Informatik 11
Embedded Software

RWTH AACHEN UNIVERSITY

Bachelorarbeit

Bachelor Thesis

Entwicklung eines Frameworks für die automatisierte Architekturvalidierung

Development of a framework for automated architecture validation

Rushanth Rasaratnam
Matrikelnummer: 367117

Aachen, 21. Juli 2025

Gutachter:
Prof. Dr.-Ing. Stefan Kowalewski
Prof. Dr. rer. nat. Bernhard Rumpe

Betreuer:
David Klüner M.Sc.

Diese Arbeit wurde vorgelegt am
Lehrstuhl Informatik 11 – Embedded Software

This is to be done

Zusammenfassung

An abstract is basically a very short summary of the thesis topic. It describes short and precise the content of the thesis and what has been achieved. An abstract should not contain any discussion, quotation or reference to figures, chapters and so on. Please note that if you write your thesis in german, there should be a german and english version of the abstract. Although an abstract should generally not contain examples we provide here an example abstract for this document:

There exists a large variety of latex templates such as guidelines explaining or demonstrating the common structuring, dos and donts of bachelor and master theses. This document aims at providing an example for a typical structure of such theses containing useful knowledge about scientific writing and providing students with a correspondent latex template for bachelor or master theses at i11. Moreover, it gives some advice on very basic latex functionality for citation and figure handling.

Eidesstattliche Versicherung

Rasaratnam, Rushanth	367117
Name, Vorname	Matrikelnummer

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Bachelorarbeit mit dem Titel

Entwicklung eines Frameworks für die automatisierte Architekturvalidierung

selbständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Ort, Datum	Unterschrift

Belehrung:

§ 156 StGB: Falsche Versicherung an Eides Statt
Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt
(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.
(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:

Ort, Datum	Unterschrift

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1. Einleitung

An introduction motivates the topic of the thesis and explains the origin of the task. It starts often with a very general statement followed by explanations which difficulties (of course with relevance to the thesis topic) arise. Finally, the difficulties are narrowed down to the topic of the thesis. At this point the reader should understand the relevance of the problem being addressed by the thesis. An example for an introduction could look like the text following below.

Motivate the topic, what is this thesis about and why is that important?

At the end of their bachelor or master studies young students have to write a final thesis. For many of them this is their first written academic document of mentionable relevance. Thus, those students have no experience with academic writing and therefore require often some basic guidelines. Guidelines are usually provided directly by the thesis advisers up front and/or during thesis review. Providing such guidelines in a written and structured form to the students results hopefully in less review work for advisor and student, more consistent and clear theses and hence theses of better quality.

Dies ist eine Anmerkung was noch zu machen ist.

1.1. Aufgabenstellung

This document aims at providing a basic template for students writing their bachelor or master thesis at Lehrstuhl Informatik 11 (I11). It shall provide a basic overview on how such writings are structured and provide some useful hints.

1.2. Gliederung

The outline of a thesis should point out how the different chapters are related to each other and that the thesis is well structured. It should show that the different chapters form a single coherent document. An example how such an outline could look like for this document is given below.

Chapter 2 gives an introduction and overview on latex citations and figures to provide some basic knowledge how to handle those latex-constructs. The proximate Chap. 3 presents an overview of work related to this document. Chapter **??** introduces subsequently how citations have to be used in order to avoid wrong citation styles

or even plagiarism. It therefore requires information introduced before in chapter 2. Additionally, this chapter provides some hints dealing with figures in latex and their quality. The document is concluded by Chap. 6 which gives also an outlook how this document could be extended in the future.

2. Grundlagen

Dieses Kapitel führt in die für das Framework relevanten Konzepte und Technologien ein. Zunächst wird der Begriff der E/E-Architektur samt ihrer Evolution und ihres Aufbaus erläutert. Anschließend wird die Softwarearchitektur behandelt: Nach der Definition grundlegender Begriffe werden die Prinzipien der Serviceorientierten Architektur vorgestellt und darauf aufbauend die Rolle von Middleware in der Fahrzeugtechnik beleuchtet. Ein weiterer Abschnitt erläutert die Motivation, Ziele und Methoden zur Validierung von Architekturen. Den Abschluss bildet die Vorstellung des für die Implementierung verwendeten Webentwicklungs-Stacks.

2.1. Automobile E/E-Architektur

Für alle softwaregesteuerten Funktionen innerhalb eines Fahrzeuges bildet die elektrische/elektronische Architektur (E/E-Architektur) das physikalische und logische Fundament. Sie definiert die grundlegende Organisation von Hardwarekomponenten wie Electronic Control Unit (ECU), Sensoren, Energieversorgungssystemen und Kabelbäumen und steuert deren Kommunikation und Interaktion, um die erwarteten Funktionen zu realisieren [Jia19]. Diese Architektur hat im Laufe der Zeit mehrere Varianten durchlaufen und wird stetig weiterentwickelt. Im Folgenden werden die evolutionären Stufen der E/E-Architektur genauer betrachtet.

Evolution der Architekturentwicklung

Ursprünglich waren Fahrzeuge hauptsächlich mechanische Produkte, was sich jedoch mit der Einführung von Elektronik, wie Motorsteuergeräte und Airbags, änderte und die Ära der Elektrifizierung innerhalb der Automobilindustrie begann. Zunächst waren, aufgrund der geringen Anzahl, die Verbindungen zwischen den elektronischen Komponenten meist Punkt-zu-Punkt, sprich Komponenten wurden direkt miteinander verbunden. Dies führte jedoch durch die steigende Anzahl an ECUs, zu einem Anstieg der Komplexität und des Kabelbaums. Um die Kommunikationseffizienz zu verbessern und die Komplexität zu reduzieren, wurden Controller Area Network (CAN)-Busse eingeführt. In der Folge wurde auch an Technologien wie Local Interconnect Network (LIN) und FlexRay entwickelt und eingesetzt, um die verschiedenen Kommunikationsanforderungen zu erfüllen, sodass Komponenten innerhalb einer Domäne Informationen gemeinsam nutzen.

Das Konzept der Funktionsdomäne rückte, aufgrund der wachsenden Komplexität und Funktionen immer stärker in den Vordergrund. ECUs und Funktionen wurden in Domänen wie Fahrwerk, Karosserie und Infotainment unterteilt. Durch die Nachfrage nach domänenübergreifender Kommunikation wurde die Gateway-Funktion, welche zuvor von einzelnen ECUs übernommen wurde, zunehmend in einem dedizierten, zentralen Gateway gebündelt. Obwohl dieser Ansatz die Mainstream-Lösung war, führte die Zunahme der Funktionen und Rechenanforderungen zu einer starken Zunahme der ECUs. Dies hatte komplexe Kabelbäume mit erhöhtem Gewicht und Kosten zur Folge, außerdem konnte der Ausfall durch das ausfallen des zentralen Gateways zu einem Zusammenbruch des gesamten Netzwerks führen.

Um diesem Problem entgegenzuwirken, wurden Domain Control Units (DCUs) eingeführt, welche Funktionen einiger ECUs innerhalb einer jeweiligen Funktionsdomäne integrieren und somit die Anzahl der ECUs und die Belastung der Gateways reduzieren. Um die bestmögliche Kommunikation zu garantieren, werden DCUs über das Automotive Ethernet mit dem zentralen Gateway verbunden, welches eine höhere Bandbreite bietet.

Ein weiterer Ansatz, der neben der domänenbasierten Architektur existiert, ist die zonenbasierte Architektur. Um Komplexität und das Gewicht der Verkabelung zu reduzieren, werden Komponenten anstatt nach Funktionen, basierend auf ihrer physischen Position innerhalb des Fahrzeuges unterteilt. Anstatt wie bei DCUs die Funktionen einer Domäne zu übernehmen, übernehmen Zone Control Units (ZCUs) die Funktionen, die in den entsprechenden physischen Zonen stattfinden. Auch hier wird die Kommunikation über das Automotive Ethernet durchgeführt. Dieser Ansatz reduziert die Anzahl der ECUs und Kabelaufwand (Anzahl, Länge, Gewicht) und verringert so die Komplexität der E/E-Architektur deutlich

Erst die Evolution der E/E-Architektur ermöglicht moderne Softwarekonzepte. Das folgende Unterkapitel beschreibt daher die Softwarearchitekturen, die diese neue Hardware-Basis nutzt.

2.2. Softwarearchitekturen in Fahrzeugen

Die allgemeine Definition einer Softwarearchitektur ist nach Bass et al. [BCK21] eine systematische Gliederung eines Softwaresystems in einzelne Bausteine, die Festlegung ihrer Schnittstelle sowie die Prinzipien, nach denen diese Komponenten zusammenwirken und organisiert sind. Seitdem sich der Schwerpunkt von Fahrzeugen weg von mechanikzentrierten Systemen bewegt hat und hin zu intelligenten, softwaredefinierten Plattformen, gibt es innerhalb der Automobilindustrie einen großen Fortschritt bezüglich der Softwarefunktionalität. Um bei diesem Fortschritt den Überblick zu behalten, ist eine fundierte Softwarearchitektur unerlässlich.

Zu Beginn setzte man auf klassische, geschichtete Architekturen wie OSEK/VDX und AUTOSAR Classic, jedoch merkte man mit dem Aufkommen des Konzepts eines Software-Defined Vehicle (SDV) sowie der steigenden Software-Komplexität, dass die klassische Softwarearchitektur an ihre Grenzen stieß. Es gibt einige Gründe, wie die starre, monolithische Schichtenstruktur: Funktionen sind fest an Schichten gebunden, was Anpassungen einzelner Komponenten oder den Austausch von Modulen kaum umsetzbar machte.

In diesem Unterkapitel wird ein Einblick in die Service-orientierte Architekturen (SOAs) gegeben, die beschriebenen Schwächen der klassischen Softwarearchitektur überwinden. Anschließend wird die Middleware erläutert, die als technische Vermittlungsschicht dient.

2.2.1. Serviceorientierte Architektur

Die Notwendigkeit, Softwaremodule dynamisch bereitzustellen erfordert einen flexiblen Ansatz. Hier setzt die SOA, welche bereits in der Webentwicklung fester Bestandteil ist, an. Nach Rumez et al.[RGKS20] ein Architekturmuster, bei dem Anwendung als lose gekoppelte, wiederverwendbare Dienste organisiert werden. Jeder Dienst bietet eine klar definierte Funktionalität über gut beschriebene Schnittstellen und kann zur Laufzeit von beliebigen Clients zugegriffen werden.

Im Gegensatz zum klassischen signalorientierten Ansatz, bei dem die Konfiguration der Kommunikationspfade vollständig statisch zur Design-Zeit erfolgt, setzt der SOA-Ansatz auf eine dynamische Konfiguration während der Laufzeit. Durch das dynamische Aufrufen von Diensten zur Laufzeit erhält die Architektur mehr Flexibilität. So lassen sich Funktionen einfacher hinzufügen, entfernen oder bearbeiten, ohne dass eine komplette Neuzuweisung der Software erforderlich ist. Dank der Middleware-Schicht, auf die im nächsten Abschnitt näher eingegangen wird, sind die verschiedenen Anwendungen unabhängig von dem Steuergerät, auf dem ein Dienst läuft, sowie vom zugrunde liegenden Netzwerk aufgebaut. Diese Entkoppelung von der Plattform ermöglicht es der Software, wiederverwendbar und portierbar zu sein. Ein weiteres Merkmal des SOA-Ansatzes sind Kommunikationsmuster, die Daten nur bei Bedarf übertragen. Im Gegensatz dazu verschickt die klassische Softwarearchitektur in zyklischen Abständen Daten – selbst wenn kein Empfänger vorhanden ist. Dadurch kann die entstehende Netzwerklast auf das Nötigste reduziert werden, was die Effizienz steigert. Darüber hinaus kapselt jeder Dienst hinter einer Schnittstelle eine klar definierte Funktionalität. So lassen sich diese Dienste in unterschiedliche Kontexte und Projekten wiederverwenden, wodurch der Entwicklungs- und Testaufwand gesenkt wird. Ein weiterer Vorteil ist die Integration externer und On-Demand-Dienste: SOA unterstützen die dynamische Anbindung von Backend- und Cloud-Diensten, welche weitgehend transparent ist. Außerdem lassen sich SOA an wachsenden Anforderungen anpassen ohne dass das Gesamte System neu zu strukturieren. Aus den genannten

Gründen erweißt sich die SOA als flexiblere und anpassungsfähiger als die klassische Variante der Softwarearchitektur.

Um die Vorteile der SOA realisieren zu können benötigt es eine Zwischenschicht zwischen Anwendung und Netz. Im folgenden Abschnitt wird die Middleware-Schicht untersucht, welche die SOA-Prinzipen technisch umsetzt. Dabei wird zunächst auf Aufbau und Funktionen eingegangen. Anschließend wird die ASOA-Middleware genauer untersucht.

2.2.2. Automobile Middleware

Innerhalb moderner, verteilter Softwaresysteme ist die Middleware-Schicht eine wichtige Vermittlungsebene. Neely et al. [NDN06] definieren die Middleware als Schicht zwischen Anwendungssoftware und Systemsoftware. In Systemen mit Netzwerkanbindung vermittelt sie zusätzlich zwischen Anwendungssoftware, Betriebssystem und Netzwerkkommunikationsschichten. Dabei liegt ihre Hauptaufgabe darin, die Komplexität des zugrunde liegenden Systems zu abstrahieren [NDN06]. Im Bereich der Fahrzeugtechnik wird die Middleware noch einmal in zwei Sub-Schichten unterteilt [KMK+24]:

- Kommunikations-Middlewares (untere Schicht), die den Datenaustausch zwischen ECUs organisieren und Quality-of-Service (QoS)-Funktionen bereitstellen.
- Architekturplattformen (obere Schicht), die umfassende Frameworks für die Entwicklung und Bereitstellung automobiler Softwaresysteme liefern.

Nun wird untersucht, wie die Kommunikations-Middlewares und Architekturplattformen konkret arbeiten.

Als untere Sub-Schicht sorgt die Kommunikations-Middleware dafür, dass sämtliche Details des Datenaustauschs im In-Vehicle Network (IVN) abstrahiert und Anwendungen von der zugrunde liegenden Netzwerktopologie entkoppelt sind. Sie sorgt zudem dafür, dass strukturierte Daten wie z.B. Sensormesswerte automatisch seralisiert und über Ethernet, CAN-Gateway oder Shared-Memory-Kanäle verteilt werden. Dies wird durchgeführt, ohne dass Entwickler sich mit den Kommunikationsprotokollen befassen müssen. Außerdem ermöglicht sie die Definition und Einhaltung von QoS-Parametern. Des Weiteren sorgt ein intelligentes Routing- und Topologie-Management auf der Netzwerk-Ebene dafür, dass Nachrichten den bestmöglichen Pfad für die Übertragung wählen, während ein Real-Time-Scheduler die Übertragungen koordiniert.

Mit der Kommunikations-Middleware als Basis, löst die Architekturplattform Herausforderungen in Bezug auf die Entwicklung von Automobil-Softwaresystemen. Sie stellt eine umfassende Laufzeitumgebung bereit und bietet Entwicklern standardisierte Dienste und APIs. Das Ziel ist es die bestehende Komplexität weiter zu verringern, sodass Entwickler den Fokus auf die Anwendungslogik setzen können, anstatt sich mit

systemnahen Aufgaben zu beschäftigen. Abgesehen von der Kommunikation, gehört zunächst das umsetzen der Software-Designmustern, wie das zuvor angesprochene SOA-Prinzip, zu den Kernaufgaben. Ebenso Ressourcenmanagement und Orchestrierung , bei der die Ressourcen von inaktiven Diensten freigegeben werden. Das Ziel hierbei ist eine situationsabhängige optimale Ressourcenzuweisung. Als nächstes ist die deterministische Ausführung sowie die Unterstützung bei der Einhaltung von Echtzeitanforderung zu nennen, um Verzögerungen im Softwaresystem zu vermeiden. Des Weiteren bietet sie oft Funktionen zur Cybersicherheit, wie Authentifizierung, Zugriffskontrolle, Verschlüsserung etc. an um Schwachstellen innerhalb des IVNs zu reduzieren. Darüber hinaus sorgt die obere Schicht dafür, dass Over-the-Air-Updates, Installation/Deinstallation von Software-Paketen, sowie die Konfiguration von Anwendungen während der laufzeit durchgeführt werden und bietet dementsprechend Funktionen an. Eine weitere wichtige Aufgabe ist die Verwaltung des Systemzustands und die Überwachung des Lebenszyklus von Softwarekomponenten. Nicht zuletzt bietet die Architekturplattform Entwicklungswerkzeuge an, welche für die Entwicklung und Bereitstellung verwendet werden.

2.3. Validierung von Architekturen

Wie bereits zu Beginn der Arbeit dargelegt, ist die frühzeitige Prüfung von Architekturen von entscheidender Bedeutung. Das technische Ziel der Architekturvalidierung ist daher, die Korrektheit und Konsistenz eines Systementwurfs anhand definierter Qualitätsmethoden, wie ISO 26262 für funktionale Sicherheit oder ISO 25010 was sind die Zielefür Softwarequalität, zu bewerten, noch bevor die Implementierung beginnt [VRST15]. Dieses Ziel lässt sich in die folgenden Teilziele unterteilen: Früherkennung von Fehlern, Kostenreduzierung, Sicherstellung von Qualität und Standardkonformität, Verifikation des Systemverhaltens und der Anforderungen, Beherrschung der Systemkomplexität und Verbesserung des Verständnisses[VRST15] [Kan19] [BKB19]. Im folgenden werden die unterschiedlichen Ansätze zum validieren der Architekturen genauer betrachtet.

Validierungsverfahren

Es existieren veschiedene Ansätze zur Validierung eingesetzt, um die Gültigkeit von automobilen Systemen zu gewährleisten. Diese lassen sich in folgende Kategorien unterteilen[KPKS10] [VRST15] [WPK+16]:

2.4. Webentwicklungs-Stack

3. Stand der Technik

In diesem Kapitel werden die bestehenden Werkzeuge zum Spezifizieren von Architekturen im Bereich der Fahrzeugindustrie sowie die von ihnen verwendeten Verfahren zur Architekturvalidierung. Dies ermöglicht ein besseres Verständnis für den Beitrag, den die Abschlussarbeit leistet.

3.1. Etablierte Werkzeuge und Plattformen

Um die zunehmende Komplexität der E/E-Architekturen in modernen Fahrzeugen mit Blick auf autonome Fahrfunktionen und Fahrassistenzsystemen zu bewältigen, wird aktiv an Software gearbeitet, die es Entwicklern und Ingenieuren ermöglicht, funktionale, softwarebezogene und E/E-Architekturen zu modellieren bzw. zu spezifizieren [AHFK22] [Sch16]. Im folgenden werden die Werkzeuge PREEvision und Simulink + System Composer, welche als Industriestandard gelten, und Capella als kostenlose Alternative analysiert.

3.1.1. Vector PREEvision

PREEvision ist ein kommerzielles, modellbasiertes Werkzeug zur Entwicklung und Optimierung verteilter, eingebetteter Systeme in der Automobilindustrie. Das Hauptziel des Werkzeugs ist es, die zunehmende Komplexität der E/E-Architekturen innerhalb der SDVs beherrschbar zu halten [AHFK22]. Das Werkzeug richtet sich dabei an die weit akzeptierten Automobilstandards wie AUTomotive Open System ARchitecture (AUTOSAR) RIF/ReqIF[1], KBL[2] und VEC[3] [Sch16]. Es werden außerdem die drei System-Engineering-Prinzipien unterstützt: Abstraktion (Implementationsaspekte auf eine konzeptionellere Ebene abstrahieren), Dekomposition (System kann in jeder Schicht hierarchisch zerlegt werden), Wiederverwendung (Komponenten und Modelle können von verschiedensten Produktlinien und Varianten verwendet werden) [Sch16].

Der Schwerpunkt von PREEvision ist jedoch, die auf Abbildung 3.1 grob dargestellte Schichtenarchitektur.Die Abbildung zeigt den Weg von der abstrakten Anforderungsebene zur konkreten Implementierung. Ausgehend von den Anforderungen wird eine Logische Funktionsarchitektur entworfen, welche dann in eine SOA und

[1] Requirements Interchange Format: www.automotive-his.de/rif
[2] Kabelbaumliste: www.vda.de
[3] Vehicle Electric Container: www.vda.de

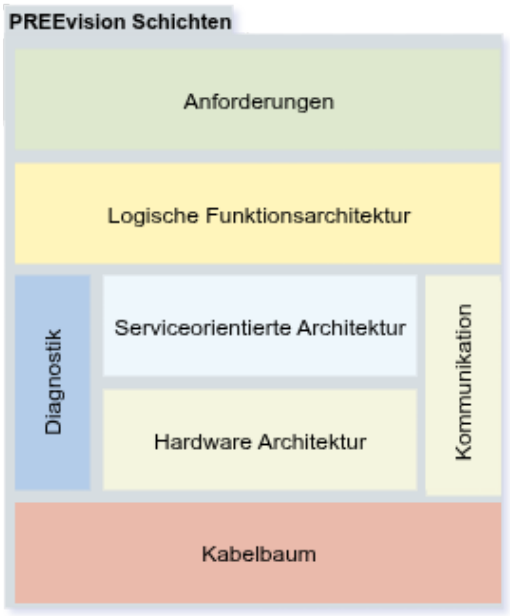

Abbildung 3.1.: Schematische PREEvision Schichtenarchitektur. (Nachzeichnung von [Vec])

eine Hardware Architektur überführt wird, bis hin zum physischen Kabelbaum. Dieser durchgängige Ansatz stellt sicher, dass alle Aspekte des Systems, inklusive Diagnostik und Kommunikation, miteinander verbunden sind (Single-Source-of-Truth) [Sch16]. Eine detaillierte Abbildung der Schichtenarchitektur ist im Appendix A.1 zu finden.

3.1.2. Mathworks Simulink + System Composer

Simulink und System Composer sind kommerzielle Werkzeuge, die eng miteinander integriert sind, um modellbasierte Systementwicklung (MBSE) zu unterstützen [WVK+20]. Während System Composer für die Modellierung und die Analyse der statischen System- und Softwarearchitektur verwendet wird, liegt der Fokus von Simulink in der Simulation des dynamischen Verhaltens dieser Architekturen. Dieses Zusammenspiel ermöglicht einen Übergang von dem reinen Architekturentwurf zu dem konkreten, ausführbaren Designmodell [Cha20].

Wie diese enge Integration praktisch umgesetzt wird, ist in Abbildung 3.2 zu erkennen: Eine in System Composer entworfene Architekturkomponente wird direkt mit einem detaillierten Verhaltensmodell in Simulink verknüpft. Der diesem Vorgehen zugrundeliegende Arbeitsablauf folgt dem V-Modell, bei dem das Systemverhalten phasenweise modelliert und validiert wird [The25]. Um komplexe und Verhaltens-

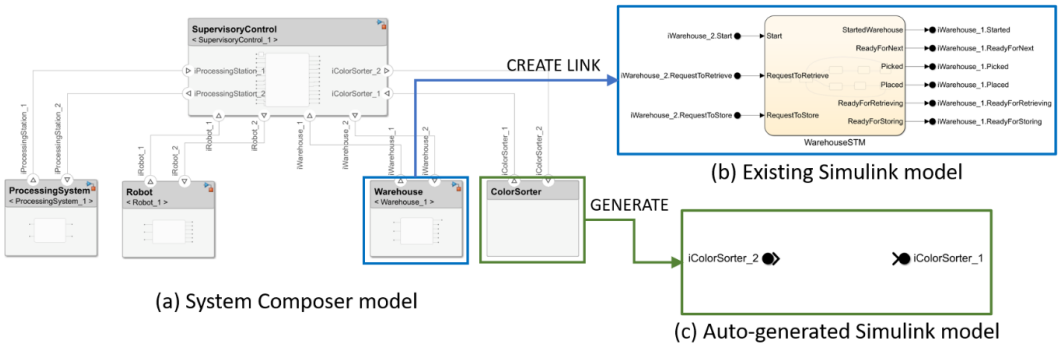

Abbildung 3.2.: Verknüpfung einer System Composer Architektur mit einem Simulink Verhaltensmodell. (Entnommen aus [Cha20])

weisen modellieren zu können, werden spezielle Werkzeuge innerhalb von Simulink genutzt [Cha20]:

Stateflow dient zur Modellierung der Logik von Systemen mittels Zustandsautomaten, Flussdiagrammen oder Wahrheitstafeln. Es wird insbesondere für die übergeordnete Steuerung (Supervisory Control) und dem Fehlermanagement genutzt [Cha20]. Die Fähigkeit, ereignisdiskrete Systeme zu modellieren/ analysieren, wird durch SimEvents ermöglicht. Mithilfe von Nachrichten- und Entitätenbasierten Konzepten können Leistungsmerkmale wie der Durchsatz oder die Latenz optimiert werden [Cha20].

Wie auch bei PREEvision, unterstützt dieser gesamte Ansatz die zentralen System-Engineering Prinzipien: Abstraktion, Dekomposition und Wiederverwendung[The25].

3.1.3. Eclipse Capella

Capella ist ein nicht-kommerzielles Modellierungswerkzeug, das auf der Engineering-Methode ARCADIA[4] basiert [Roq16]. Diese umfassende Methodik gibt einen klaren, schrittweisen Konstruktionsprozess vor, der den Anwender von der Bedarfsanalyse bis zum detaillierten Architektur-Design führt [let].

Die Abbildung 3.3 zeigt die vier Konstruktionsphasen von Arcadia. Der Prozess beginnt mit der Betriebsanalyse, die analysiert, was die Erwartungen, Anwendungsbedingungen, Integrations-, Verifikations- und Validierungsvorraussetzungen des Endnutzer sind. Darauf aufbauend folgt die Funktionale & Nicht-Funktionale Bedarfsanalyse, in der die Systemfunktionen und nicht-funktionalen Anforderungen abgeleitet werden.

Bei Arcadia haben Bedarfsanalyse und Modellierung, Architekturentwicklung und -validierung sowie Anforderungsanalyse die gleiche Wichtigkeit. Wobei die

[4] ARChitecture Analysis and Design Integrated Approach

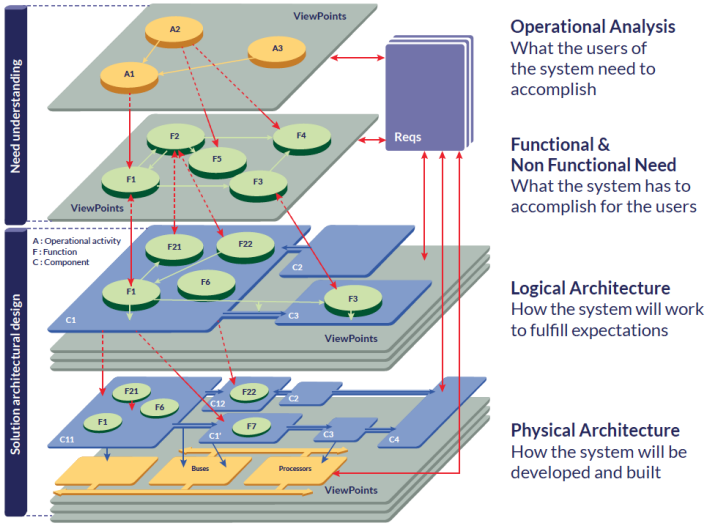

Abbildung 3.3.: Das Arcadia-Vorgehensmodell in Capella als vier übereinander gestapelte Architekturebenen (Entnommen aus [let])

Anforderungen gegen eine frühe Version des Architekturmodells auf Robustheit und Machbarkeit geprüft werden.

Nachdem System, Hardware und Software modelliert wurden, wird eine logische Architektur entwickelt. Dabei wird nach einem Kompromiss zwischen Entwurfsfaktoren, (nicht-funktionalen) Bedingungen und Ansichten gesucht. Jede Ansicht beschäftigt sich mit bestimmten Aspekten wie funktionale Konsistenz, Schnittstellen, Leistung, Echtzeit, Sicherheit, Integration, Wiederverwendung, Kosten und Risiko.

Abschließend beschreibt die physische Architektur die Umsetzung auf konkrete Hardware- und Softwarekomponenten und sichert durch die Trennung der Aspekte, Schichtenarchitekturen und standardisierte Interaktionsmuster eine effiziente, sichere Entwicklung und IVVQ[5] [let].

Während Arcadia die komplette linke Seite des V-Modells abdeckt, deckt das Produktlebenszyklusmanagement (PLM) die rechte Seite des Modells ab und garantiert somit eine vollständige Umsetzung des V-Modells [20224].

3.2. Validierungsverfahren im Detail

Ein zentraler Aspekt moderner Architekturwerkzeuge ist die Fähigkeit, entworfene Modelle zu validieren und deren Qualität sicherzustellen. Die in dieser Arbeit analy-

[5] Integration, Verifizierung & Validierung, Qualifizierungsaktivitäten (Testing)

sierten Werkzeuge verfolgen dabei unterschiedliche Ansätze und decken verschiedene Bereiche der Architekturvalidierung ab, die in Tabelle 3.2 vergleichend dargestellt werden.

Validierungsart	**PREEvision**	**Capella**	**Simulink**
Modell-Integrität & Konsistenz			
Inkonsistenz/Vollständigkeit	✓	✓	✓
Nachverfolgbarkeit	✓	✓	✓
Ebenenübergreifende Realisierung	✓	✓	
Anforderungs- & Design-Validierung			
Validierung gegen Anforderungen	✓	✓	✓
Analyse von Designalternativen	✓	✓	✓
Quantitative Analysen & Metriken			
Performance-Analyse	✓	✓	✓
Netzwerk- und Ressourcen-Metriken	✓		
Statische Analyse		✓	✓
Sicherheits- & Zuverlässigkeitsanalysen			
Sicherheitsanalysen	✓	✓	
Fehlerfortpflanzungsanalyse		✓	
Verhaltensbasierte Validierung			
Dynamische Simulation			✓
Versions-Validierung			
Prüfung von Modellversionen	✓	✓	✓

Tabelle 3.2.: Auflistung der Architekturvalidierungen nach Werkzeug

Eine kurze Analyse der ausgewählten Werkzeuge zeigt, dass jedes von ihnen seine spezifischen Stärken in unterschiedlichen Bereichen der Architekturvalidierung besitzt. Während PREEvision bei E/E-spezifischen Metriken wie Netzwerk- und Ressourcenmetriken sowie Sicherheitsanalysen dominiert [Sch16], liegen die Stärken von Capella in der Sicherstellung der Architekturkonsistenz über mehrere Ebenen hinweg [Roq16]. Als einziges Werkzeug bietet Simulink zudem die Möglichkeit zur dynamischen Validierung des Systemverhaltens mittels ausführbarer Simulation [The25].

In der Praxis ergibt sich durch diese hohe Spezialisierung eine zentrale Herausforderung: Keines der Werkzeuge deckt alle notwendigen Validierungsverfahren ab. Darüber hinaus sind diese Systeme meist geschlossene Plattformen. Die Anbindung neuer oder unternehmensspezifischer Validierungsalgorithmen, die vom Hersteller nicht vorgesehen sind, ist häufig gar nicht oder nur mit erheblichem Aufwand möglich. Es fehlt an flexiblen und offenen Schnittstellen zur Integration externer Algorithmen.

Genau an dieser Stelle setzt die Bachelorarbeit an. Ziel dieser Arbeit ist es, das generische, webbasierte *ArchitekturTool* um ein Framework zur automatisierten Architekturvalidierung zu erweitern. Durch die Implementierung einer offenen API-Schnittstelle soll die bestehende Lücke hinsichtlich mangelnder Flexibilität geschlossen und die einfache Anbindung beliebiger Validierungsalgorithmen ermöglicht werden.

4. Konzeption & Design

Nachdem in den vorherigen Kapiteln die Grundlagen gelegt und die Lücken der bestehenden Systeme analysiert wurden, wird nun der Entwurf der Lösung thematisiert. In diesem Kapitel wird der Entwurf des Frameworks zur automatisierten Architekturvalidierung von Anforderung bis zum detaillierten Design nachvollziehbar dargelegt. Im ersten Abschnitt werden daher die Spezifikationen der funktionalen und nicht-funktionalen Anforderungen präsentiert. Auf dieser Basis wird anschließend der Architekturentwurf vorgestellt, welcher die Einbettung in das übergeordnete System erläutert. Die nachfolgenden Abschnitte fokussieren diesen Entwurf durch die detaillierte Konzeption der Programmierschnittstelle sowie dem Design der grafischen Benutzeroberfläche.

4.1. Anforderungsanalyse

Eine präzise Definition der Anforderungen ist die zwingende Voraussetzung für eine zielgerichtete Entwicklung. Basierend auf der Analyse der bestehenden Systemlücke werden daher die funktionalen und nicht-funktionalen Anforderungen an das zu entwickelnde Framework in Tabelle 4.2 spezifiziert. Diese bilden die verbindliche Grundlage für alle nachfolgenden Design- und Implementierungsentscheidungen.

Im Folgenden wird die Notwendigkeit für jede dieser Anforderungen näher erläutert.

Algorithmen-Application Programming Interface (API)

Ein zentrales Element des Frameworks ist die Bereitstellung der API, um die Validierungsalgorithmen an das ArchitekturTool binden zu können (FA-1). Diese Funktion ist die Grundvoraussetzung, um das System, wie im Ziel der Arbeit gefordert, erweiterbar zu gestalten. Um die Integration weiterer bzw. Anpassung bestehender Funktionalitäten für Entwickler zu vereinfachen und eine hohe Kompatibilität zu gewährleisten, wird zudem die Einhaltung des REST-Standards gefordert (NFA-1). Dieser etablierte Standard fördert eine lose Kopplung zwischen Framework und den angebundenen Algorithmen und ermöglicht eine schnelle Einarbeitung für Entwickler, die nicht vertraut sind mit dem Programm.

ID	Anforderung	Beschreibung
		Funktional
FA-1	Algorithmen-API	Bereitstellung einer API, die es Nutzern ermöglicht, Validierungsalgorithmen zu verwalten (hochladen, bearbeiten, löschen) und auszuführen.
FA-2	Algorithmenverwaltung-GUI	Bereitstellung einer GUI zur Verwaltung von Validierungsalgorithmen. Diese muss Funktionen zum Hochladen, Bearbeiten, Löschen sowie zum Starten der Ausführung umfassen.
FA-3	Ergebniss-GUI	Bereitstellung einer GUI zur visuellen Darstellung der Ergebnisse von Validierungsläufen. Die GUI muss verschiedene Ausgabeformate (z. B. textuell, grafisch) unterstützen.
FA-4	Erster Validierungsalgorithmus	Bereitstellung eines ersten Validierungsalgorithmus, der eine grundlegende Architekturprüfung durchführt.
		Nicht-Funktional
NFA-1	Standardisierte API	Die zu entwickelnde API muss den Prinzipien des RESTfolgen.
NFA-2	Benutzerfreundlichkeit	Die GUI muss intuitiv bedienbar sein.

Tabelle 4.2.: Anforderungen an das Framework

Grafische Benutzeroberfläche

Für die Interaktion mit dem Nutzer ist eine grafische Benutzeroberfläche zur Verwaltung der Algorithmen essenziell (FA-2). Sie muss alle notwendigen Funktionen wie das Hochladen und Ausführen der Validierungsalgorithmen ermöglichen. Um einen reibungslosen Ablauf beim benutzen des Tools zu gewährleisten, muss die Bedienung intuitiv und selbterklärend sein (NFA-2). Die Benutzerfreundlichkeit ist entscheidend für die Ingenieure und Entwickler im Arbteitsalltag. Des Weiteren müssen die Ergebnisse der Architekturvaliderungen visuell aufbereitet und ausgegeben werden (FA-3). Ohne eine Visualisierung der Ergebnisse ist es für den Nutzer schwer ersichtlich, was genau die Validierung ergeben hat. Die Unterstützung verschiedener Visualisierungsarten, wie textueller oder grafischer Ausgaben, ermöglichen es dem Nutzer, die Fehlerursache bzw. den Engpass effizient zu identifizieren.

Erster Validierungsalgorithmus

Das Anbinden eines ersten Validierungsalgorithmus (FA-4) ist aus mehreren Gründen wichtig. Zum einen lässt sich damit das Framework auf Korrektheit prüfen, sprich

es dient als Proof of Concept. Mit einem Validierungsalgorithmus lässt sich zum einen prüfen, ob das Hochladen, Bearbeiten und Ausführen der API wie gefordert funktioniert. Zum anderen können zukünftige Ingenieure und Entwickler den Algorithmus als Referenz nutzen um weitere Algorithmen anzubinden.Schließlich ermöglicht dieser erste Algorithmus die Durchführung von aussagekräftigen Tests und bildet die Grundlage für die Evaluation dieser Arbeit.

4.2. Systemarchitektur

Bei dem bestehenden ArchitekturTool handelt es sich um eine dockerisierte Webanwendung, deren Architektur auf einem React-Frontend, einem Node.js-Backend mit dem Express-Framework und einer PostgreSQL-Datenbank mit Prisma-ORM basiert. Um die geforderte Architekturvalidierung zu realisieren, wird diese Architektur, wie in Abbildung 4.1 dargestellt, um eine dedizierte Validierungs-API sowie um zugehörige Frontend-Module erweitert.

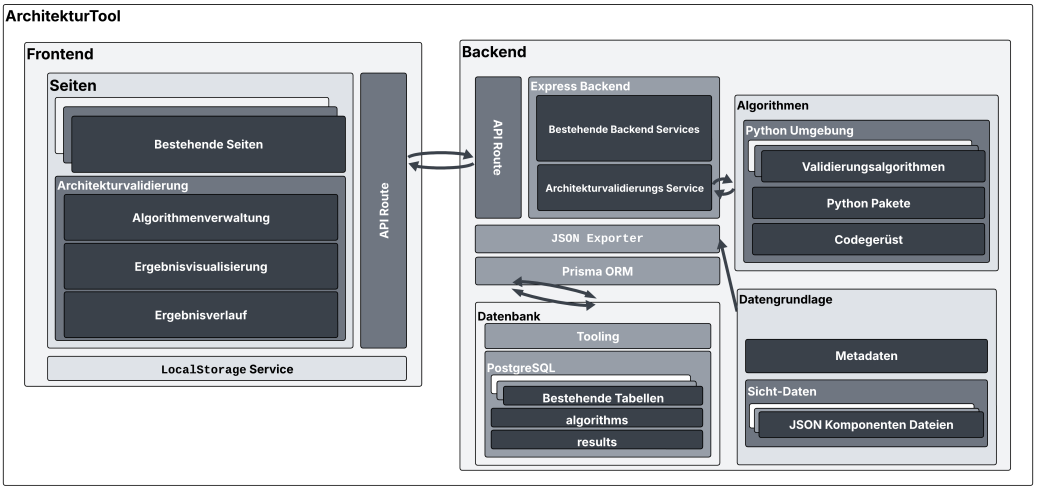

Abbildung 4.1.: Blockdiagramm des ArchitekturTool's

Wie im Blockdiagramm dargestellt, lässt sich die Erweiterung in vier Hauptbereiche unterteilen:

Algorithmenverwaltung (Frontend)

Diese Komponente bildet die zentrale Verwaltungseinheit für den Nutzer. Es stellt eine Benutzeroberfläche zur Verfügung, um Validierungsalgorithmen zu verwalten (hochladen, bearbeiten, löschen). Darüber hinaus ermöglicht es dem Nutzer, einen

hochgeladenen Algorithmus und eine im ArchitekturTool spezifizierte Sicht auszuwählen und einen Validierungslauf zu starten.

Ergebnisvisualisierung (Frontend)

Diese Komponente ist ausschließlich für die Darstellung der Validierungsergebnisse zuständig. Es bereitet die Ergebnisse eines Validierungslauf auf und stellt sie dem Nutzer, je nach Definierung, in textueller, beispielsweise als Fehlerlist, oder als grafischer Visualisierung, zum Beispiel als Balkendiagramm, vor.

Validierungsalgorithmus-API (Backend)

Validierung-Ausführen (Backend)

4.3. Konzeption der Komponenten

4.3.1. Programmierschnittstelle

4.3.2. Grafische Benutzeroberfläche

4.3.3. Validierungsalgorithmus

5. Evaluation

Since there is no evaluation for this document, this chapter lists some points which may be considered when evaluating software/algorithms/procedures.

- Comparison with other approaches to achieve the same goal as this work
- Measure resources needed by a developed tool/procedure/etc. regarding (time consumption, cpu usage, memory usage, ...)
- What about the quality of the results?
- Are there any problems with the procedure/applicability or restrictions?
- Evaluate (if possible) using „real world examples“ and not only academic ones.

Please note that they do not all have to apply or be suitable for any thesis topic. It might be that evaluation results are not presentable in a structured and readable way. For instance, if tables become to large. In such cases it might be helpful to place the results in the appendix **??** and refer to them. This applies also for figures, test specifications and tables at any other place in the thesis and is the only exception allowing forward references. The evaluation should state very clear what is possible and what is not. If there are any, the limits of the developed approach should be shown using suitable examples. Do not be afraid to show limitations of approaches. This does generally not undermine the authors achievement if it is clear and can be explained why the limitations exist. The author should avoid to use vague words like better, worse, etc. to describe the evaluation results. This applies in general for the other chapters of the thesis, too.

The next chapter of this document is the final chapter, which should always conclude the work presented before.

6. Conclusion

The aim of this document was to provide students at the end of their studies with a template for their written thesis. Due to the common lack of experience in the field of academic writing this work is intended to provide a template for structured writing. Therefore, this entire document is structured as a thesis should usually be.

Moreover it provides some hints how citations should be used and how figures should be dealt with to achieve high quality versions of latex documents. In contrary to this sentence, a conclusion should not introduce new information, about the topics discussed before, which has not yet been presented.

The following (optional) section provides some further ideas for potential extension of this work.

6.1. Future Work

There are many possible ways how this short document could be extended in the future. One may think of additional explanations regarding latex and its use, with the extend to an entire latex tutorial. A further extension could be the definition of helpful latex commands or a documentation on commonly used latex commands and packages.

Literaturverzeichnis

[20224] Arcadia (Ingenieurwesen). In: Wikipedia (2024), Dezember

[AHFK22] Askaripoor, Hadi ; Hashemi Farzaneh, Morteza ; Knoll, Alois: E/E Architecture Synthesis: Challenges and Technologies. In: Electronics 11 (2022), Januar, Nr. 4, S. 518. http://dx.doi.org/10.3390/electronics11040518. – DOI 10.3390/electronics11040518. – ISSN 2079–9292

[BCK21] Bass, Len ; Clements, Paul ; Kazman, Rick: Software Architecture in Practice, 4th Edition. 1st edition. Erscheinungsort nicht ermittelbar : Addison-Wesley Professional, 2021. – ISBN 978–0–13–688597–9

[BKB19] Bucher, Harald ; Kamm, Simon ; Becker, Jürgen: Cross-Layer Behavioral Modeling and Simulation of E/E-Architectures Using PREEvision and Ptolemy II. In: SNE Simulation Notes Europe 29 (2019), Juni, Nr. 2, S. 73–78. http://dx.doi.org/10.11128/sne.29.tn.10472. – DOI 10.11128/sne.29.tn.10472. – ISSN 23059974, 23060271

[Cha20] Chatterjee, Joyeeta: Applications of MathWorks Tools in Model-Based Systems Engineering. Eindhoven, Eindhoven University of Technology, Diplomarbeit, November 2020

[Jia19] Jiang, Shugang: Vehicle E/E Architecture and Its Adaptation to New Technical Trends. In: WCX SAE World Congress Experience, 2019, S. 2019–01–0862

[Kan19] Kang, Eun-Young: A Formal Verification Technique for Architecture-based Embedded Systems in EAST-ADL

[KMK+24] Klüner, David P. ; Molz, Marius ; Kampmann, Alexandru ; Kowalewski, Stefan ; Alrifaee, Bassam: Modern Middlewares for Automated Vehicles: A Tutorial. https://arxiv.org/abs/2412.07817v1, Dezember 2024

[KPKS10] Kollmann, Steffen ; Pollex, Victor ; Kempf, Kilian ; Slomka, Frank: Comparative Application of Real-Time Verification Methods to an Automotive Architecture. (2010)

[let] Let Yourself Be Guided with Arcadia: A Comprehensive Methodological and Tool-Supported Model-Based Engineering Guidance. https://mbse-capella.org/arcadia.html,

[NDN06] Neely, Steve ; Dobson, Simon ; Nixon, Paddy: Adaptive Middleware for Autonomic Systems. In: annals of telecommunications - annales des télécommunications 61 (2006), Oktober, Nr. 9-10, S. 1099–1118. http://dx.doi.org/10.1007/BF03219883. – DOI 10.1007/BF03219883. – ISSN 0003–4347, 1958–9395

[RGKS20] Rumez, Marcel ; Grimm, Daniel ; Kriesten, Reiner ; Sax, Eric: An Overview of Automotive Service-Oriented Architectures and Implications for Security Countermeasures. In: IEEE Access 8 (2020), S. 221852–221870. http://dx.doi.org/10.1109/ACCESS.2020.3043070. – DOI 10.1109/ACCESS.2020.3043070. – ISSN 2169–3536

[Roq16] Roques, Pascal: MBSE with the ARCADIA Method and the Capella Tool. In: 8th European Congress on Embedded Real Time Software and Systems (ERTS 2016). Toulouse, France, Januar 2016

[Sch16] Schäuffele, Jörg: E/E Architectural Design and Optimization Using PREEvision. In: SAE 2016 World Congress and Exhibition, 2016, S. 2016–01–0016

[The25] The MathWorks, Inc.: System Composer™ Getting Started Guide (R2025a) / The MathWorks, Inc. 2025. – Forschungsbericht. – 90 S.

[Vec] Vector Informatik GmbH: PREEvision – Model-Based E/E Development from Architecture Design to Series Production (Fact Sheet, Version 1.1) / Vector Informatik GmbH. – Forschungsbericht. – 2 S.

[VRST15] Venkitachalam, Hariharan ; Richenhagen, Johannes ; Schlosser, Axel ; Tasky, Thomas: Metrics for Verification and Validation of Architecture in Powertrain Software Development. In: Proceedings of the First International Workshop on Automotive Software Architecture. New York, NY, USA : Association for Computing Machinery, Mai 2015 (WASA '15). – ISBN 978–1–4503–3444–0, S. 27–33

[WPK+16] Weissnegger, Ralph ; Pistauer, Markus ; Kreiner, Christian ; Schuss, Markus ; Römer, Kay ; Steger, Christian: Automatic Testbench Generation for Simulation-based Verification of Safety-critical Systems in UML:. In: Proceedings of the 6th International Joint Conference on Pervasive and Embedded Computing and Communication Systems.

Lisbon, Portugal : SCITEPRESS - Science and Technology Publications, 2016. – ISBN 978–989–758–195–3, S. 70–75

[WVK+20] Watkins, Christopher B. ; Varghese, Jerry ; Knight, Michael ; Petteys, Becky ; Ross, Jordan: System Architecture Modeling for Electronic Systems Using MathWorks System Composer and Simulink. In: 2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC), 2020. – ISSN 2155–7209, S. 1–10

A. Appendix1

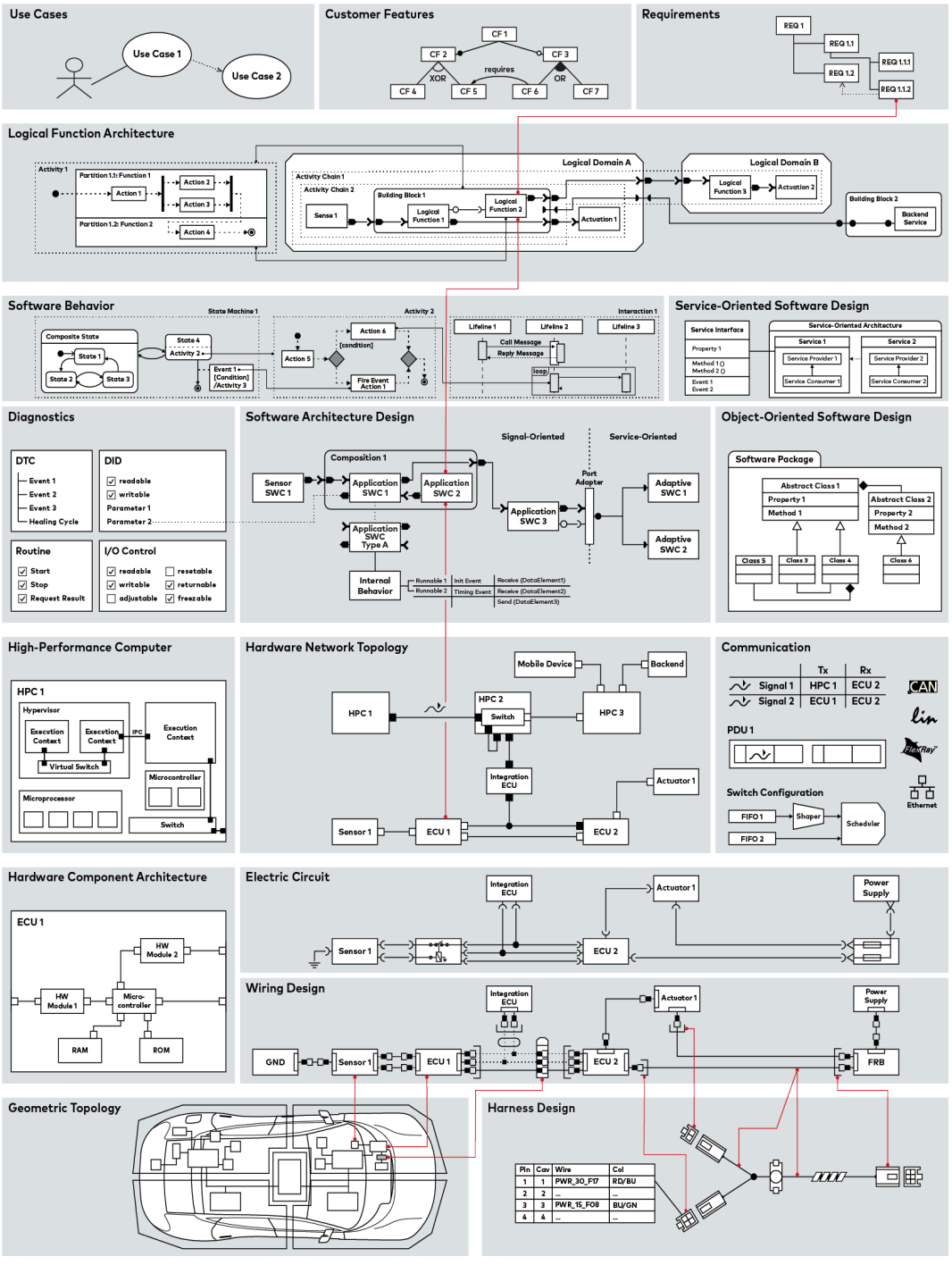

Abbildung A.1.: Detaillierte Abbildung des PREEvision Schichtenmodels