen erforderlich ist, die die Einhaltung des ISO 26262 (ISO 1 2018) Standards verlangen. Wenn die Richtlinien des iSEF korrekt appliziert werden, ist die Verknüpfung von Elementen verschiedener Modellsichten dauerhaft auf Elemente derselben Abstraktionsebene beschränkt, um falsche und nicht implementierbare Integrationen aufgrund von „Level Hopping“ zu verhindern. Vorgegebene Dekompositionsmodelle und spezialisierte Stereotypen innerhalb des iSefML-Profils werden deshalb bereitgestellt, um weitere Echtzeitmodellprüfungen während des Entwurfs von Architekturelementen zu erleichtern.

9.7 SCHRITT 7 - TECHNICAL SYSTEM DEVELOPMENT

Mit der Integration von Komponenten aus den Software- und Hardware-Disziplinen wird die technische Modellsicht eines Systems of Interest (SoI) modelliert. Der Technical Viewpoint stellt dazu hauptsächlich die Interaktion zwischen den logischen Knoten des Systems (wie Anwendungen) und technischer Elemente wie Mikrocontrollern bereit.

Es ist entscheidend zu überprüfen, wie die logischen Knoten mit externen Elementen über die technischen Schnittstellen (Ports) der Verarbeitungseinheit (z. B. Treiberknoten) kommunizieren, wie in Abbildung 13 dargestellt.

Es ist noch zu beachten, dass jede technische Schnittstelle des Mikrocontrollers eine Hardware-Software-Schnittstelle (HSI) symbolisiert, die analogen oder digitalen Eingänge und sogar Datenschnittstellen für die Kommunikation über den Controller Area Network (CAN)-Bus umfasst. Anstatt direkt mit vorhandenen Kontrollern zu integrieren, konzentriert sich diese Modellsicht darauf, die technisch wesentlichen Schnittstellen für den Aufbau des Controllers zu identifizieren. Die tatsächliche Implementierung des vorhandenen Kontrollers wird in späteren Phasen auf der physischen Modellsicht durchgeführt.

Es ist bei der technischen Modellsicht zu beachten, dass die logischen Knoten (Softwarekomponenten), hier vereinfacht dargestellt, durch Komposition integriert werden, da der Fokus auf dem Zusammenspiel der Elemente liegt. Reell gesehen, muss der Prozess der Codegenerierung und Kompilierung durchlaufen werden. Diese Modellsichten werden dann aber durch die *Verteilungsdiagramme* mittels des UML-Profils dargestellt, wo der Fokus dann auf den Software-spezifischen technischen und physischen Artefakten liegt, die hier in der Systemdarstellung nicht relevant sind.

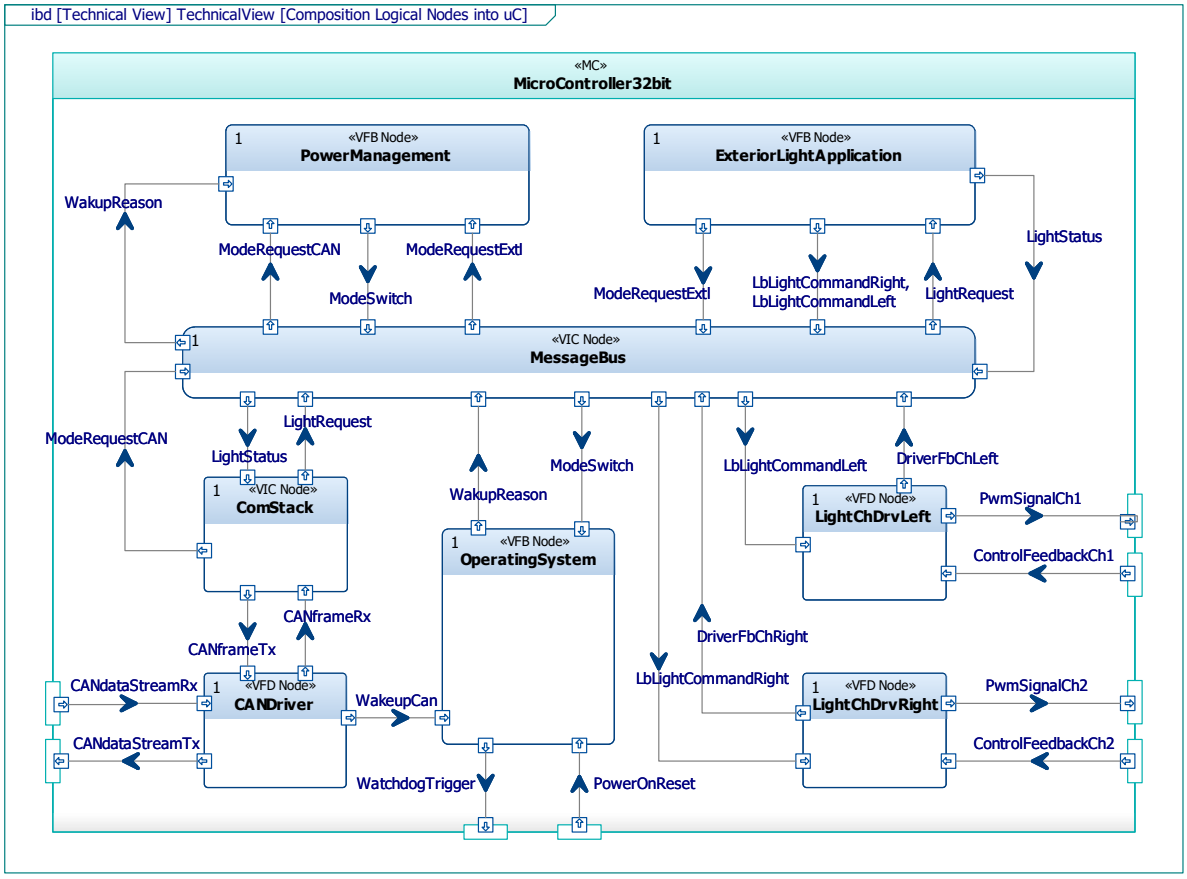


Abbildung 13: Implementation logischer Knoten in einen technischen Mikrocontroller

Das Modellieren der technischen Architektur entsprechend der iSEF-Methode, insbesondere mit den HSI-Elementen, ermöglicht eine ausführbare Simulation logischer Elemente innerhalb einer Verarbeitungseinheit, noch bevor die Software entworfen wird. Dies ermöglicht bereits der Überprüfung technischer Einschränkungen wie Zeitplanung, Ressourcen und Timing in der frühen Phase der Entwicklung. Abbildung 13 veranschaulicht, wie die „Exterior Light Application“ aus Abbildung 12 in die Architektur des Mikrocontroller eingebettet ist und mit anderen logischen Knoten wie dem „Energiemanagement“ oder dem „Kommunikationsstapel“ interagiert. Ein Nachrichtenbus wird zusätzlich integriert, um alle logischen Signale zu leiten. Dieses Verfahren ermöglicht eine Simulation hardwareabhängiger Sicherheitsmechanismen zwischen Hardware und Software zur Überprüfung der sicherheitskritischen Systementwicklungen bis zu ASIL D (ISO 1 2018).

Durch die Methoden des iSEF wird es möglich, Systemprototypen zu erstellen, noch bevor die eigentliche Hardware verfügbar ist, was die Entwicklungskosten erheblich reduzieren kann. Solche detaillierten Modellierungen stellen jedoch einige Herausforderungen an das Wissen des Architekten dar und erfordern einen disziplinierten Ansatz zur vollständigen Entwicklung aller Funktionsknoten und Schnittstellenelemente. Daher ist oft ein Kompromiss im Modellierungsumfang erforderlich, wobei sich auf die Teile des Systems konzentriert wird, die für das Verständnis und die Simulation essenziell sind, während andere Teile statisch und nicht vollständig ausführbar bleiben. Es ist also wichtig frühzeitig zu erkennen, dass für einen modellbasierten Modellieransatz auch an eine entsprechende Ausbildung des Systemingenieurs gedacht werden sollte.

9.8 SCHRITT 8 - PHYSISCHE REALISIERUNG DER ARCHITEKTUR

Die physische Modellierung wird schließlich verwendet, um System- und Hardware-Ingenieuren bei der Architektur von Endprodukten, hauptsächlich elektronischen Steuergeräten (ECUs), zu helfen. Physische Modellierung könnte jedoch auch angewendet werden, um Halbleiterchipsätze zu entwerfen, ihre Architektur zu dokumentieren oder auch gesamte Fahrzeug-E/E-Systeme zu modellieren. Das iSEF unterstützt dieses Ziel auf verschiedene Weisen. Es stellt Modellelemente bereit, um Mikrocontroller basierend auf vordefinierten Vorlagen und Portkonfigurationen zu entwickeln. Zudem richtet es das Device für die Software und das grundlegende System ein. Ferner hilft es, das Endbenutzersystem, also das Fahrzeug, präzise zu definieren. Zum Beispiel wird ein gesamtes Bordnetzwerk (Wire-Harness) mit den Modellen aller Steuergeräte zusammen integriert.

Es ist damit also möglich, hochkomplexe Systeme in einer Modellintegration durch einen OEM mit den zugelieferten Modellanteilen von Zulieferern bereits zusammenzuführen und zu verifizieren, ohne dass es nötig ist, einen einzigen Prototyp zu bauen. Das kann am Ende zu signifikanten Entwicklungskosteneinsparungen führen, da weniger Korrekturschleifen erfahrungsgemäß notwendig werden.

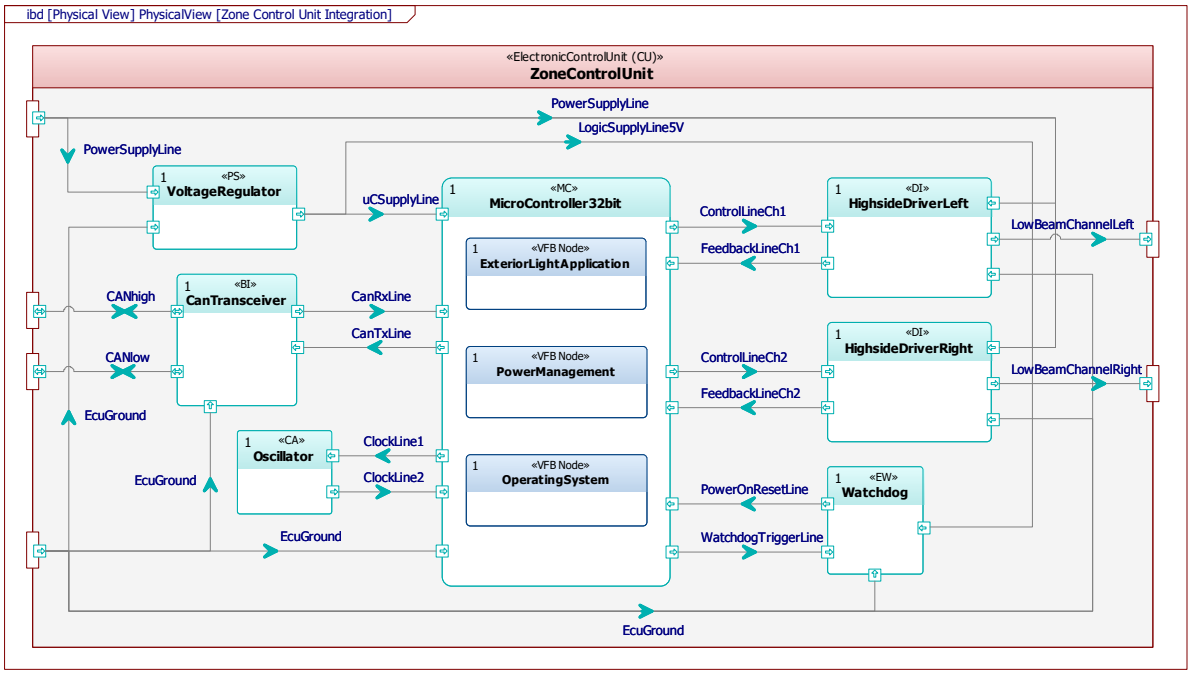


Abbildung 14: Komposition der technischen Elemente in das physische Steuergerät

Abbildung 14 illustriert einen Ansatz neben anderen Möglichkeiten zum Aufbau der HW-Architektur einer ECU basierend auf technischen Lösungen, die durch technische Elemente definiert sind, wie sie typischerweise von einem Systemingenieur festgelegt werden. In diesem Beispiel werden die zuvor entworfenen technischen Elemente (z. B. der technische Mikrocontroller aus Abbildung 13) in eine physisch zu entwickelnde ECU integriert und mit den erforderlichen Ports der ECU verbunden. Mit dieser Architektur kann der Systemingenieur alle technischen Lösungen und notwendige physikalische Einschränkungen an einen Hardware-Ingenieur weitergeben, der unter Beachtung dieser das physische System (ECU) entwirft. Das iSEF zielt darauf ab, in Zukunft eine große Menge vordefinierter physischer Standardkomponenten bereitzustellen, um diese Modellierungsansätze zu unterstützen und den Entwicklungsprozess zu beschleunigen.

Ein weiterer Ansatz, der in diesem Artikel nicht gezeigt wird, beinhaltet die Modellierung einer vollständigen physischen Architektur für das endgültige Produkt. Dabei kombiniert die ECU alle relevanten physischen Elemente zu einer vollständig optimierten Hardware-Architektur. Diese Optimierung erfolgt in Bezug auf Leistung und Kosten, basierend auf technischen Merkmalen und gegebenen technischen Einschränkungen. Das Ergebnis ist ein spezifisches Hardware-Design. Diese Art der Architektur, auch als physisches Block Diagramm bezeichnet, ermöglicht dann die Entwicklung von wiederverwendbaren, standardisierten physischen Teilkomponenten, welche noch einmal die Entwicklungszeit wesentlich verringern können, wenn Lösungen bekannt sind und aus Baukastenelementen architektonisch zusammengesetzt werden können. Gerade für große Plattformen im Automobilbau bietet sich dieser Ansatz an, um signifikant die Entwicklung zu rationalisieren. Aber dafür ist ein nicht zu unterschätzender Anfangsinvest notwendig, um ebendiese Baukastenelemente zu erstellen.

10 FAZIT UND AUSBLICK

In den kommenden Jahren wird es bedeutende Veränderungen in der Fahrzeugelektronik geben, die die Notwendigkeit für skalierbare, flexible Plattformen für die Systementwicklung, einschließlich des Übergangs zur nächsten Generation der E/E-Architektur (Seidel & Forlingieri 2023), intensivieren. Ein Systemarchitekturentwicklungsansatz, der über aktuelle Regeln und traditionelle dokumentenbasierte Methoden hinausgeht, ist erforderlich, um diesem neuen und herausfordernden Kontext gerecht zu werden. Der in diesem Artikel vorgestellte ACMBSE-Ansatz umgesetzt im invenio Systems Engineering Framework betont die zentrale Rolle der Systemarchitektur in der modellbasierten Entwicklung. Die Bedeutung des Aufbaus und der Implementierung dieses architektonischen Ansatzes wurde von dem Autor in diesem Artikel hervorgehoben, indem einige zentrale Leitprinzipien vorgestellt wurden:

(1) Die Entwicklung eines Modellierungsframeworks wie das iSEF ist für eine kohärente und effiziente Entwicklung über verschiedene Ingenieursabteilungen hinweg unerlässlich. Diese Praxis kann auch in anderen Branchen angewendet werden, wie bei Airbus, das MOFLT übernommen hat (Ducamp et al., 2022). Durch die Anwendung von ACMBSE Prinzipien können Herausforderungen wie besseres Architekturverständnis, Komponentendefinition und nachhaltige Kommunikation im Team und mit Stakeholdern überwunden werden.

(2) Der Artikel hebt die Bedeutung der Etablierung verschiedener, aber miteinander verbundener Modellsichten während der Systementwicklung hervor. Das Regelwerk des iSEF bietet dazu 7 grundsätzliche Modellsichten, ähnlich wie analoge Frameworks (Roques 2016; Ducamp et al., 2022), die ein Systemdesign aus unterschiedlichen Perspektiven ermöglichen, mit einer nahtlosen Verbindung von System- zur Softwarearchitektur. Im Automobilbereich erstreckt sich diese Methode von der Fahrzeugintegration bis zur ECU-Entwicklung und deckt alle wesentlichen architektonischen Aspekte ab, wobei sie als Bindeglied für die gesamte Produktbeschreibung dient.

(3) Der Autor hat eine der zahlreichen potenziellen Anwendungen des iSEF und seiner Standpunkte systematisch demonstriert. Ein grundlegendes Prinzip von MBSE zur Etablierung der Modellierung kann im iSEF durch die Anwendung der in diesem Artikel dargestellten Prinzipien der Verknüpfung von Ansichten erreicht werden. Obwohl nicht immer alle Viewpoints und System Abstraction Levels gleichzeitig erforderlich sind, können unterschiedliche Aspekte desselben Viewpoints während der Entwicklung komplexer Systeme wie eines Fahrzeugs genutzt werden.

Obwohl dieser Artikel sich hauptsächlich auf die Analyse- und Entwurfsphase der Systementwicklung mittels ACMBSE konzentriert hat, wurde in 8 einfachen Schritten die Modellierung einer Systemarchitektur illustriert. Ein kritischer Aspekt bleibt jedoch zu behandeln: die modellbasierte Verifikation und Validierung. Die Sicherstellung der Präzision und Kohärenz einer komplexen Systemarchitektur erfordert die Verifizierung der semantischen und syntaktischen Genauigkeit des Modells und die Konsistenz zwischen den interagierenden Modellkomponenten. Daher sollten zukünftige Werkzeuge technische Verfahren wie den Einsatz von Just-in-Time-Modellprüfungen beinhalten, die die im Framework zugrunde liegenden Methoden und Validierungsmechanismen umsetzen und für den Benutzer leicht anwendbar machen.

Ein weiterer wichtiger Punkt, den der Autor kurz angesprochen hat, ist die Modellierung der Variabilität funktionaler und technischer Features, die die gewünschten Eigenschaften in einer Produktlinienentwicklung von Anfang an berücksichtigen. Das Ziel des Artikels war jedoch nicht zu zeigen, wie diese Merkmale systematisch modelliert und genutzt werden können, um mehrere Produktvarianten zu entwickeln. Ein zukünftiger Artikel könnte, dem Beispiel von Forlingieri und Weilkiens (2022) folgend, zeigen, wie das Variantenmodellieren über die verschiedenen Viewpoints und Abstraction Levels mittels des iSEF realisiert werden kann.

Abschließend weist dieser Artikel auf den Architektur-fokussierten modellbasierten Aspekt der Systementwicklung hin, der ausschließlich innerhalb des Modellierungswerkzeugs dargestellt wird. Es ist jedoch notwendig, die Bedeutung der Nutzung des Systemmodells als Brücke zur Verbindung mit den anderen domänenspezifischen Disziplinen innerhalb des Ingenieurslebenszyklus zu betonen. MBSE ersetzt nicht das Anforderungsmanagement, Testmanagement, Softwareentwicklung oder die mechanische Disziplin. Vielmehr ermöglicht das iSEF die nahtlose Integration dieser Disziplinen durch ein Gesamtmodell für einen nahtlosen und kohärenten Systementwicklungsprozess. Das invenio System Engineering Framework (iSEF) hat die ersten Schritte unternommen, um dieses Ziel durch architekturzentriertes modellbasiertes Systems Engineering (ACMBSE) zu erreichen.

11 REFERENZEN

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 BIOGRAPHIE

Enrico Seidel ist Senior Consultant bei der invenio Systems Engineering GmbH und ist gegenwärtig verantwortlich für den Bereich Modellbasierte Entwicklung. Er leitet als technischer Architekt die Entwicklung des iSEF. Zuvor war er Senior Technical Expert für Continental Automotive Singapore im Bereich Systems Engineering (SE). Innerhalb des Geschäftsbereichs (BA) Architektur und Vernetzung war er verantwortlich für die Definition der SE-Arbeitsprinzipien in der Gruppe Engineering Excellence. Mit seiner über 17-jährigen Erfahrung im Systems Engineering entwickelte er gemeinsam mit dem Team das Capability Architecture Framework (CAF) und treibt nun mit dem Nachfolgeprodukt iSEF in der invenio AG die Ideen von MBSE weiter voran.