Andreas Voraberger, BSc.

Entwicklung einer AUTOSAR konformen Multicore-Serienproduktions-Plattform

MASTERARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

Masterstudium Softwareentwicklung - Wirtschaft

eingereicht an der

Technischen Universität Graz

Betreuer

Dipl.-Ing. Dr.techn. Christian Kreiner
Dipl.-Ing. Georg Macher

Institut für Technische Informatik

Dipl.-Ing. Dr.techn. Eric Armengaud
AVL List GmbH

Graz, Oktober 2015

Kurzfassung

Im letzten Jahrzehnt sind die Anforderungen im Hardware und Software Bereich nahezu explodiert. Der Umfang an Schnittstellen, zahllose Standards und die unüberschaubare Komplexität der Softwaremodule schufen die Notwendigkeit eines offenen Industriestandards. Hierbei handelt es sich um AUTomotive Open System ARchitecture (AUTOSAR), einer Partnerschaft verschiedener Automobilhersteller und Zulieferer. Heute ist der Standard, momentan in Version 4.2, global akzeptiert und trägt wesentlich zur Austauschbarkeit und Kompatibilität von Hardware und Softwaremodulen bei.
Durch die enorme Zunahme an sekundären Systemen im Fahrzeug, ergibt sich eine Anforderungssteigerung an die Rechenleistung. Diese ist durch klassische Einkern Prozessoren praktisch nicht mehr zu bewältigen. Daher ist das Thema Mehrkernprozessor im Automotive Bereich aktueller als je zuvor, wirft allerdings verschiedene Probleme auf, welche unbedingt zu lösen sind.
Diese Diplomarbeit beschäftigte sich mich mit einer AUTOSAR konformen Entwicklungsumgebung. Die Hauptaufgabe bestand in der Erstellung eines Mehrkernprojekttemplates, welches ein effizientes Einpflegen von Anwendungssoftware ermöglicht. Als Unterstützung wurde ein Generator zur automatischen Runtime Environment (RTE)-Signalkonfiguration entwickelt, dieser erstellt AUTOSAR konforme AUTOSAR XML (ARXML)-Dateien auf Basis von CAN DataBase Container (DBC)-Dateien.
Als erstes Ergebnis dieser Arbeit wurde ein Demonstrator in das Projekttemplate integriert. Dieser ist ein Aufbau bestehend aus einem Gaspedal, einer Drosselklappe und einer TriCore Steuerplatine. Gezeigt wurde dadurch die reibungslose Verarbeitung analoger Daten und die Interaktion über den Controller Area Network (CAN)-Bus in einer Mehrkernumgebung.
Nach erfolgreicher Integration des Demonstrators beschäftigte sich das Projekt damit, ein Kundenprojekt in das Projekttemplate einzupflegen. Der Fokus lag nicht mehr auf der generellen Funktionsfähigkeit des Projekttemplates, sondern auf einzelnen komplexeren Basissoftwaremodulen, der Steuerung über die RTE und der Handhabung für den Anwendungssoftwareentwickler. Das abschließende Ziel dieser Diplomarbeit war die Bewertung der Erfolgsfaktoren dieses AUTOSAR-Toolchain Ansatzes für die Serienproduktion.

Abstract

The requirements within the hardware and software field nearly exploded in the last decades. The amount of interfaces, countless standards and the incomprehensible complexity of the software modules, made open industry standards necessary. This involves the AUTOSAR, an alliance of different automotive manufacturers and suppliers. This standard, currently in version 4.2., is nowadays globally accepted and contributes significantly to the interchangeability and compatibility of hardware and software modules. The market requirements, concerning the computing power, increase due to the radical growth of secondary systems in the automobile. These computing powers are not manageable with classical single-core processors. Therefore multi-core processors are more relevant than ever before within the automotive area. However, it also raises various different problems, which need to be solved.

This diploma thesis explores and discusses AUTOSAR-compliant development environments. The main task was to create a multi-core project template, which makes it possible to effectively and rapidly migrate application software. A generator was created to support the RTE-Signal Configuration and Scaling, which creates AUTOSAR compliant ARXML-Data on the basis of CAN DBC-files.

The first result of this work, was the integration of a demonstrator in the project template. This structure consists of a throttle, a valve and a TriCore control board. With this the smooth processing of analogue data, as well as the interaction via the CAN-Bus in a multi-core environment was shown.

After the integration of the demonstrator has been successful, the project migrated a real customer project in the project template. The focus no longer lay on the general functionality of the project template, but on the individual more complex basic software modules, the control system via the RTE and on how the application software developers use it. The final aim of this diploma thesis was the evaluation of the success factors of this AUTOSAR-Toolchain approach for the serial production.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit identisch.

Statuary Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources. The text document, which has been uploaded to TUGRAZonline, is identical with the present master thesis.

Graz, am 29.10.2015

Ort, Datum
Place, Date

Unterschrift
Signature

Danksagung

Diese Diplomarbeit ist im Zuge einer Zusammenarbeit zwischen dem Institut für Technische Informatik an der Technischen Universität Graz und der AVL List GmbH entstanden. Daher möchte ich zuerst meinem Betreuer vom Institut für Technische Informatik, Georg Macher für seinen Einsatz, seine Begeisterung und die fortwährende Unterstützung im Laufe des Projekts danken.
Von Seiten der AVL List GmbH gilt mein Dank Eric Armengaud für die Betreuung sowie Harald, Ismar und Martin, welche mir in der Firma immer mit Rat und Tat zur Seite standen.
Ein großer Dank gilt Stefan Pointner, denn wir konnten alle Herausforderungen, beginnend bei der Studienwahl bis zum Abschluss unserer Diplomarbeit, auf unseren Wegen gemeinsam meistern.
Weiters möchte ich mich bei meinen Eltern, Johann und Kornelia Voraberger für Ihre Unterstützung und den stetigen Rückhalt auf meinem Lebensweg bedanken. Außerdem gilt mein Dank meinen Geschwistern, Thomas und Christina, sowie den Familien Burgstaller, Wiesinger und Herler, welche immer ein offenes Ohr für mich hatten.
Abschließen möchte ich mit dem Dank an Sabrina Ter-Haar, der wichtigsten Person an meiner Seite. Danke für deine Herzlichkeit, Geduld und Unterstützung im vergangenen Jahr.

Andreas Voraberger
Graz, Oktober 2015

Danke an

Georg, Christian, Eric, Harald, Ismar, Martin, Stefan, Johann, Kornelia, Thomas, Christina, Friedrich, Marianne, Ferdinand, Marianne, Wolfgang, Klaudia und Sabrina.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Es ist ein Trend unserer Zeit, dass mechanische Systeme zunehmend durch elektronische ausgetauscht werden. Dieser Wandel der Technologie wird durch die steigende Anzahl an notwendigen Systemen, auf immer kleiner werdendem Raum, vorangetrieben und durch die Möglichkeit, immer kleinere Kontrollstrukturen verwenden zu können, unterstützt. [46]
Im Automotive Bereich manifestiert sich diese Tatsache im letzten Jahrzehnt, durch die explodierende Anzahl an eingebetteten Systemen pro Fahrzeug.
Zu Beginn der Mikrocomputer Ära in der Automotive Industrie ging die Entwicklung Richtung einer einzelnen Electric Control Unit (ECU) pro funktionalem System. So entwickelte Bosch ein Netzwerkprotokoll zur effizienten Kommunikation zwischen einzelnen Steuergeräten. Dieses Protokoll nennt sich CAN, es ermöglichte den Transfer großer Datenmengen in Echtzeit und wurde 1990 erstmals in einem Fahrzeug eingesetzt. In der Praxis wurden diese zahlreichen Steuergeräte von vielen verschiedenen Zulieferern produziert. Dies hatte zur Folge, dass aufgrund von unterschiedlichen Standards und Schnittstellen, das Einbinden neuer Steuergeräte sehr unübersichtlich wurde. Um diesem unnötigen Anstieg an Komplexität entgegen zu wirken, wurde im Jahr 2003 das AUTOSAR-Konsortium ins Leben gerufen. [57] Es wurde ein offener Industriestandard entwickelt, welcher unter anderem Spezifikationen zur Softwarearchitektur, Programminterfaces, Dateiformate und die AUTOSAR-Methodologie enthält [31].
Im Jahr 2008 existierten bereits durchschnittlich 80 vernetzte Systeme in einem neuen Personenkraftwagen [44]. Systeme wie Antiblockiersystem (ABS), Airbag und Antriebsstrang wurden weitgehend unabhängig voneinander gesteuert. Es ergab sich zwar der Vorteil, dass die einzelnen Systeme unabhängig von ihrer Umgebung waren, allerdings ließ sich der Vorteil von Vernetzung und Informationsaustausch nicht von der Hand weisen. Dadurch entstand eine komplexe und kostspielige Vernetzung verschiedenster Electric Control Units (ECUs), quer durch das ganze Fahrzeug.
Überladene Netzwerke weisen eine erhöhte Fehlerquote auf, sie müssen intensiver getestet werden und sind schwerer wartbar [44]. Daher gibt es eine steigende Tendenz die Anzahl an ECUs pro Fahrzeug zu verringern. Dieser Ansatz wurde durch die AUTOSAR-Initiative ermöglicht und fördert eine zentralisiertere Architektur, wodurch Kommunikationswege und der Umfang des Fahrzeugnetzwerkes deutlich reduziert werden konnten. [49]
Diese Entwicklungen, plus die steigende Anzahl an Sekundärsystemen und Unterhaltungssoftware im Fahrzeug führen dazu, dass die geforderte Rechenleistung zunehmend zum

Flaschenhals wird. Jener Bereich, der in der Desktop Computer Industrie bereits längst gängige Praxis ist, greift nun auch auf die Automotive Industrie über, dabei handelt es sich um die Mehrkernthematik. [38] Chiphersteller haben den Punkt erreicht an dem es nicht mehr möglich ist, die geforderte Leistung durch anheben des Prozessortakts zu erreichen. Der Mehrkernansatz eröffnet nun einerseits viele interessante und effiziente Wege der Parallelisierung, stellt jedoch gleichzeitig Probleme mit verteilten Ressourcen und der Einhaltung von Echtzeitkriterien in den Raum. [49]
Das AUTOSAR-Release 4.1 inkludiert nun erstmals Entwicklungsrichtlinien bezüglich eines Mehrkernsystems. [4]

1.1 Aufbau und Ziel dieser Arbeit

Das Ziel dieser Diplomarbeit ist, ein Mehrkernprojekttemplate zu erstellen. Es wird eine zertifizierbare, für die industrielle Serienproduktion einsetzbare AUTOSAR-Entwicklungsumgebung verwendet. Dieses Projekttemplate soll ein zügiges und reibungsloses Einpflegen von Anwendungssoftware ermöglichen. Als Unterstützung wird ein Generator zur automatischen RTE-Signalkonfiguration und Skalierung entwickelt.
Nach erfolgreicher Erstellung des Projekttemplates, wird ein Demonstrator, bestehend aus Gaspedal, Drosselklappe und TriCore Steuerplatine, integriert. Gezeigt wird dadurch die reibungslose Verarbeitung analoger Daten und die Interaktion über den CAN-Bus in einer Mehrkernumgebung, sowie die Steuerung über die RTE. Darauf aufbauend erfolgt die Integration eines realen Kundenprojekts, wo der Fokus nicht mehr auf der generellen Funktionsfähigkeit des Projekttemplates liegt, sondern auf einzelnen komplexeren BSW-Modulen und der Migration von Anwendungssoftware (ASW).
Das Hauptaugenmerk in Bezug auf die AUTOSAR-Entwicklungsumgebung liegt auf den mitgelieferten BSW-Modulen. Im Zuge des Aufbaus des Projekttemplates gilt es sich mit den verschiedenen AUTOSAR-Schichten auseinander zu setzen und darin die notwendigen BSW-Module zu konfigurieren.
Am Ende dieser Diplomarbeit steht die Bewertung der Erfolgsfaktoren dieses AUTOSAR-Toolchain Ansatzes für die Serienproduktion im Mittelpunkt.

1.2 Kontext der Arbeit zur Industrie

In Kooperation mit dem Institut für Technische Informatik der TU Graz und der AVL List GmbH entstand diese Diplomarbeit.
Der AUTOSAR-Standard hat sich im letzten Jahrzehnt zum führenden globalen Automotive Software Standard entwickelt und genießt höchste Akzeptanz. [31] Diese Entwicklung führt zu einer relativ großen Bandbreite an Tools und Entwicklungsumgebungen in der AUTOSAR konformen Softwareentwicklung, was eine Evaluierung dieser unumgänglich macht.

Kapitel 2

Entwicklung von Softwareprojekten im Automotive Bereich

2.1 Anfänge der Automotive Softwareentwicklung

Die Branche der Automotive Industrie hat seit der Erscheinung von Mikrocomputern in kürzester Zeit sämtliche Entwicklungssprünge der traditionellen Desktop Computer erfahren. Durch die wachsenden Ansprüche an die Steuergeräte im Fahrzeug, wurde der Evolutionssprung in der bis dahin von Elektrotechnikern dominierten Domäne sehr rasch vollzogen. Während anfänglich der Hauptnutzen in der Basissoftware von Steuergeräten gesehen wurde, hat sich dieser Bereich heute zu interaktiven Netzwerken von unzähligen ECUs ausgewachsen. [24]
Da die verschiedenen Steuergeräte von unterschiedlichen Herstellern entwickelt wurden, ergaben sich die Probleme von Integrierbarkeit und Wiederverwertbarkeit. [57] Das anfängliche Fehlen global akzeptierter Entwicklungsstandards und Plattformen, sowie die enorme Komplexität der ECU-Netzwerke und die verschiedenen Auffassungen der Hersteller über die Bedeutung von Automotive Software, führten zu erheblichen Problemen. Der Schlüssel zur erfolgreichen Softwareentwicklung in der Fahrzeugindustrie führte schlussendlich über den unausweichlichen Weg der Standardisierung. Neue Projekte werden nun ausschließlich nach den gegeben Standards definiert und erfüllen somit die Anforderungen der heutigen Zeit. In der Praxis existieren weiterhin veraltete Systeme, welche häufig in standardisierte Projekte migriert werden müssen. Man unterscheidet zwischen Prozess und Produkt basierten Ansätzen. [24]

2.2 Standardisierung - Prozessbasiert

Da verschiedene bereits ausgearbeitete Qualitätsstandards, wie zum Beispiel ISO9001, dem Automotive Software Bereich nicht gerecht wurden, ging man zuerst nach dem internationalen Standard für Softwareprozesse, der ISO/IEC 15504 (SPICE) vor. [24]
Im Jahr 2001 beschloss eine Initiative der führenden europäischen Autohersteller die Entwicklung des sogenannten Automotive-SPICE. Dabei handelt es sich um ein zur ISO/IEC 15504 konformes Modell, welches speziell für den Prozess der Softwareentwicklung im Au-

tomotive Bereich entwickelt wurde. Es umfasst drei Basiskonzepte: das Prozess-Referenz-Modell, welches Definitionen über Entwicklungsprozesse und dessen Zusammenhänge gibt, das Prozess-Bewertungs-Modell, welches Möglichkeiten zur Prozessbeurteilung liefert und einem Framework zur Bewertung auf Basis des Prozess-Bewertungs-Modells. [42] Dieser Standard führte nicht nur zu deutlich höherer Softwarequalität und einer Möglichkeit Software miteinander zu vergleichen, sondern erzeugte auch wesentliche ungewollte Effekte. Diese wären eine weitaus bessere Projektüberwachung, Herstellerauswahl und schneller Identifikation interner Schwächen in einzelnen Produktprozessen. [24]

2.3 Standardisierung - Produktbasiert

Ein Zusammenschluss von führende Fahrzeughersteller und Autoindustriezulieferer im Jahr 1990 unter dem Namen Motor Industry Software Reliability Association (MISRA) führte zur Erstellung verschiedener Richtlinien, welche die Sicherheit und Zuverlässigkeit von Automotive Software erhöhen sollen. Dazu gehören unter anderem Richtlinien für die Softwareentwicklung für Fahrzeuge, den korrekten Einsatz der Programmiersprachen C und C++ in Fahrzeugsoftware, für kritische Systeme, für modellbasierte Softwareentwicklung und für die Definition von Testcases. [64]
Im Jahr 1993 gründeten führende deutsche Automobilunternehmen das OSEK-Konsortium. Neben verschiedenen Spezifikationen, bezüglich Kommunikationsstacks und Netzwerkprotokollen, ist der bedeutendste Standard das OSEK-OS. Es definiert ein Echtzeitsystem mit standardisierten Interfaces welches mit Hardware Events in einer modularen Umgebung umgehen kann und leicht skalierbar ist. [1]
In den nachfolgenden Kapiteln werden die aktuell wohl wichtigsten Standards, OSEK sowie AUTOSAR ausführlich beschrieben.

Kapitel 3

OSEK/VDX

3.1 Einführung in OSEK

Der Wunsch nach Standardisierung und somit Kostensenkung entstand sehr früh, durch die immer komplexer werdenden, vernetzten Teilsysteme in der Fahrzeugindustrie. Bereits in den Anfängen erkannte man das Potential einer modularen Architektur. Der Fokus sollte dabei auf klar definierte Schnittstellen liegen. Zunächst konzentrierte man sich hauptsächlich auf das Hardwaredesign. Durch die steigende Komplexität der Softwarekomponenten rücken diese jedoch heutzutage stetig in den Vordergrund. [67]
Zu diesem Zweck wurde 1993 das industrielle Standardisierungsgremium OSEK, von verschiedenen, großteils deutschen, Fahrzeugherstellern und Zulieferern der Automotive Industrie gegründet. Nach dem Zusammenschluss mit der französischen Vehicle Distributed Executive (VDX)-Initiative erschuf man 1995 schlussendlich eine Definition für einen Betriebssystemstandard, auf welchem heute unter anderem auch das AUTOSAR-Betriebssystem aufbaut. Seither nennt man dieses Gremium OSEK/VDX, welches im Rahmen dieser Arbeit allerdings als OSEK bezeichnet wird. [67]
Das Ziel besteht darin, einen offenen Industriestandard für verteilte Steuergeräte in Fahrzeugen zu entwickeln. Es gibt zwei Hauptgründe für dieses Vorhaben. Zum Einen der steigende Aufwand in der Entwicklung von Steuergeräten und zum Anderen die Inkompatibilität zwischen verschiedenen Herstellern, aufgrund von unterschiedlichen Protokollen und Interfaces. [52]
Teilnehmende Firmen profitieren von einhergehenden Kosteneinsparungen, sowie von einer erwarteten Qualitätssteigerung im Bereich der Steuergeräte. Diese Steigerung der Qualität äußert sich nicht allein in der Funktionalität oder Sicherheit, sondern vor allem auch in der Portierbarkeit, Wiederverwertbarkeit und Flexibilität. Es wird eine Struktur vorgelegt, in denen sich die Marktteilnehmer hauptsächlich an spezifizierte Interfaces, nicht aber an Architekturentscheidungen halten müssen. Um den modularen Ansatz zu unterstützen, wird die ASW durch die Spezifikation von abstrakten Interfaces von der BSW entkoppelt, sowie auch die Benutzerschnittstellen unabhängig von der Hardware und dem Netzwerk spezifiziert. Die OSEK/VDX-Spezifikation beruht auf vier wesentlichen Konzepten. Hierzu zählt die Spezifikation des "OSEK-OS", die interne und externe Kommunikation, sowie auch die OSEK-Beschreibungssprache. Das Zusammenspiel der ersten drei Komponenten, welche während der Laufzeit aktiv sind, ist in Abbildung 3.1 ersichtlich. [52]

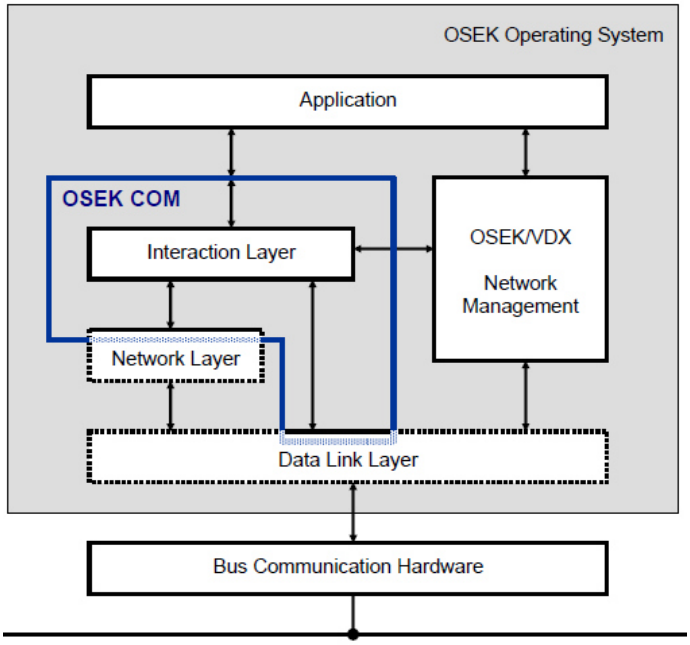

Abbildung 3.1: OSEK-Schichtenmodel (Quelle: [52])

3.2 OSEK-Betriebssystem

Das "OSEK-OS" verfolgt als Einkern-Betriebssystem, das Ziel eine effiziente Ressourcennutzung für verteilte ECUs zu ermöglichen. In der Automotive Industrie handelt es sich stets um zeitkritische Anwendungen mit einer harten Echtzeitanforderung. Um dieser Bedingung nach zu kommen werden alle notwendigen Funktionen für ein ereignisgesteuertes Betriebssystem zur Verfügung gestellt. "OSEK-OS" definiert einen Standard, der ein Maximum an Performance bei einem Minimum an Ressourcenverbrauch, auf einer großen Hardwarebandbreite, gewährleisten soll. Die Schnittstellen zwischen der ASW und dem Betriebssystem werden über standardisierte Systemservices definiert, was die ASW entkoppelt und somit die Portierbarkeit, Wiederverwertbarkeit und Wartbarkeit stark erhöht. Verschiedene Tasks, Applikationen, Zeitgeber und sonstige Betriebssystemkonfigurationen werden statisch, vor der Laufzeit, definiert. Das lauffähige Betriebssystem wird im Anschluss an die Konfiguration, während der Systemgenerierung erstellt. Dies hat zur Auswirkung, dass es während der Laufzeit nicht mehr modifiziert werden kann. [59]
Ein Hauptziel der "OSEK-OS"-Definition ist die Portierbarkeit von ASW. Das bedeutet, dass die Verwendung der selben Software auf unterschiedlichen ECUs, bei möglichst wenigen Anpassungen, passiert. Um diese Vorgabe umsetzen zu können, räumt OSEK allerdings ein, dass die Standardisierungen alleine nicht ausreichend sind. Es wird auf

einige andere, zentrale Merkmale die in einem Softwareentwicklungsprozess wesentlich sind verwiesen. Beispiele hierfür sind internationale Entwicklungsrichtlinien, das Dateimanagementsystem, der Compiler, die Speicherarchitektur der ECU, das Zeitmanagement der ECU, verschiedene Mikrocontroller Schnittstellen und der Einsatz von Application Programming Interface (API)-Aufrufen selbst. [59]
Um den unterschiedlichen Anforderungen bezüglich Funktionalität und Kapazität der Industrie gerecht zu werden, wurden vier unterschiedliche Konformitätsklassen definiert. Diese Klassen ermöglichen eine genaue Anpassung an die verwendeten Hardware- und Softwarekomponenten im System. Eine Applikation, welche für eine bestimmte Klasse definiert wurde, muss nahtlos in eine andere Implementierung eines OSEK-Systems portiert werden können. Nur wenn die Definition der Services, der Funktionsbereich und das Systemverhalten exakt der jeweiligen Klasse entsprechen, darf von einer OSEK konformen Entwicklung gesprochen werden. Dadurch wird eine strikte Einhaltung des Standards sicher gestellt. [59]
Die vorhin angesprochenen standardisierten Systemservices, welche die ASW vom Betriebssystem entkoppelt, lassen sich anhand ihrer Funktionalität in verschiedene Gruppen einteilen. Die erste Gruppe stellt das Taskmanagement dar. Hier werden Verhaltensweisen bezüglich Taskaktivierung, Taskterminierung, Taskwechsel sowie Taskzustände definiert. Die zweite Gruppe beschäftigt sich mit den Synchronisierungsvorgängen. Einerseits beziehen sie sich auf das Ressourcenmanagement und andererseits das Eventmanagement. Eine Gruppe umfasst das Interruptmanagement, welches Services für die Interruptverwaltung zur Verfügung stellt. Eine weitere Gruppe regelt die Alarme, welche als relative und absolute definiert sind. Zusätzliche Gruppen regeln die interne Prozessor-Nachrichtenverwaltung und das Fehlermanagement. [59]

3.3 OSEK-Kommunikation

Die OSEK Kommunikation (OSEK COM) ist eine standardisierte Kommunikationsumgebung für eine OSEK konforme ASW-Entwicklung, welche eine stark erhöhte Portierbarkeit von ASW zum Ziel hat. Hier ist hervorzuheben, dass diese Standardisierung zwar für OSEK konforme Betriebssysteme erdacht wurde, jedoch ausdrücklich auch für nicht konforme Systeme umsetzbar ist. Hauptsächlich beschäftigt man sich in diesem Rahmen mit der Spezifikation von Kommunikationsinterfaces und Kommunikationsverhalten. Diese dienen sowohl für die interne als auch externe Kommunikation einer ECU zwischen vernetzten Komponenten, innerhalb eines Fahrzeugs. In diesem Kontext definiert man vorrangig die genaue Funktionalität der Kommunikationselemente. Es werden zum Beispiel API-Aufrufe für Services zum Datenaustausch zwischen Interrupts und Tasks definiert. Um die Software von unterschiedlichen internen und externen Kommunikationsprotokollen zu entkoppeln und somit die Wiederverwertbarkeit und Portierbarkeit zu erhöhen, darf der Zugriff auf diese Services ausschließlich über die eben erwähnten API-Aufrufe stattfinden. Zugunsten der Skalierbarkeit wurde weitgehend auf Hardwaredefinitionen verzichtet.

Wesentlich hingegen ist ein möglichst ressourcenschonender Betrieb auf vielen gängigen Plattformen. [53]
Aus Abbildung 3.1 lässt sich die Positionierung der OSEK COM-Komponente, sowie der drei wesentlichen adressierten Kommunikationsschichten herauslesen. Die oberste der drei Schichten stellt die Interaktionsschicht dar Diese beinhaltet die OSEK COM API für den internen Nachrichtenverkehr, sowie Services von unteren Schichten für die externe Kommunikation. Die darunter liegende Schicht ist die Netzwerkschicht. Für diese werden von OSEK COM Mindestanforderungen definiert, um den vollen Funktionsumfang der, soeben angesprochenen, Interaktionsschicht zu gewährleisten. Im Generellen ist die Netzwerkschicht vom benutzten Kommunikationsprotokoll abhängig und ist für die Nachrichtenaufteilung sowie den Nachrichtenempfang zuständig. Dies wird durch die Verwendung der Services der Data-Link-Schicht erreicht, wo OSEK COM wiederum nur die Mindestanforderungen definiert. [53]

3.4 OSEK-Netzwerkmanagement

Die OSEK Netzwerk Monitoring (OSEK NM)-Komponente ist eine standardisierte Basisinfrastruktur, welche die reibungslose Kommunikation zwischen mehreren ECUs zum Ziel hat. Das Hauptziel ist hierbei die Einsparung von Entwicklungszeit, sowie die Steigerung von Zuverlässigkeit und Sicherheit. Um OSEK konform zu Entwickeln, muss man die OSEK NM-Definitionen in jedem verbundenen Netzwerkknoten umsetzen. [54]
Es werden zwei alternative Netzwerkmonitoring Mechanismen bereitgestellt. Das indirekte Monitoring überwacht die Anwendungsnachrichten und das direkte Monitoring kommuniziert mit der Netzwerkmanagement Schicht. [63] Zusätzlich werden zwei standardisierte Interfaces bereitgestellt. Einerseits auf der Softwareebene zwischen der ASW und der Netzwerkmanagementschicht, und andererseits zwischen den Kommunikationsmedien. Weitere Services, welche innerhalb dieser Definition vorgegeben werden, sind die Konfiguration und Initialisierung des Netzwerkinterfaces, das Monitoring verschiedener Knoten, ein Status Monitoring vom gesamten Netzwerk, der Diagnosesupport und die entsprechenden API-Funktionen. [54]

3.5 OSEK-Beschreibungssprache

Eine Vereinheitlichung der Konfigurationsdateien ist unumgänglich, da es nicht zielführend wäre, bei der Entwicklung der einzelnen Komponenten im Detail auf OSEK-Datentypen eingehen zu müssen. Der OSEK Implementation Language (OIL)-Standard definiert alle relevanten Operating System (OS)-Objekte und ist wesentlich für die Portierbarkeit von Konfigurationen. [65] Das Hauptziel des OIL-Standards besteht darin, einen Mechanismus für die Konfiguration einer OSEK konformen Anwendung in einer einzelnen Central

Processing Unit (CPU) zur Verfügung zu stellen. Dabei besitzt jede CPU seine eigene OIL-Beschreibung. [55]
Eine OIL-Datei setzt sich immer aus zwei Teilen zusammen, nämlich der Implementierungsdefinition und der Applikationsdefinition. Die Implementierungsdefinition spezifiziert alle zur Implementierung gehörigen Features und Beschränkungen, wie zum Beispiel die möglichen Prioritäten von Tasks. Der zweite Teil, die Applikationsdefinition, beinhaltet die Deklaration des CPU-Containerobjekts, wo alle anwendungsspezifischen Standardobjekte definiert werden. Hierbei gilt, dass immer alle Standardobjekte vorhanden sind, wobei jedes Objekt selbst veränderbare Attribute besitzt. Die Standardobjekte sind CPU, OS, APPMODE, ISR, RESOURCE, TASK, COUNTER, EVENT, ALARM und COM. [65]

Kapitel 4

AUTomotive Open System ARchitecture (AUTOSAR)

4.1 Entstehung von AUTOSAR

Das AUTOSAR-Konsortium wurde 2003 von den Kern-Partnern Bayerische Motoren Werke AG, Robert BOSCH GmbH, Continental AG, Daimler AG, Ford Motor Company, General Motors Holding LLC, Peugeot Citroen Automobile S.A., Toyota Motor Corporation und Volkswagen AG gegründet [3]. Heute werden mehr als 80 Prozent der globalen Automotive Produktion von AUTOSAR-Mitgliedern erstellt [31].
Die Partnerschaft gliedert sich in vier unterschiedliche Gruppen, wie sich aus der Struktur in Abbildung 4.1 herauslesen lässt.

Die neun Kernpartner, welche ihr KnowHow zusammen getragen haben, um einen Standard für eine offene System Architektur in der Automotive Industrie voran zu treiben, welche den Bedürfnissen von zukünftiger Fahrzeug Software gerecht wird. [3] Sie bilden den Kern einer verteilten sowie virtuellen Organisation, welcher die Steuerung, Kontrolle, das Management und administrative Verantwortung dieses Projekts obliegt. [26]
Die zweite wichtige Gruppe bilden die Premium Mitglieder, welche gemeinsam mit den Kernpartnern aktiv an der Entwicklung des AUTOSAR-Standards beteiligt sind. Zu deren Aufgaben zählen unter anderem Arbeitsgruppen leiten, technische Entscheidungen treffen und die Bearbeitung von Veröffentlichungen. [16]
Die große Gruppe der "Associate Partners" repräsentiert, mit aktuell ca. 100 Mitgliedern [3], die aktiven Nutzer des AUTOSAR-Standards. Sie erhalten alle relevanten Dokumente und Informationen im Voraus, sind allerdings nicht aktiv an der Entwicklung und Standardisierung beteiligt. Abschließend existiert noch eine Kategorie, welche mit den "Development Partners" und "Attendees" die Rolle der Unterstützer einnimmt. [16]
Der Erfolg begründet sich aus den Vorteilen, die für jeden Bereich der Branche entstehen. Für die OEM entsteht die Möglichkeit der Wiederverwendbarkeit von Softwaremodulen und eine starke Vereinfachung von Integrativen Tasks. Dies führt zu dem Ergebnis, dass man sich viel intensiver auf die Entwicklung innovativer Technologie fokussieren kann, wobei man automatisch eine Kostensenkung im Gegensatz zum weniger standardisierten Ansatz erreicht. Für die Zulieferer ergibt sich der Vorteil der Kompatibilität zueinander,

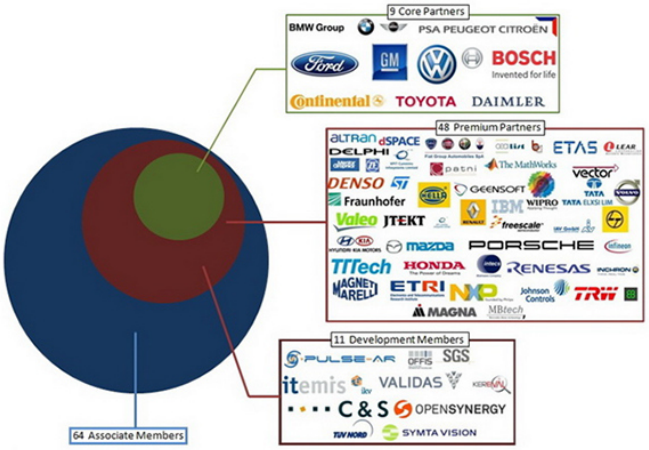

Abbildung 4.1: Globale Teilnehmer im AUTOSAR-Netzwerk (Quelle: [40])

was eine Fokussierung auf die Kernbereiche des jeweiligen Marktteilnehmers begünstigt. Zusätzlich ist zu bedenken, dass man die Anzahl unterschiedlicher Softwareversionen, für individuelle Kunden, in Grenzen halten kann und somit die Wiederverwertbarkeit innerhalb des jeweiligen Betriebs verstärkt wird. [3]
Wenn man die BSW betrachtet, stützte sich AUTOSAR weitgehend auf die bereits vorhandenen und in der Industrie weitläufig akzeptierten, Standarddefinitionen von OSEK, Internationale Organisation für Normung (ISO), Herstellerinitiative Software (HIS), Association for Standardisation of Automation and Measuring Systems (ASAM) sowie auf die Definitionen von Kommunikationsstandards wie CAN. AUTOSAR gibt keine verbindlichen Vorschriften bezüglich der Implementierung vor, sondern erarbeitet lediglich Standards, wobei die Umsetzung den Marktteilnehmern selbst überlassen wird. [66]
Neben der BSW wird ein Betriebssystem, Kommunikationsservices und eine RTE bereitgestellt. Dennoch fokussiert sich AUTOSAR von Beginn an auf den generellen Entwicklungsprozess, deshalb wurden nicht nur die Interfaces zwischen den teilnehmenden Mitgliedern standardisiert, sondern auch eine Methodologie definiert. Somit kann man das volle Potential nutzen, wenn man sich nicht nur auf die Basis Softwaremodule versteift, sondern alle Elemente systematisch nach dem AUTOSAR-Standard implementiert. [57]
Heute werden maschinenlesbare ECU-Spezifikationen verwendet. Dieser Umstand ermöglicht die Wiederverwendung und Unabhängigkeit von ECU, BSW, ASW und den jeweiligen Softwaretests. Dadurch entstand ein Trend zur effizienten Arbeitsteilung, mit dem Hang zur Spezialisierung in der Automotive Industrie. [57]

4.2 Ziele von AUTOSAR

Die AUTOSAR-Gemeinschaft strebt einen Standard an, welcher die verschiedenen Bestandteile zukünftiger Fahrzeugapplikationen weiter entkoppelt und die notwendigen Interfaces weitgehend standardisiert. Während der gesamten Entwicklung der Richtlinien wird das Ziel, dass der Standard einen Gewinn für alle beteiligten Partner darstellt, stets im Auge behalten. [26] Die generellen Ziele wurden von den Kernpartnern definiert und richten sich nach dem Motto: "Cooperate on standards, compete on implementation". [66] Die größte Überlegenheit ergibt sich aus den Möglichkeiten der Wiederverwertbarkeit von Softwaremodulen und der stark vereinfachten Integration klar definierter Interfaces. Dies resultiert in einer stark erhöhten Flexibilität in der Projektplanung und einer breiten Skalierbarkeit einzelner Softwaremodule. AUTOSAR zielt darauf ab, in Zukunft der Standard in der Automotive Industrie zu sein und adressiert deswegen mit seinem Ansatz sowohl Sicherheitsfragen, Risiko- und Komplexitätsminimierung als auch die Wartbarkeit erstellter Module. [3]
In der ersten Entwicklungsphase des Standards, beschäftigte man sich unter anderem mit dem Antriebsstrang, Fahrwerk und Sicherheitsthemen. In Bezug auf diese Domänen wurde der Fokus auf einige genau definierte Ziele gelegt. Man beschäftigte sich mit den Kriterien von Sicherheit, Wartbarkeit und Verfügbarkeit. Durch die Beteiligung verschiedenster Industrieführer, war auch die Skalierbarkeit auf unterschiedlichsten Plattformen von zentralem Interesse. Dadurch war eine weitgehende Normierung der Basissystemfunktionen unumgänglich. Ein hoher Grad an Austauschbarkeit und Wiederverwertbarkeit in der gesamten Branche war die Folge. [26]
Aus den von AUTOSAR definierten Zielen lässt sich für alle Beteiligten eine deutliche Qualitätssteigerung in der Erstellung und Verarbeitung von Fahrzeugsoftware, sowie eine Kostensenkung von skalierbaren Systemen ableiten.

4.3 Errungenschaften in Phasen und Releases

Aktuell setzt sich der Lebenszyklus der AUTOSAR-Releases aus vier unterschiedlichen Phasen zusammen. Die Aufbauphase bezeichnet die Erstentwicklung eines Release. In der Entwicklungsphase gibt es untergeordnete Releases und Revisionen, welche Neuerungen sowie Fehlerbehebungen beinhalten. Die Wartungsphase beinhaltet lediglich Fehlerkorrekturen. In der Dokumentierungsphase erfolgt die Wartung ausschließlich durch Erweiterung der List of Known Issues (LOKI). AUTOSAR setzt bei der Verbreitung ihrer Standardversionen darauf, dass immer nur zwei Releases aktiv weiterentwickelt werden. [58]
Das AUTOSAR-Konsortium hat eine gemeinsame Roadmap definiert, welche vier aufeinander folgende Phasen vorgibt. Innerhalb dieser wird parallel an den Hauptreleases und Korrekturreleases gearbeitet. (Siehe Abbildung 4.2)
Die **Phase 1** beinhaltet die AUTOSAR-Releases 1.0, 2.0 und 2.1. Das Hauptziel war die vollständige Spezifikation der Architektur, der Methodologie und der Templates in der

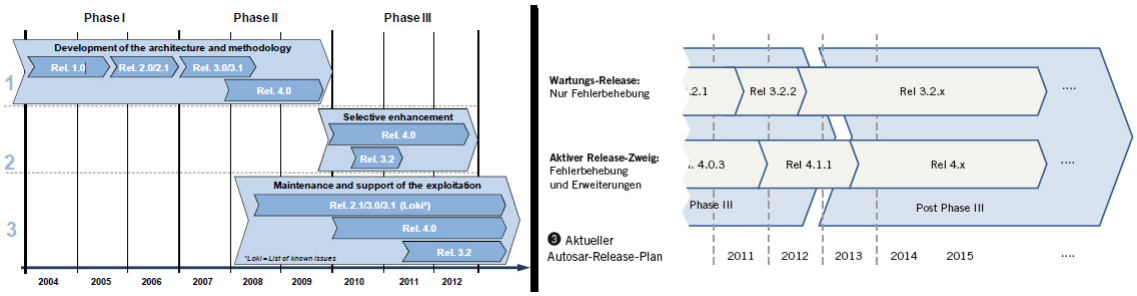

Abbildung 4.2: AUTOSAR-Zeitlinie (Quelle: [58, 31])

Zeit von 2003 bis 2006. Dabei beschäftigte man sich im Release 1.0 hauptsächlich mit der Evaluierung und Ausformulierung der Spezifikation. Ziel war es Teile der BSW zu validieren und Verfeinerungen am Konzept vorzunehmen. [31] Im Release 2.0 beschäftigte man sich damit, Feedback von Version 1.0 und bisher fehlende Module, wie LIN Unterstützung und I/O Hardwareabstraktion, einzuarbeiten. Im Anschluss wurde ein Validator implementiert, welcher als Basis für das Release 2.1 diente. Man verfolgte den Ansatz, die Verbindung zwischen den Parametern unterschiedlicher BSW-Module herzustellen. Dieses Release war die erste wirklich marktreife Veröffentlichung und es wurden sogleich auch in Serienproduktionen verwendet. [26]
AUTOSAR **Phase 2** war von 2007 bis 2009 geplant und beinhaltet die Releases 3.0, 3.1 und 4.0. Die Hauptziele waren die Einarbeitung der Ergebnisse von Phase 1 und der Feedbacks der Mitglieder. Anfang 2008 erschien Release 3.0, welches eine große Anzahl an Verbesserungen und Erweiterungen bereitstellte. Man beschäftigte sich nun bereits aktiv damit Interface Module für die interne Fahrzeugkommunikation zu entwickeln und Multimedia Benutzerschnittstellen sowie Systeme für Insassen- und Fußgängersicherheit zu definieren. Das Hauptziel von Release 3.1 war, On-Board-Diagnostic (OBD) für die BSW-Module zugänglich zu machen. Zum Abschluss dieser Periode folgt das Release 4.0, welches erstmals wieder neue Konzepte in Bezug auf die Architektur und die Methodologie beinhaltete. Man wandte sich der funktionalen Sicherheit, dem Kommunikationsstack und den Templates zu. Die ISO 26262 wurde integriert und man nahm Bezug auf die, für die Automotive Industrie noch neue, Mehrkern (MC)-Architektur. Zu diesem Zeitpunkt gab es bereits ein breites Spektrum an standardisierten Applikationsinterfaces für alle fünf adressierten Fahrzeug Domänen: Antriebsstrang, Fahrwerk, Insassen- und Fußgängersicherheit, Multimedia und "Body and Comfort". Gleichzeitig wurde der AUTOSAR-Standard bereits von vielen Unternehmen verwendet, um Serienreife Produkte auf den Markt zu bringen. Es war nun an der Zeit Möglichkeiten zur Prüfung der Übereinstimmung zum Standard zur Verfügung zu stellen. Dies war wesentlich, um ein Abweichen vom Standard von vornherein zu unterbinden, um weiterhin einen hohen Grad an Wiederverwertbarkeit und Portierbarkeit gewährleisten zu können. [29]
In **Phase 3** (2010 bis 2012) wurde man erstmals auf die Widersprüche zwischen Innovation und Stabilität aufmerksam. Es wurden, zu diesem Zeitpunkt, bereits 2 Hauptreleases paral-

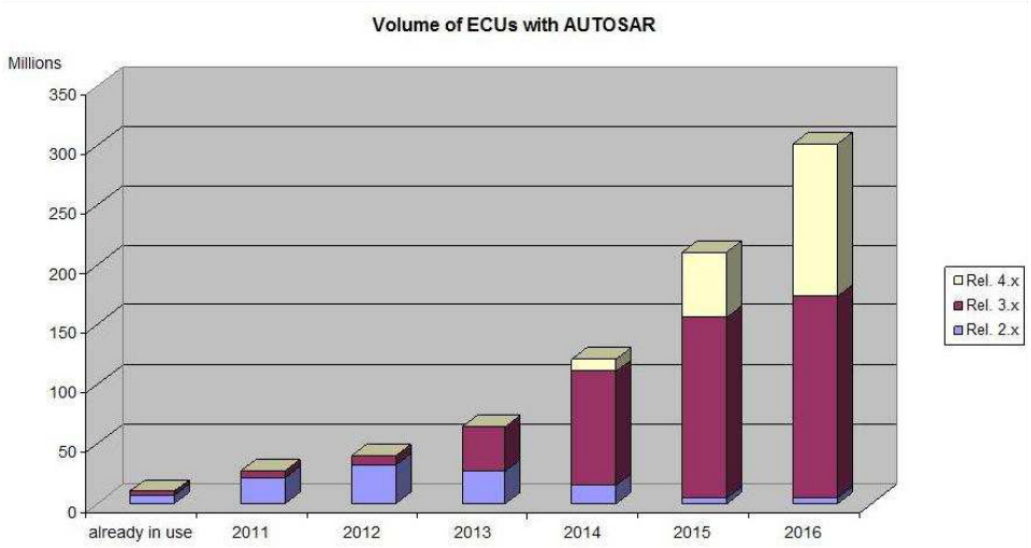

Abbildung 4.3: Volumen an AUTOSAR-ECUs (Quelle: [31])

lel genutzt, welche man beide warten und mit Updates versorgen musste. Daher beschloss man das Release 5.0 zu verschieben und sich voranging auf die Wartung und selektive Erweiterung, der in der Serienproduktion verwendeten Versionen, zu konzentrieren. Man beschäftigte sich weiter mit funktionaler Sicherheit, weitete die MC-Unterstützung aus, erweiterte verschiedene BSW-Module, sowie die Methodologie und Templates. [31, 29]
Seit dem Jahr 2013 befinden wir uns in der **Post Phase 3**. Das definierte Ziel ist die Anpassung an das Marktumfeld mit dem Fokus auf Abwärtskompatibilität sowie der Stabilisierung der bestehenden Releases 3.x und 4.x. In Bezug auf die Standarddefinition wird das Hauptaugenmerk auf die Weiterentwicklung der funktionalen Sicherheit, besonders in Bezug auf die Automotive Safety Integration Level (ASIL)-Zertifizierung, gelegt. Die Definition von MC-Architekturen wird bedeutend erweitert, zum Beispiel um Möglichkeiten für effizienteres "Load-Balancing" zu erhalten. Weitere wichtige Neuerungen werden rund um die ETHERNET Schnittstelle nötig. Außerdem weitet man die AUTOSAR-Methodologie aus, um aussagekräftigere Timing-Analysen zu ermöglichen und eine iterative Entwicklung zu unterstützen, harmonisiert die Diagnosesoftware-Implementierung und implementiert ein effizientes Energiemanagement zum Energiesparen in ECUs. [58]
In Abbildung 4.3 wird die vergangene und aktuell geplante Produktion an AUTOSAR-ECUs in den verschiedenen Releases von den Kernpartnern veranschaulicht.

4.4 Automotive System Architektur

Um die technische Umsetzung des AUTOSAR-Modells verstehen zu können, sollte man zunächst die generelle Fahrzeug System Architektur näher betrachten.

Auf die Hardware bezogen stellt sich der Aufbau wiederum als eine Abstraktion mit verschiedenen Stufen dar, wobei die oberste Schicht als "System Level" bezeichnet wird. Ein Fahrzeug hat eine Vielzahl an Netzwerken. Wenn man die, in Kapitel 4.3 bereits erwähnten Domänen eines Fahrzeugs betrachtet, kann man jeder mindestens ein eigenes Kommunikationsnetzwerk zuordnen. In der modernen Industrie sind diese miteinander verbunden, woraus sich eine Vielzahl an neuer Möglichkeiten für die Entwickler bieten. [30]
In den jeweiligen Netzwerken, befindet sich das "Network Level", welches die mittlere Abstraktionsschicht beschreibt. Jedes Netzwerk besteht aus mehreren ECUs, welche über verschiedene Kommunikationssysteme miteinander verbunden sind und wiederum Daten untereinander austauschen (Siehe Kapitel 4.6). [30]
Als "ECU-Level" bezeichnet man die unterste Ebene der Fahrzeugarchitektur, hier stehen die Hauptbestandteile der ECUs im Vordergrund. Eine ECU besteht aus einer oder mehreren Micro Controller Units (MCUs), welche über analoge und digitale Kommunikationsschnittstellen verfügt, sowie Communication Controllers (CCs) welche für die Kommunikation mit den Netzwerken zuständig ist. [30]

Betrachtet man die Softwarearchitektur, geht AUTOSAR einen sehr konsequenten Weg der strikten Trennung zwischen ASW und BSW. Die Funktionalitäten rund um das Betriebssystem, die Kommunikationsprotokolle und verschiedene Diagnose Module werden zur Gänze in der BSW implementiert. Währenddessen werden in der ASW ausschließlich Anwendungsbezogene, zum Beispiele das Auswerten von Sensoren, umgesetzt. Um den Verbindungsaufbau zwischen diesen beiden, voneinander unabhängigen Systemen, kümmert sich die Zwischenschicht RTE, wobei in dieser Schicht ebenfalls die Infrastruktur zur Kommunikation zwischen verschiedenen Softwarekomponenten bereitgestellt wird. [30]

4.5 AUTOSAR-Methodologie

Als weiteres Hauptthema der Entwicklungen dieses Konsortiums gilt die AUTOSAR-Methodologie, welche die Kernpunkte zur Entwicklung beschreibt, sowie ein Metamodell, welches zur genauen Standardisierung der Konzepte eingehalten werden muss, um ein abgeschlossenes, AUTOSAR konformes System zu erhalten. Obwohl es sich hierbei um keine komplette Prozessbeschreibung handelt, werden trotzdem alle Schritte in der Entwicklung bis hin zur Generierung von ausführbarem Programmcode durch die Definition von Datenaustauschformaten und standardisierten Methoden unterstützt. [26] Beschrieben wird ein modellbasierter Entwicklungsansatz, wobei die Hardware und Softwarearchitektur, so-

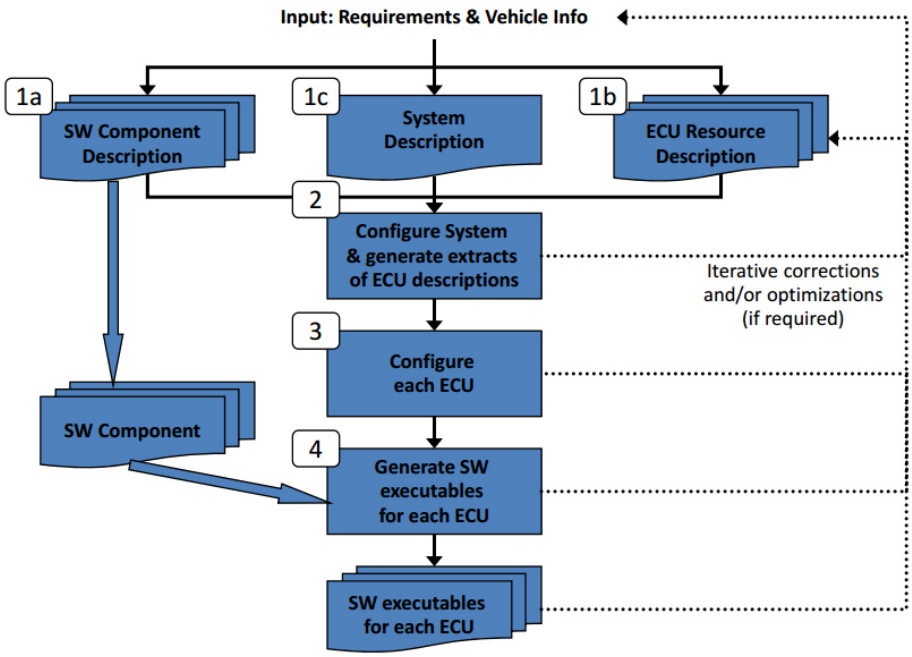

Abbildung 4.4: Der AUTOSAR-Prozess (Quelle: [15])

wie der Netzwerkaufbau formal modelliert wird. Das Metamodell unterstützt dabei den kompletten Prozess bis hin zur Integration und ist an die Meta Object Facility (OMG) angelehnt. [17] Hierbei handelt es sich weder um eine genaue Prozessbeschreibung, noch um ein Businessmodell. Daher gibt es keine genaue Reihenfolge in der die Aktivitäten abgehandelt werden sollen. Im Generellen lässt sich der Entwicklungsprozess in einzelne Schritte unterteilen. Im ersten Schritt werden notwendige Informationen zur Systemkonfiguration gesammelt, diese sind formale Definitionen von Softwarekomponenten, ECU-Hardwareressourcen und Netzwerk Topologien. Im Anschluss werden diese Definitionen dazu verwendet, die Softwarekomponenten der jeweiligen ECU zuzuordnen. Durch die Eingliederung in den Virtual Functional Bus (VFB) erfolgte die Modellierung bis dahin Hardware unabhängig. Die nächsten Schritte erfolgen nun direkt auf der jeweiligen ECU, daher werden alle nötigen Informationen zur weiteren Konfiguration aus der Systemdefinition extrahiert. Nach der Integration aller zusätzlich relevanter Konfigurationsdaten enthält die vollständige ECU-Konfiguration alles Nötige um ausführbaren Programmcode erzeugen zu können. [61]
Diese Definition eines AUTOSAR-Systems erfolgt anhand von Templates, welche eine formale Beschreibung erlauben und somit das Metamodell bezeichnen. [61]
Der genauere Ablauf eines AUTOSAR-Prozesses ist in Abbildung 4.4 dargestellt.
Durch die vielseitigen Möglichkeiten eignen sich diese Richtlinie hervorragend um auf Basis eines verteilten Entwicklungsprozesses Synergien zu nutzen und dadurch von einer erhöhten Effektivität zu profitieren. [26]

4.6 Schichtenmodell / Technisches Umsetzung

Die Zeit in der man unzählige einzelne Mikroprozessoren in einem Fahrzeug mit möglichst schlanker und in sich abgeschlossener Steuerungssoftware verwendet ist spätestens seit der Einführung des AUTOSAR-Standards Geschichte. Heute setzt man auf leistungsstarke ECUs und Kommunikation zwischen den Softwaremodulen. Die Entwicklungen, welche das AUTOSAR-Konsortium vorantrieb, führte zu Plattformunabhängigkeit und Portierbarkeit. [26]
Der Kern von AUTOSAR ist eine sehr gut strukturierte und standardisierte BSW, welche sich trennend zwischen die Hardware des Steuergeräts und der ASW stellt, wobei die Module sogar untereinander austauschbar sein sollen. Somit erreicht man eine absolute Entkoppelung von ASW und Steuergerät. [26] Dieses Ziel liegt dem Gedanken zugrunde, dass man jede beliebige ASW auf unterschiedliche Steuergeräte portieren kann. Dadurch dürfen sich, ohne Anpassungen vornehmen zu müssen, lediglich Latenzzeiten ändern jedoch keine funktionalen Unterschiede entstehen. [66]
Der VFB trennt die Applikationen, welche aus verschiedenen, miteinander verbundenen Softwarekomponenten bestehen können, von der darunterliegenden Infrastruktur. Er ist unabhängig von den miteinander verbundenen ECUs, welche zur Realisierung dieses Konzept allerdings über eine standardisierte BSW und entsprechend konfigurierte RTE verfügen müssen. [26] Im Gegensatz zum VFB, welcher die virtuelle Spezifikation der Kommunikation der einzelnen Komponenten liefert, repräsentiert die RTE, die tatsächliche Implementierung dieser virtuellen Konzepte für eine spezifische ECU. Die RTE ist außerdem eine Zwischenschicht, welche die strikte Trennung der ASW von der darunterliegenden BSW zur Folge hat. Somit ist der Grundstein für eine virtuelle Interaktion zwischen den Softwarekomponenten gelegt. [48]

Wie in Abbildung 4.5 dargestellt, baut die AUTOSAR-Architektur auf einem Schichtenmodell auf. Unter der RTE liegt nun die BSW, welche die zusätzlichen nicht funktionalen Services zur Verfügung stellt. [26] Um zwischen Hardware abhängigen und unabhängigen Funktionen unterscheiden zu können, teilt sich die BSW in die nachfolgenden drei Schichten. Von der obersten zur untersten Schicht wären dass "Service Layer", "ECU-Abstraction Layer" und "MicroController Abstraction Layer". Durch diese drei Schichten der BSW ziehen sich in horizontaler Linie vier Grundbausteine. Das sind Systemdienste, Speicherverwaltung, Kommunikationsdienste und Harware Ein-/Ausgabedienste. Als fünften Grundbaustein gibt es die sogenannten "Complex Drivers". Diese erlauben eine Loslösung vom herkömmlichen Dreischichtenmodell. Dieser Ansatz kann einerseits bei der Migration bereits existierender Module sehr nützlich sein, vor allem wenn es notwendig ist temporär direkte Schnittstellen zur RTE zu implementieren und andererseits bei extrem Zeitkritischen Peripheriekomponenten. Dies ist der Fall, wenn bestimmte Sensoren unter Einhaltung des Schichtenmodells über die vorgegebenen Reaktionszeiten kommen. [66]
Die oberste Schicht in der BSW, genannt "Service Layer", stellt verschiedene System Services zur Verfügung. Sie besteht aus dem Hardware abhängigen Betriebssystem, sowie

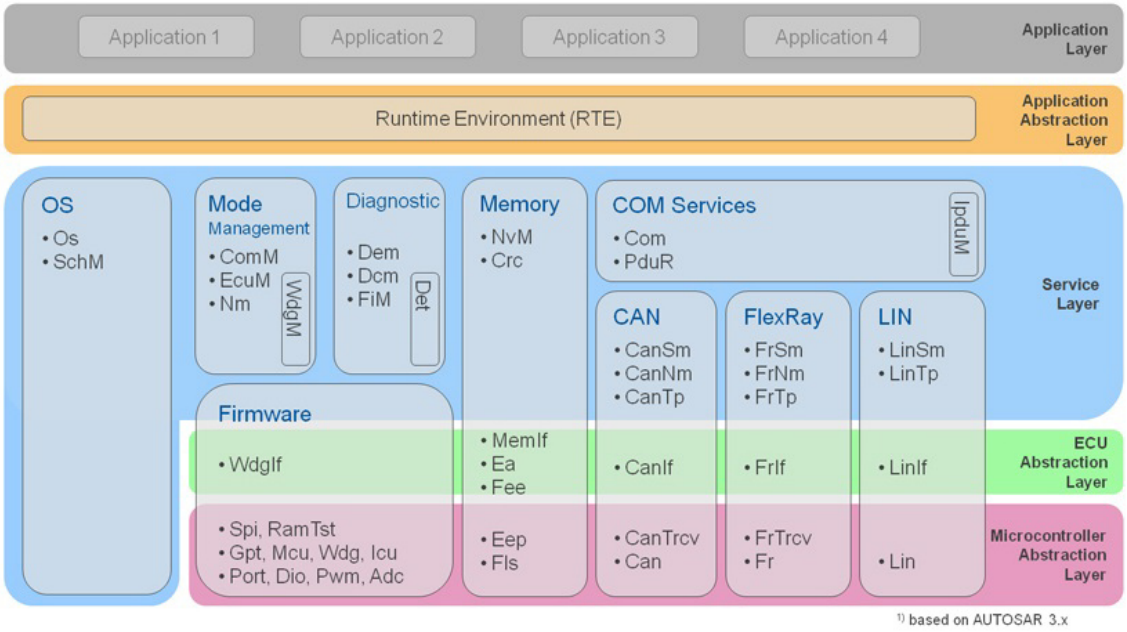

Abbildung 4.5: AUTOSAR-Schichtenmodell (Quelle: [47])

den von der Hardware unabhängigen Modulen Speicherverwaltung, ECU-Manager und den Diagnose Protokollen. [26]
Der darunter liegende "ECU-Abstraction Layer" ist zwar ECU spezifisch, aber dennoch unabhängig vom Mikrocontroller selbst. Seine Zuständigkeit ist die Abstraktion des ECU-Layouts von den darüber liegenden Komponenten. Die Unabhängigkeit wird durch den, vorhin bereits erwähnten, wiederum darunter eingeordneten MicroController Abstraction Layer (MCAL) erreicht. In dieser Schicht finden sämtliche spezifischen Mikrocontroller Treiber, welche durch die starke Hardwareabhängigkeit direkt vom jeweiligen Hersteller implementiert werden muss, wie zum Beispiel für die Kontrolle von digitaler und analoger Kommunikation sowie zur Steuerung des CAN-Ports, platz. [26]

Das AUTOSAR-Konzept erlaubt den verschiedenen BSW-Modulen mehrere unterschiedliche Ansätze der Konfiguration. Der "Pre-Compile"-Ansatz ist geeignet um die beste Laufzeitperformance zu bieten. Mit der "Post-Build"-Variante kann man auf erneutes Kompilieren nach Konfigurationsänderungen verzichten und genießt so erhöhte Flexibilität. Die "Link-Time" bildet den Mittelweg der beiden vorhergehenden Ansätze. [26]
Als Dateiformat wurde ARXML eingeführt. Mit diesem lassen sich alle BSW-Module parametrisieren und vollständig konfigurieren. Das hierfür gewählte Format musste vollkommen plattformunabhängig und jederzeit zwischen verschiedenen Visualisierungs- und Konfigurationstools sowie innerhalb der, in die Projektentwicklung eingebundenen,

Partner austauschbar sein. Dazu muss auch die Möglichkeit bestehen, einen generischen Editor auf die Datei anwenden zu können. [56]

Eine weitere Stärke des AUTOSAR-Ansatzes ist die unmissverständliche Fehlerbehandlung, welche eine durch alle Schichten, bis hinauf zur Applikationsschicht, durchgängige Fehlerkommunikation ermöglicht. Dies kann zum Beispiel für Diagnosezwecke, wie das erkennen defekter Kommunikationswege, eingesetzt werden. Sowohl für die RTE-Schicht sowie auch für BSW-Module sind eindeutige Vorgänge, welche beim Fehlerfall ausgeführt werden, vorgesehen. So werden zum Beispiel Fehlerevents ausgelöst, welche entweder direkt ausgelesen oder über das Development Error Tracer (DET) und Diagnostic Event Manager (DEM)-Modul überwacht werden können. Zur erhöhten Transparenz retournieren sämtliche API-Funktionen entsprechend gut dokumentierte Fehlernummern. Das DET-Modul ist entscheidend für die Entwicklungsphase. Speziell bei der Integration und Implementierung hat es die Funktion verschiedene Diagnoseevents zu speichern und zu melden. Das DEM-Modul ist hingegen entscheidend für den laufenden Betrieb, wo es für die Überwachung der regelmäßigen Events zuständig ist. [26]

4.6.1 Herausforderungen

Aufgrund der immensen Anzahl verschiedener Dateiformate und der damit einhergehenden Probleme der Kompatibilität und Austauschbarkeit, hat AUTOSAR einen weitgehend global akzeptierten Standard etablieren können. Dies führte zu einer neuen Dimension an Wiederverwertbarkeit und des verteilten Entwicklungsansatzes. [61]
Diese Tatsache ändert allerdings nichts an der erheblichen Menge an verschiedenen Konfigurationstools, welche nun zwar das selbe standardisierte Dateiformat verwenden, allerdings trotzdem das Risiko von Datenverlust beim Dateiaustausch zwischen verschiedenen Tools deutlich erhöhen. Um diesen negativen Umstand zu umgehen, sind in der Praxis verschiedene Generatoren, Importer und Exporter im Einsatz. [61]
Im Rahmen von AUTOSAR sind zwar sämtliche Standards und die Methodologie hervorragend definiert, was allerdings ausbleibt ist eine Definition Wer, Was, Wann im laufe des Softwareentwicklungsprozesses zu tun hat. Dies ist zwar ein eindeutig gewollter Umstand, um die Flexibilität der teilnehmenden Partner nicht zu schmälern, kann jedoch in der Praxis zu erhöhtem Kommunikationsaufwand führen. [61]
Zum jetzigen Zeitpunkt gibt es drei Hauptreleases und mehrere Nebenreleases, welche mitunter eingeschränkt kompatibel zueinander sind. Diese Gegebenheit kann zu erhöhtem Definitionsaufwand innerhalb eines Nutzernetzwerkes führen. [3, 61] Zusätzlich werden die vollen Sicherheitsstandards erst ab Version 4.0 umgesetzt, was ein erhöhtes Risiko beim Entwickeln auf Basis des, noch immer weit verbreiteten, 3.x Releases mit sich bringt. [51]

4.7 "AUTOSAR-OS"

Da das "OSEK-OS"-Betriebssystem die Basis für das "AUTOSAR-OS" liefert, ist es unter anderem abwärtskompatibel und implementiert die selben API-Aufrufe. [66]
Es sind zwei Arten von Tasks definiert. Einerseits Basistasks, welche nicht von Systemaufrufen blockiert werden können und sequentiell abgearbeitet werden, und andererseits erweiterte Tasks, welche aus mehreren miteinander synchronisierten Blocks bestehen, die in einen wartenden Zustand fallen können. [32] Im Gegensatz zu OSEK, wo zeitgesteuerte Abläufe mit Alarmen getriggert werden müssen, definiert AUTOSAR die Scheduletabellen auf Hardware oder Softwarezählern basierend. Dadurch werden Tasks einfacher zu bestimmten vordefinierten Zeiten, auf Basis eines Zählerstandes, in den "ready state" versetzt. Ist dieser Zustand gesetzt, hängt die tatsächliche Ausführungszeit immer noch von den Prioritäten dieser und der anderen Tasks ab. [66]
Einen weiten Bereich des Betriebssystems stellt die Interrupt Service Routine (ISR) dar, welche auch weitgehend vom OSEK-Standard übernommen wurde. An dieser Stelle gibt es zwei Kategorien von ISR. Eine welche den Aufruf bestimmter OS-Services erlaubt und eine wo dies zur Gänze verboten ist. Der Aufruf dieser wird von externen Events getriggert, dabei ist die Priorität eines ISR immer größer als die der verschiedenen Tasks am Betriebssystem. Je nach verwendeter Hardware und Konzept, kann er jedoch von anderen aktivierten ISR unterbrochen werden. Da es im Scheduling wenig Unterschied macht, ob es sich um einen ISR oder herkömmlichen Task handelt und der OSEK-Standard einen Ressourcenaustausch zwischen diesen zulässt, ist es, je nach Konzeption, möglich aus einem ISR heraus Tasks zu aktivieren. [32] Um die Zuverlässigkeit zu erhöhen gibt es außerdem verschiedene Überwachungsmechanismen. Für normale Tasks gibt es Überprüfungen, damit der maximale Stackbereich nicht überschritten wird. Um den meist zeitkritischen Anforderungen einer ISR nachzukommen, kann eine Zeitüberwachung implementiert werden, womit die Ausführungszeit und maximale Ausführungszeit sowie die Aufrufhäufigkeit überwacht werden. Im Fehlerfall kommt es jeweils zum Aufruf einer Funktion namens "Protection Hook", welche das weitere Vorgehen beschreibt und vom Betriebssystem bereitgestellt werden muss. [66]
AUTOSAR hat auch mehrere Konzepte von "Protected OS" übernommen. Da wäre zum Beispiel das Zusammenfassen zu Anwendungsgruppen der sämtlichen Betriebssystemobjekte, welche zur Laufzeit im engen Zusammenspiel arbeiten, wie etwa Tasks, Alarme, verschiedene Ressourcen, Events und Zähler. Zur Überprüfung der korrekten Funktion während der Laufzeit, wird ein, im AUTOSAR-Standard so genannter, "Service Protection"-Task verwendet. Für besonders zeitkritische Anwendungsgruppen, wie zum Beispiel die Auswertung von Sensoren, welche in sehr kurzen Zeitabständen Messwerte liefern, gibt es nun die Möglichkeit die gesamte Gruppe in einen "Trusted Application"-Status zu versetzen, bei dem diese Sicherheitsüberprüfungen aus Performance gründen deaktiviert werden. [66] Das gesamte Anwendungsgruppendesign dient vor allem beim MC-Ansatz, wie in Kapitel 4.9 genauer beschrieben, als Grundlage für eine reibungslose Funktion.

Auch beim Scheduling lehnt sich AUTOSAR stark an OSEK an, erweitert jedoch das Konzept hinsichtlich der bereits erwähnten "Timing Protection" erheblich. Den einzelnen Tasks werden in der Konfigurationsphase bereits entsprechende Ausführungsprioritäten zugeteilt. Kommen während der Laufzeit mehrere gleich priorisierte Tasks zusammen, werden diese nach dem First In – First Out (FIFO)-Prinzip abgearbeitet. Das Scheduling findet entweder anhand der einzelnen Tasks, oder der, bereits vorhin angesprochenen Anwendungsgruppen statt. Letztere können präemptiv, nicht präemptiv oder gemischt präemptiv konfiguriert sein, wobei diese Eigenschaften somit auf Taskebene definiert werden müssen. Eine wesentliche Stärke weist AUTOSAR mit dem "Timing Protection"-Service auf, welches das Auftreten von Fehlern bezüglich der Maximalen Ausführungszeit, das Blockieren von Ressourcen sowie die Wiederholungsrate von Tasks oder einer ISR kontrolliert. [32] In der AUTOSAR-Softwarearchitektur ist das Betriebssystem für das gesamte Scheduling, inklusive dem der BSW, zuständig. [66]
Die Verarbeitung verteilter Ressourcen ist ein weiterer wichtiger Punkt im AUTOSAR-Betriebssystem. Es regelt den gleichzeitigen Zugriff auf eine Ressource mit Hilfe des OSEK-Priority Ceiling Protocol (PCP). Dieses arbeitet so, dass ein Task welcher auf eine Ressource zugreift sofort die Ressourcenpriorität übernimmt. Somit erhalten andere Tasks, welche währenddessen auf diese Ressource zugreifen wollen, keine Zeit auf der CPU. [32]
Periodische Tasks werden mit Hilfe von Zählern, Alarmen, sowie mit dem bereits erwähnten Scheduletabellen Konzept umgesetzt. Das Zusammenspiel funktioniert so, dass ein Hardware- oder Softwarezähler bei einem bestimmten, vorher definierten Zählerstand, einen Alarm auslöst, welcher dann wiederum einen Task oder ein Event triggert. AUTOSAR definiert eine globale Netzwerkzeit für die Ausführung von Scheduletabellen, wodurch lokale Zeitnehmer zyklisch synchronisiert werden müssen. [32]
In "AUTOSAR-OS 2.0" gibt es noch die Möglichkeit einer weitgehend automatischen Konfiguration mit Hilfe einer entsprechenden OIL-Datei. Für diesen Zweck ist allerdings seit Release 3.0 der Extensible Markup Language (XML)-Standrad Pflicht. AUTOSAR definiert selbst sein Einsatzgebiet und verweist daher in der Spezifikation, beispielsweise für die Implementierung eines Navigationssystems, auf andere Betriebssysteme wie "Embedded Linux". In diesem Fall sieht "AUTOSAR-OS" einen "AUTOSAR-OS Abstraction Layer" vor, damit AUTOSAR kompatible Funktionen einfach auf das System portierbar sind. [66]

4.8 AUTOSAR und Safety

Die Anzahl an sicherheitskritischen Systemen nimmt stetig zu. Immer mehr Softwaremodule werden auf immer weniger, dafür jedoch ungleich leistungsfähigeren, ECUs verteilt und sollen effizient miteinander kommunizieren. So kommt es auch dazu, dass auf einer ECU unterschiedlich sicherheitskritische Software ausgeführt wird. Hierzu benötigt man wiederum spezielle Sicherheitssysteme. In diesem Fall wäre die Hauptaufgabe, dass

ein Fehler in einem weniger sicherheitskritischen System keinerlei negative Auswirkungen auf ein Sicherheitskritisches nimmt (Siehe Kapitel 4.8.1). [27] Das Betriebssystem "AUTOSAR-OS" selbst verfügt über Mechanismen zur Überwachung von Speicherzugriffen und der Stackauslastung sowie der Ausführungsdauer von Softwareelementen. Es werden auch die gängigen Signalüberwachungsroutinen implementiert und die verbundenen Hardwarekomponenten mit entsprechenden Überprüfungen zyklisch auf Fehlerfälle getestet. Zur zusätzlichen Absicherung der Datenübertragung wird die "E2E-Protection Library" empfohlen. Diese soll die Datenübertragung auf allen Ebenen durch die Implementierung eines Sequenzzählers, von Prüfsummen, einer Timeouterkennung, einem Nachrichtenzähler sowie unterschiedlich berechneten Identifikator (ID)-Nummer kontrollieren. [66]
In der Automotive Industrie gibt es unterschiedlichste Arten von Datenübertragungsfehlern, sei es nun ein Datenverlust, multiples Empfangen der selben Nachricht, eine Zeitverzögerung, ein Timeout oder gar ein manipulierter Datensatz. Für diese Art von Fehlern sieht AUTOSAR drei unterschiedliche Konzepte zur Problemlösung vor. Hier wäre einerseits die bereits erwähnte "E2E-Protection Library" und andererseits zwei verschiedene Ansätze, welche sich das Communication (COM)-Modul, die Basis der Datenübertragung selbst, zu Nutze machen. Der einfachere Ansatz bezüglich des COM-Moduls, ist die Realisierung eines Zählers, welcher diesen beim Senden und Empfangen von Nachrichten aktualisiert und vergleicht. Da das COM-Modul unabhängig vom darunter liegenden Bussystem ist, lässt sich allerdings auch ein zweiter, redundanter Kommunikationspfad zur Überprüfung implementieren. [51]

Aufgrund der extrem unterschiedlich priorisierten Tasks in einem System, kommt bei AUTOSAR-Systemen präemptives Scheduling zum Einsatz. Das bedeutet, dass Tasks nur einen bestimmten, definierten Zeitraum auf der CPU zur Verfügung haben und unterbrochen werden können. Für die Automotive Domäne bedeutet dies, dass höher priorisierte Tasks bei Bedarf andere Tasks unterbrechen können, um ihre vorgegebene Laufzeit einhalten zu können. Um dieses System effizienter und weniger fehleranfälliger zu machen, stellt AUTOSAR ein Service bereit, welches sich "Timing Protection" nennt. Dessen Aufgaben sind die Überwachung von Ausführungszeiten der Tasks, Ankunftszeiten von Events und der Sperrzeit von verteilten Ressourcen. Ist diese Funktion in einem System implementiert, behebt man damit einen Großteil der möglichen Timing Probleme und eröffnet sich eine neue Dimension an Schedulingmöglichkeiten. Der wesentliche Vorteil ist, dass dieses Service die maximalen Ausführungszeiten berücksichtigt und wie in Abbildung 4.6 ersichtlich, in den Schedulingvorgang einarbeitet. Was passiert wenn es dennoch zu einer Überschreitung der Ausführungszeit kommt, lässt sich laut AUTOSAR genau definieren. Im Normalfall wird der Task beendet, oder die gesamte Applikation oder auch die ECU neu gestartet. Somit sind die Sicherheitsvorschriften der ISO 26262 (Siehe Kapitel 4.8.1), in Bezug auf "absence of error propagation", erfüllt. [27]
Um höchste Sicherheitsstandards und Zertifizierungen zu ermöglichen integrierte man die ISO 26262 in Phase 2 der AUTOSAR-Entwicklung. [29]

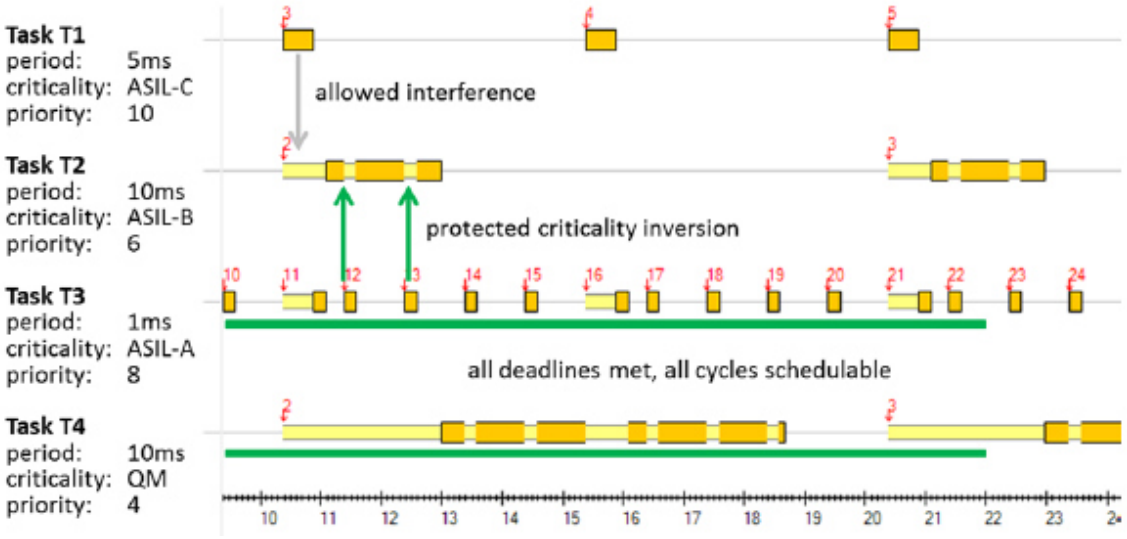

Abbildung 4.6: Scheduling mit "Timing Protection"-Service (Quelle: [27])

4.8.1 Definition von Safety im Automotiven Kontext

Die ISO 26262 stellt heute eine der wichtigsten Sicherheitsnormen in der Automotive Industrie dar und ist eine Adaption von IEC 61508 um die Bedürfnisse dieser Branche zu erfüllen. Dabei bestehen die nachfolgenden zehn Parts, welche unter dem namen "Road vehicles - Functional safety" zusammengefasst sind. [36]

- Part 1: Vokabular
- Part 2: Management von funktionaler Sicherheit
- Part 3: Konzeptphase
- Part 4: Produktentwicklung auf Systemebene
- Part 5: Produktentwicklung auf Hardwareebene
- Part 6: Produktentwicklung auf Softwareebene
- Part 7: Produktion und Betrieb
- Part 8: Unterstützende Prozesse
- Part 9: ASIL orientierte und sicherheitsorientierte Analysen
- Part 10: Informative Guideline [36]

Auch in Bezug auf Elektrisch/Elektronisch (E/E)-Systeme stellt die ISO 26262 fundamentale Richtlinien zur Verfügung, auf die man sich in der AUTOSAR konformen Entwicklung stützen muss. Da die funktionale Sicherheit bereits im Entwicklungsprozess wesentlich geprägt wird, wurde ein Automotive Sicherheitslebenszyklus definiert, beginnend von Management, über Entwicklung, Produktion, Service bis zur Außerbetriebnahme. Weiter

existiert ein Automotive spezifischer Risikoansatz zur Entscheidung des passenden ASIL-Levels. Außerdem werden Voraussetzungen für die Validierung und für die Beziehung zum Lieferanten geliefert. [36]
Beispiele für extrem wichtige, AUTOSAR bezogene Definitionen wären zum Beispiel der Grundsatz, dass eine sicherheitskritische Softwarekomponente nicht durch eine unkritische negativ beeinflusst wird, was den ISO 26262 Begriffen "freedom from interference" und "error propagation" zugrunde liegt. Ausserdem gibt diese Norm den "Watchdog Manager" als wesentlichen Bestandteil des funktionalen Sicherheitskonzepts an. [27]

4.9 Mehrkernansatz

Die Hauptgründe für den Einsatz von MC-Architekturen im Automotive Bereich sind Zuverlässigkeit, Performance und Kosten. Genauer gesagt sind die zwei Hauptursachen, die Reduktion vieler kleinerer ECUs zu einer gemeinsamen und die Implementierung unzähliger neuer performancehungriger Anwendungen. Der Punkt an dem diese Bedürfnisse effizient mit einer simplen Taktsteigerung erreicht werden können, ist aufgrund von Energieverbrauch und Hitzeentwicklung, mittlerweile überschritten. [37]
Ein MC-Prozessor ist eine Kombination von mindestens zwei unabhängigen Central Processing Units (CPUs) auf einem einzelnen Chip. Wobei jede CPU typischerweise einen eigenen Level 1 Cache besitzt und sich einen Level 2 Cache mit den anderen Kernen teilt (Siehe Abbildung 4.7). Generell gibt es zwei Möglichkeiten für das Scheduling in einer MC-Architektur. Erstens globales Scheduling, hier werden sämtliche Aufgaben in einer einzelnen Kette abgearbeitet, und zweitens, verteiltes Scheduling, hier besitzt jeder Kern seine eigene Abarbeitungskette zu denen die Aufgaben statisch zugeordnet sind. In der Praxis gibt es auch hybride Scheduling Ansätze für MC-Architekturen. [50]
In der Domäne der Desktopcomputer hat sich der MC-Ansatz längst etabliert. Es existieren praktische keine Einkernarchitekturen mehr im Endkundenbereich. Der Grund, warum dieser Schritt im Automotive Bereich noch nicht vollzogen ist, liegt allerdings an den unterschiedlichen Voraussetzungen, denn man steht hier einer harten Echtzeitanforderung und extrem geringen Fehlertoleranz gegenüber. Das Problem liegt nicht bei der durchschnittlichen Ausführungszeit die ein Task benötigt, sondern bei der Worst Case Execution Time (WCET), welche in dieser Domäne im Ernstfall das Leben der Insassen gefährden kann, wenn sie die maximale Ausführungszeittoleranz überschreitet. Eine Timing Analyse ist bereits im Einkernbereich eine Herausforderung, nimmt jedoch trotzdem in der MC-Architektur eine ungleich größere Dimension an. Dies kommt zum Tragen wenn das MC-System nicht im Lockstep Modus betrieben wird, wo vereinfacht gesagt, die selben Operationen auf mehreren Kernen parallel zur Kontrolle ausgeführt werden. Man bezeichnet dies als "Inter-Core"-Parallelismus. Um die verschiedenen Kerne untereinander zu verbinden, benötigt man Bussysteme, Netzwerke, verteilte Ressourcen und dynamisch geroutete Kommunikationsstrukturen. [38]

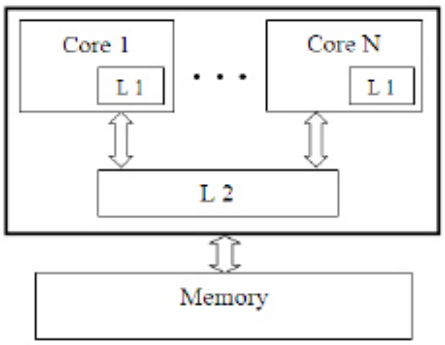

Abbildung 4.7: Mehrkernarchitektur (Quelle: [50])

Wie bereits erwähnt, unterstützt AUTOSAR den MC-Ansatz ab dem Release 4.0. Als Kerndefinition gibt AUTOSAR die Verwendung von verschiedenen Anwendungsgruppen vor, wo verschieden Betriebssystemobjekte wie Zähler, Tasks und Scheduletabellen zusammengefasst und in der Konfiguration statisch an einen Kern gebunden werden. Abbildung 4.8 zeigt die Verteilung von Tasks auf bereits angesprochene Anwendungsgruppen, welche unterschiedlichen Kernen zugewiesen werden. Hier ist auch zu sehen, dass das AUTOSAR-Konzept eine eigene Instanz des Betriebssystems auf jedem Kern vorsieht. [39]
Bezüglich der Softwareverteilung auf die verfügbaren CPUs, setzt AUTOSAR voraus, dass der überwiegende Anteil an BSW auf einem einzigen Kern ausgeführt wird, welcher Hauptkern genannt wird. [39] Die Verteilung wird statisch in der Konfigurationsphase durchgeführt, eine Lastenverteilung während der Laufzeit zwischen den CPUs ist dabei nicht vorgesehen. Dieser Ansatz knüpft an das vorhin erklärte verteilte Scheduling an. Nach dem Hochfahren müssen sich die parallel laufenden Kerne einmalig synchronisieren und das Scheduling gleichzeitig starten. Ab diesem Zeitpunkt werden die jeweiligen Tasks parallel auf den zugewiesenen Kernen abgearbeitet und können mit klassischen Funktionsaufrufen auch Tasks oder Events, nicht jedoch Ressourcen, auf anderen CPUs aufrufen. Zur Synchronisation werden "Spinlocks" bereitgestellt, diese verfügen über einen blockierenden sowie nicht blockierenden Aufruf. Als abschließenden Punkt ist zu erwähnen, dass es für die verschiedenen Softwarekomponenten der Anwendungsebene, bezüglich des Datenaustauschs, keinen Unterschied macht auf welchem Kern sie liegen. Dieser erfolgt mit Hilfe der Inter-OS-Application Communication (IOC)-Funktionen über einen gemeinsamen Speicherbereich. [66]

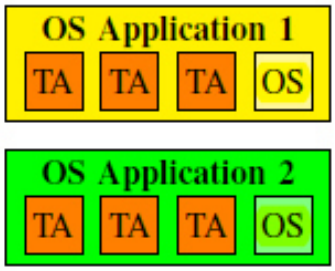

Abbildung 4.8: Anwendungsgruppen auf zwei CPUs verteilt (Quelle: [39])

4.10 AUTOSAR im Überblick

Die Sinnhaftigkeit des AUTOSAR-Standards wird durch den enormen Zustrom an führenden Unternehmen der Automotive Industrie eindeutig belegt. Die Einteilung in Module und Schichten unterstützt die Austauschbarkeit der Elemente großteils und standardisiert Schnittstellen, Komponenten sowie Methodiken. Daraus ergeben sich die Vorteile von Wiederverwendbarkeit und besserer Integrierbarkeit, innerhalb der einzelnen Partner und auch der Projekte. AUTOSAR deckt zwar noch nicht jeden Bereich der Fahrzeugentwicklung ab, so wie es auch im Infotainment Segment noch Schwächen gibt, jedoch werden mit jedem Release die Spezifikationen verfeinert und erweitert. Abzusehen ist, dass in den nächsten Jahren die kontinuierliche Umstellung auf AUTOSAR konforme Systeme fortgeführt wird. Die Existenz und ständige Weiterentwicklung des AUTOSAR-Standards wird maßgeblich dazu beitragen, dass sich die Entwicklung in Zukunft immer stärker auf Endverbraucherbedürfnisse fokussieren kann. [66]

Kapitel 5

Konzeptentwicklung

Das erstellte Konzept soll eine möglichst einfache und effektive Integration von nicht vollständig AUTOSAR konformer ASW, in ein AUTOSAR konformes Projekttemplate ermöglichen. Zusätzlich soll es möglich sein, die jeweilige ASW auf verschiedene Kerne einer möglichen Mehrkernplattform intelligent zu verteilen. Dazu ist es notwendig eine Toolchain zu definieren, automatische Konfigurationsdateien zu erstellen und die wesentlichen BSW-Module vor zu konfigurieren. Bevor die erstellte Struktur an einem realen Kundenprojekt umgesetzt werden kann, soll sie durch einen Demonstrator getestet werden.
Der modulare Aufbau von AUTOSAR-Projekten ermöglicht diesen Ansatz, setzt allerdings eine strikte Trennung von ASW und BSW voraus. Dies wird vor allem durch den Einsatz einer RTE-Schicht ermöglicht. In der Praxis sind allerdings nicht alle potentiell zu integrierenden Anwendungsprojekte von der Hardware entkoppelt, was einen erhöhten Migrationsaufwand darstellt. Ein wesentlicher Teil bei der Migration von Kundenprojekten stellt auch die Signalkonfiguration, sei es von CAN, Pulsweitenmodulation (PWM) oder Analog-Digital Konverter (ADC)-Signalen, dar. Um eine effektive Integration in die AUTOSAR-Umgebung zu ermöglichen ist die Entwicklung eines eigenen Generators unumgänglich. Dieser Generator muss vorher beschriebene Konfigurationsdateien aus DBC-Dateien und schriftlichen Signaldefinitionen erstellen. Die so entstehende ARXML-Datei mit den zugehörigen Makro Definitionen wird in die RTE importiert.
Eine Übersicht über das erstellte Konzept ist in Abbildung 5.1 ersichtlich. Die Nummerierungen in der Grafik beziehen sich auf die Kapitel in denen eine genauere Beschreibung erfolgt.

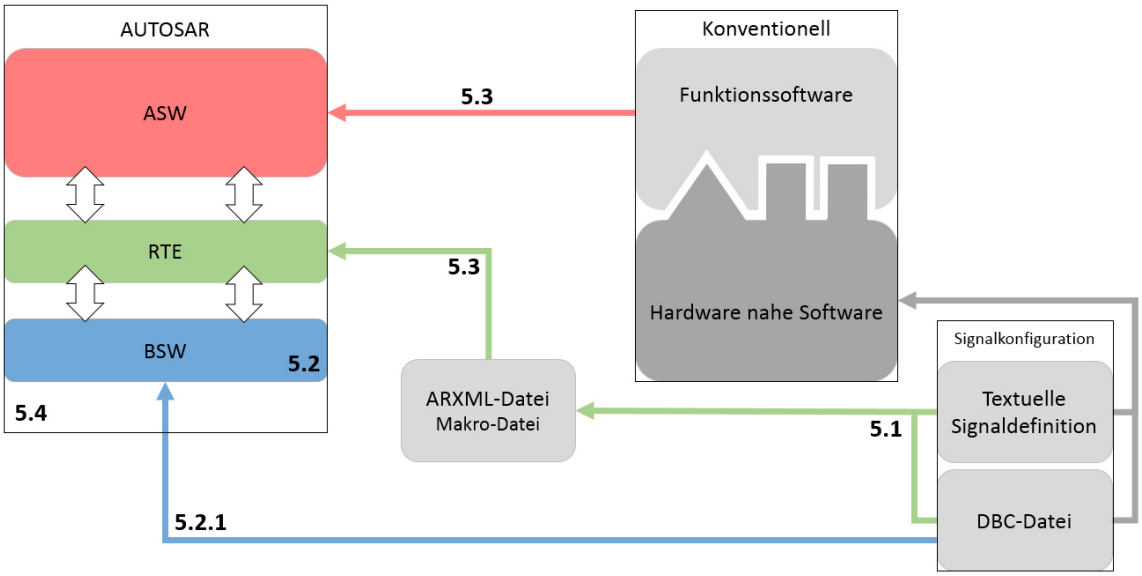

Abbildung 5.1: Konzeptüberblick

5.1 AUTOSAR XML (ARXML)-Generator

AUTOSAR bietet den ersten verbreiteten Ansatz, der den gesamten Arbeitsablauf des Entwicklungsprozesses von eingebetteter Automotive Software abbildet. Da viele Industrieteilnehmer auf modellbasierte Toolunterstützung und Code Generatoren setzen, ist die Portierbarkeit der Konfigurationen während der Softwareerstellung, sowie die anschließende Wiederverwendung klar definiert. [33]
Die Konfigurationsdateien, welche im AUTOSAR-Kontext verwendet werden, sind standardisiert und werden als ARXML-Dateien bezeichnet. Um einen eigenen Generator zur automatischen Konfiguration von verschiedenen Signalen sowie teilen der RTE erstellen zu können, ist es wichtig einen Überblick über diese Spezifikation zu erhalten.

XML ist eine hervorragende Struktur zur Sicherung und Weitergabe von Konfigurationsdaten und Definitionen. Daher greift auch AUTOSAR auf diesen Ansatz zurück. Der Vorteil liegt in der textbasierten Speicherung von Daten, welche sowohl von Maschinen als auch von Menschen einfach ausgelesen und bearbeitet werden können. [62] In einer AUTOSAR-Toolchain muss dieses Dateiformat quer durch alle relevanten Programme, welche an der Erstellung der Definitionen und Konfigurationen beteiligt sind, unterstützt werden. Die Option, eine OIL-Datei zu verwenden, ist in der aktuellen AUTOSAR-Version nicht mehr zulässig. Aus diesem Grund wurde im Zuge der fortlaufenden Definitionen, der ARXML-Standard definiert. Dieser wird sowohl bei der Beschreibung von Softwarekomponenten, Interfaces, BSW-Modulen als auch bei der RTE als Konfigurationsstandard umgesetzt. In einer AUTOSAR konformen Entwicklung wird das komplette Systemdesign,

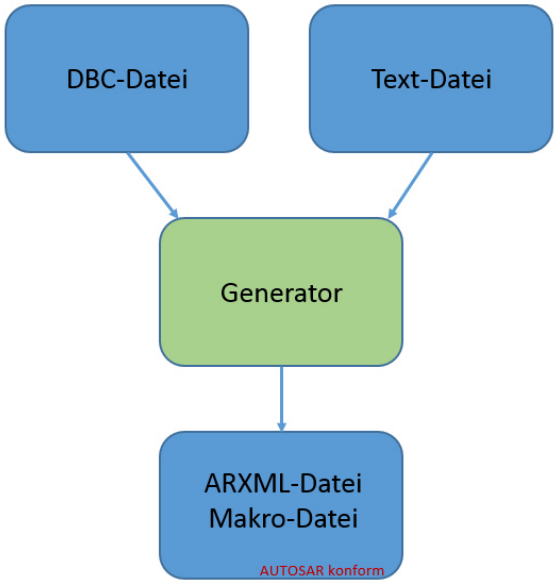

Abbildung 5.2: Konzept des ARXML-Generators für Signale

in Form verschiedener ARXML-Dateien welche die ECU beschreiben, gespeichert. Daraus ergibt sich der große Vorteil, dass sämtliche Informationen, welche im Zuge der verschiedenen Schritte im Entwicklungsprozess entstehen, gespeichert werden. Dies führt zu enormer Flexibilität, Skalierbarkeit und Wiederverwertbarkeit von AUTOSAR konformen Softwaremodulen. Am Markt bedeutet das, die Möglichkeit den Entwicklungsprozess zu standardisieren, Module von unterschiedlichen Zulieferern beziehen zu können und den Informationsaustausch erheblich zu vereinfachen. [62]
Trotz aller Spezifikation steht diese Standardisierung in der Praxis nach wie vor häufig vor dem Problem der Kompatibilität. Ein Datenverlust oder eine Fehlinterpretation bei der Portierung zwischen den am Markt verfügbaren Tools ist, trotz eines einheitlichen Datenaustauschformats, ein präsentes Problem. Hauptursachen hierfür könnten verschiedene Interpretationen der Tools sowie das Fehlen bestimmter, für das jeweilige Tool nicht relevanter, Interpretationen sein. Die relativ hohe Frequenz, in der neue Versionen des Standards veröffentlicht werden und die somit einhergehende parallele Existenz mehrerer verschiedener ARXML-Versionen in der Industrie, ist eine weitere Ursache für das Portierbarkeitsproblem. [33]

Das entstandene Konzept des Generators zur automatischen Generierung von AUTOSAR konformen sowie Toolchain kompatiblen Signalkonfigurationsdateien, welche die automatische Konfiguration der RTE unterstützen, ist in Abbildung 5.2 ersichtlich. Es handelt sich um eine flexible und leicht erweiterbare Software, welche vorhandene DBC-Dateien und textuelle Signaldefinitionen ausliest. Im zweiten Programmschritt wird aus den eingelesenen Werten eine ARXML-Datei, mit zugehörigen Makrodefinitionen, generiert.

5.2 Verwendete Basis-Software (BSW)-Module

Um ein AUTOSAR konformes Projekttemplate aufsetzen zu können, ist die Kontrolle über sämtliche Schichten der BSW sowie der RTE-Schicht unausweichlich. In einem von vornherein AUTOSAR konformen Projekt hält sich der Konfigurationsaufwand aufgrund der vorhandenen ARXML-Dateien weitgehend in Grenzen. In dieser Arbeit wird ein Konzept für eine relativ einfache und dennoch effektive Integration von nicht AUTOSAR konformer ASW in ein vordefiniertes AUTOSAR konformes Projekttemplate entwickelt. Dieser Ansatz hat zur Folge, dass man sich intensiv mit den verwendeten BSW-Modulen auseinandersetzen muss, um diese ohne Konfigurationsdateien möglichst generisch konfigurieren zu können. Welche Module hierbei näher betrachtet werden müssen, ist in Abbildung 4.5 ersichtlich. Im Rahmen dieser Arbeit wurden allerdings nicht alle abgebildeten Module konfiguriert, da zur Evaluierung und ersten beispielhaften Integration nur eine begrenzte Anzahl, wie zum Beispiel das FlexRay, LIN oder Diagnose Modul, notwendig sind. Es gibt einige wesentliche Module, welche für eine entsprechende Funktion des Projekttemplates jedoch unbedingt erforderlich sind. In der MCAL sind die Module CAN, MCU, PORT, DIO, PWM und ADC zu konfigurieren. In Bezug auf die Kommunikationsschnittstellen wird in der "ECU Abstraction Layer" das Controller Area Network Interface (CanIf)-Modul implementiert. In der Serviceschicht befindet sich das "AUTOSAR-OS", welches zur Nutzung auf Mehrkernprozessoren konfiguriert wird, sowie das Mode Management und die Module COM und PDU Router (PduR).
Diese Beschreibung zum Einsatz der BSW-Module ergibt sich aus den klaren AUTOSAR-Definitionen bezüglich der Modulunabhängigkeit, Austauschbarkeit und dem Zugriff über standardisierte Modulinterfaces. Dies führt dazu, dass sämtliche BSW-Funktionen, welche in konventionellen Systemen oft tief in der ASW verwaltet wurden, strikt voneinander getrennt werden.

Bei der Konfiguration der BSW beginnt man in der Hardware abhängigen Schicht. Zuerst konfiguriert man Hardwaretreiber und anschließend die Abstraktionsmodule. Die Treibermodule werden vom Hardwarehersteller geliefert und die Module der Abstraktionsschicht vom Steuergerätehersteller. Trotz der Standardisierung kann es hier zu ersten Problemen kommen. Aus diesem Grund schreiben die Fahrzeughersteller ihren Zulieferern sehr häufig vor, generische Basissoftwarepakete von spezialisierten Softwarefirmen zu verwenden. Im Zuge der Standardisierung der Hardwaretreiber lehnt sich AUTOSAR weitgehend an die Konzepte des HIS-Konsortiums an, wobei die API-Schnittstellen dennoch nicht direkt kompatibel sind. [66]
Fast alle Treiber sind für die synchrone Ausführung konfiguriert, führen somit ihre Aufgaben in einem Zug vollständig durch und retournieren das Ergebnis oder den Fehler. Um den zeitkritischen Anwendungsfällen gerecht werden zu können, ist auch ein Unterbrechen von anderen Tasks vorgesehen. Eine Ausnahme bilden hier zum Beispiel der ADC-Treiber, welcher aufgrund seiner zeitaufwendigen Berechnungen von vornherein asynchron ausgeführt wird. In diesem Fall werden die Ergebnisse entweder über "Polling",

einer separaten Ergebnisabfragefunktion oder über eine Rückruffunktion, welche nach Beendigung der Berechnungen, eine entsprechende Funktion zur Ergebnisweitergabe aufruft, vermittelt. Etwaige Berechnungsfehler oder Kommunikationsschwierigkeiten werden meist automatisch an das DEM-Modul, welches eine Art zentralen Fehlerspeicher darstellt, gesendet. Jeder Hardwaretreiber besitzt die "ModulName_Init()"-Funktion zur Konfiguration und Initialisierung, sowie die Deinitialisierungsfunktion "ModulName_DeInit()". [66]
Nachfolgend werden die, im Rahmen dieser Arbeit, wesentlichen Module der MCAL und "ECU-Abstraction Layer" beschrieben.

- Micro Controller Unit (MCU)-Modul

  Nach der Initialisierung der CPU wird der, nicht durch AUTOSAR standardisierte, "Startup Code" des Mikrocontrollers aufgerufen. Dieser startet die Taktgeber, ist für Systemresets zuständig und initialisiert Betriebsmodi sowie die Speicherbereiche. Ein Teil dieses Moduls ist der Generic Timer Module (GTM)-Treiber, welcher die Grundlage für das PWM-Modul bereitstellt und für die Initialisierung der "Clock" sowie der Multiplexer Register verantwortlich ist. Außerdem ist der Interrupt Request Module (IRQ)-Treiber im MCU-Modul enthalten. Dieser stellt die notwendigen Konfigurationsparameter und API-Funktionen bezüglich des Interrupt-Handlings zur Verfügung. [10, 66]

- General Purpose Timer (GPT)-Modul

  Der GPT-Treiber basiert auf der Peripherie des GTM-Treibers. Mit Hilfe dieses Treibers, werden einmalige, sowie periodische Zeitgeber und dazugehörige Timeouts konfiguriert. Es werden verschiedene API-Funktionen zum Starten und Stoppen der Zeitgeber, sowie zum Abruf der Rückruffunktion bereitgestellt. [9, 66]

- PORT-Modul

  Um die Funktionsfähigkeit des Digital Input Output (DIO)-Moduls zu gewährleisten, muss zuerst das PORT Modul vollständig initialisiert und konfiguriert sein. Hier werden zum Beispiel die Transferrichtung, sowie der Transfermodus aller Ports und Port-Pins des Mikrocontrollers definiert. Das DET-Modul steht unter anderem auch in direktem Kontakt mit dem PORT Treiber. [13, 66]

- Digital Input Output (DIO)-Modul

  Diese Komponente stellt die API-Funktionen zur Verfügung, um das Lesen und Schreiben auf einzelne digitale Schnittstellen zu ermöglichen. Die Anforderungen der jeweiligen ASIL-Standards werden erfüllt, indem die notwendigen Sicherheitsmechanismen implementiert werden. Konkret ermöglicht der DIO-Treiber einen ungepufferten Lese- und Schreibzugriff auf einzelne Input/Output (I/O)-Schnittstellen. Hier wird zwischen drei Arten von Schnittstellen unterschieden. Ein "DIO Channel" symbolisiert einen einzelnen digitalen Pin, welcher vom PORT Treiber definiert wurde. Unter einem "DIO Port" versteht man eine Hardwareseitige Gruppierung von "DIO Channels". Die Struktur der "DIO

Channel Groups" ist ein logischer Zusammenschluss von "DIO Channels" innerhalb eines "DIO Ports". [8, 66]

- Pulsweitenmodulation (PWM)-Modul

  Um PWM-Signale konfigurieren und senden zu können, wird in diesem Modul das Tastverhältnis sowie die Frequenz des PWM-Ausgangssignals konfiguriert. Es existieren API-Funktionen zum Lesen und Setzen der Ausgangssignale, sowie eine Rückruffunktion welche aufgerufen wird, wenn sich die Flanke eines PWM-Ausgangs ändert. Dabei wird der PWM-Ausgang über den Timer Output Module (TOM)-Channel des GTM-Treibers generiert. Ein PWM-Signal wird über die Parameter "Duty Cycle" sowie die "Period" definiert. [14, 66]

- Analog-Digital Konverter (ADC)-Modul

  Dieses Modul dient zum Messen von einzelnen oder gruppierten analogen Eingangssignalen. Hier kann abermals zwischen einer Einzelmessung und einer kontinuierlichen Messung unterschieden werden. Um das Abtasten zu aktivieren kann man entweder einen Softwareaufruf, einen Zeitgeber oder ein Hardwaresignal verwenden. Da dieses Modul asynchron konfiguriert wird, stellt es die Messergebnisse mit Hilfe einer Rückruffunktion dar. Um eine korrekte Funktionsweise dieses Moduls zu gewährleisten, müssen zunächst einige der bereits genannten Hardware Treibermodule konfiguriert werden. Zuerst muss der Systemzeitgeber im MCU-Modul definiert sein. Die zu überwachenden Ports müssen im PORT Modul als Eingangsports konfiguriert sein. Der GTM-Treiber muss entsprechend aufgesetzt sein, um periodische Konvertierung zu unterstützen. Hier ist Vorsicht geboten, da unter anderem auch das PWM-Modul gleichzeitig auf diese Zeitgeber zugreift. [5, 66]

- Controller Area Network (CAN)-Modul

  Dieses Treibermodul stellt Services zum Zugriff auf die CAN-Hardwarekomponente zur Verfügung. In AUTOSAR besteht eine solche Komponente aus von einander unabhängigen CAN-Controllern, welche sich zum Schreiben und Lesen eine bestimmte Anzahl an Nachrichtenobjekten teilen. In der Softwarekonfiguration eines jeden verbundenen Controllers, müssen Einstellungen bezüglich der Baudrate, der Zeitgeber sowie auch der Auslösemechanismus für das Senden oder Empfangen definiert werden. Jeder Controller kann über mehrere Nachrichtenobjekte, meist eine für die Input und eine für die Output Signale, verfügen. In diesen Nachrichtenobjekten wird jede eingehende sowie ausgehende Nachricht gespeichert. Von dort aus wird sie initial gelesen oder übertragen. Die Definition einer Filtermaske bietet dabei die Möglichkeit die empfangenen Daten zu begrenzen. Das Absetzen sowie Empfangen von Nachrichten funktioniert entweder durch zyklische Aufrufe, welche am Treiber kalibriert werden können, oder durch "Polling". [6, 66]
  Aufgrund der Wichtigkeit dieses Standards, innerhalb dieser Arbeit, sind ausführliche Details zum CAN-Bus im nachfolgenden Kapitel 5.2.1 beschrieben.

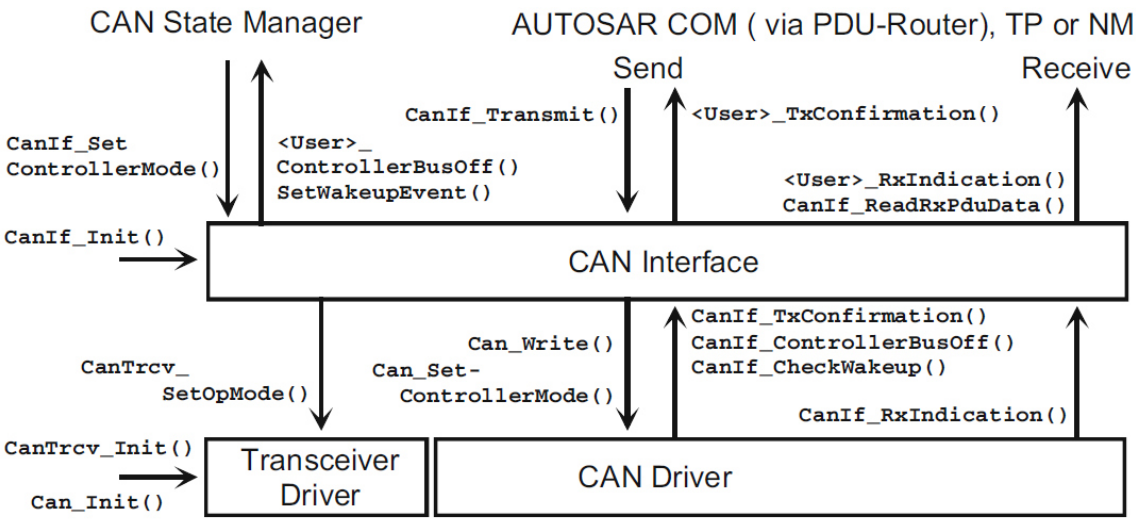

Abbildung 5.3: CanIf-Kommunikationsstruktur (Quelle: [66])

- Controller Area Network Interface (CanIf)-Modul

  Dieses Modul ist das einzige der "ECU-Abstraction Layer"-Module, welches im Rahmen der Diplomarbeit aktiv verwendet wurde. Es ist das Abstraktionsmodul zum CAN-Hardwaretreiber. Innerhalb des erwähnten Moduls existieren zwei zentrale, synchron implementierte API-Funktionen. "CanIf_Transmit()" sendet eine Nachricht direkt an das CAN-Modul und "CanIf_ReadRxPduData()" ist für das Lesen verantwortlich. Ist kein Sendespeicher frei, wird automatisch gepuffert und zu einem späteren Zeitpunkt gesendet. Wenn eine neue Nachricht vom CAN-Modul empfangen wurde, wird dies durch die Methode "CanIf_RxIndication()" mitgeteilt. Das genauen Zusammenspiel der Funktionsaufrufe ist in Abbildung 5.3 ersichtlich. Die übergeordneten Module, welche später in diesem Kapitel noch beschrieben werden, können von nun an über die vorhin beschriebene Lesefunktion auf die Daten zugreifen. Neben den Sende- und Empfangspuffern werden auch noch Akzeptanzfilter und die dazugehörigen Protocol Data Units (PDUs) definiert. Eine Protocol Data Unit (PDU) legt die Struktur einer einzelnen, bestimmten Nachricht fest und verfügt über eine Identifikationsnummer, nach welcher gefiltert werden kann, sowie einer Datenlänge. In den oberen Schichten wird solch eine Nachricht zur Verarbeitung in weitere Signale aufgesplittet. [66]

Wenn die Konfiguration der beiden unteren Schichten der BSW abgeschlossen ist, muss die "Service Layer" konfiguriert werden. Um ein funktionierendes AUTOSAR konformes System zu erhalten. In dieser Schicht beschäftigt sich diese Arbeit vor allem mit den Modulen rund um das Betriebssystem. Dies umfasst den ECU State Manager (EcuM),

COM und PduR, für die AUTOSAR konforme Umsetzung der CAN-Kommunikation.

- Operating System (OS)-Modul

  Das "AUTOSAR-OS" wird in dieser Diplomarbeit verwendet und die genaue Entstehung, sowie der Aufbau wurde bereits im Kapitel 4.7 näher erläutert. Aus diesem Grund wird nachfolgend lediglich auf verschiedene Konfigurationsmöglichkeiten sowie technische Details eingegangen. In der Praxis erfolgt in diesem Softwaremodul die Konfiguration einer Vielzahl an wesentlichen Komponenten, welche für die geregelte Ausführung von zeitkritischen Tasks in einer Mehrkernumgebung zuständig sind.
  Dieses Betriebssystemmodul unterstützt verschiedene Ausführungsmodi. Dabei ist zu beachten, dass an dieser Stelle immer ein OSEK kompatibler "OSDEFAULTAPPMODE" definiert sein muss. Ergänzend ist es möglich andere Modi, zum Beispiel für erhöhte Sicherheit oder reduzierte Funktionalität, zu definieren. Hierfür werden die zu aktivierenden Alarme, Events, Tasks, Verhaltensweisen etc. in der Modus Definition zusammengefasst und konfiguriert. Für eine gültige Konfiguration ist es notwendig, als nächsten Schritt, geeignete Software oder Hardwarezähler zu definieren. Diese werden auf Basis der verfügbaren Hardwarekomponenten erstellt und erhalten somit unter anderem ein Interrupt Level sowie ihre Tickgeschwindigkeit. Auf Basis dieser Zähler werden sogenannte Alarme erstellt, mit deren Hilfe sich Events, Tasks oder Rückruffunktionen triggern sowie die vorher definierten Zähler manipulieren lassen. Nachdem nun die Grundlage für entsprechende Tasks und Events gelegt ist, werden diese definiert. Eine wichtige Task Konfiguration, stellen die verschiedenen Initialisierungstasks für andere Hardware und Softwarekomponenten dar. Erst danach reihen sich die ASW spezifischen Task Definitionen ein. Generell verfügt ein Task über eine Prioritätseigenschaft, Konfigurationen zum Scheduling, Autostartoptionen und Mechanismen zum Garant der Ausführungszeit. Um eine mehrkernfähige Konfiguration zu erhalten, sind Konfigurationen von sogenannten Anwendungsgruppen vorzunehmen. In diesen Gruppen werden Alarme, Tasks, Events, ect. logisch zusammengefasst und anschließend ohne Umstände auf die verschiedenen Kerne verteilen. Die Möglichkeit der Anwendungsgruppen wurde bereits in Kapitel 4.9 genauer erläutert. Um einen reibungslosen Ablauf der verschiedenen, zeitkritischen Tasks gewährleisten zu können existiert außerdem eine Spinlock Funktionalität sowie das IOC-Modul.
  Natürlich existieren noch eine Unzahl weiterer Konfigurationsoptionen, wie etwa innerhalb der Ressourcenverteilung, verschiedene Scheduletabellen, Interrupt Konfigurationen, Diagnosemechanismen sowie Sicherheitsfunktionen. Da die genaue Evaluierung des AUTOSAR-Betriebssystems nicht im Fokus dieser Arbeit liegt, konzentrieren sich die Erklärungen vordergründig auf Abweichungen vom Standardmodell. [11]

- ECU State Manager (EcuM)-Modul

  Dieses Modul ist für das Management der verschiedenen ECU-Zustände verant-

wortlich, wobei in jeder Phase bestimmte Abläufe eingehalten werden müssen. Die Hauptaufgaben sind die Initialisierung und Deinitialisierung des Betriebssystems, des BSW-Schedulings, des Basis-Software Mode Manager (BswM) sowie der Treibermodule. Die Funktionsroutine teilt sich in fünf verschiedene Phasen.
In der "STARTUP"-Phase, welche sich in die Pre-OS sowie Post-OS Sequenz unterteilt, wird der ECU-Start kontrolliert, die BSW-Module initialisiert, Konsistenz überprüft, Interrupt Prioritäten aufgesetzt, Parameter für das Herunterfahren gesetzt, das Betriebssystem gestartet, Semaphoren initialisiert und der BswM initialisiert. Die "UP"-Phase nutzt hauptsächlich die "EcuM_MainFunction", welche periodisch ausgeführt wird und die Aufwachfunktionen sowie verschiedene Zeitgeber kontrolliert. Als "SLEEP"-Phase bezeichnet man den Energiesparmodus der ECU. In dieser Phase wird kein Programmcode ausgeführt; es wird auf ein Signal zum Aufwachen gewartet. In der "SHUTDOWN"-Phase wird ein kontrolliertes Herunterfahren eingeleitet, welche schlussendlich in die "OFF"-Phase übergeht. [18]

- PDU Router (PduR)-Modul

  Hier werden Services für das Routing von Interaction Layer Protocol Data Units (I-PDUs) bereitgestellt. Bei einer Interaction Layer Protocol Data Unit (I-PDU) kann es sich entweder um ein Kommunikationsinterface Modul (z.B.: COM, LinIf, CanIf) oder ein Transportprotokoll Modul (z.B.: LinTp, CanTp, FrTp) handeln. Das Routing findet nur anhand von, vor der Laufzeit konfigurierten, statischen Pfaden statt. Ein dynamischer Ansatz ist nicht vorgesehen. Dieses Modul ist generisch einsetzbar. Die Liste an kompatiblen Modulen ist nicht beschränkt. Es lassen sich zusätzlich sowohl Quell- wie auch Zielmodule definieren. Es besteht aus zwei Teilen, den Routingtabellen, wo die statischen Routingpfade definiert werden und der Router Engine, welche die eigentliche Weiterleitung übernimmt. [12]
  In Abbildung 5.4 wird die Funktionsweise, sowie das Zusammenspiel von Routingtabellen und Router Engine dargestellt.

- Communication (COM)-Modul

  Diese Komponente liegt zwischen dem PduR-Modul und der RTE. Die Hauptfunktion liegt im Verpacken der PDUs, sowie in der Bereitstellung der benötigten Interfaces. Wie das PduR-Modul, wird auch das COM-Modul unabhängig vom darunterliegenden Kommunikationsprotokoll verwendet. Im Zuge dieser Hauptaufgabe sind verschiedene Funktionen von Bedeutung, welche unter anderem die Sicherheit, Kompatibilität und Speicherung der Ereignisse unterstützen. [7]
  In der Praxis ist der Hauptkonfigurationsaufwand dieses Moduls, die Erstellung sowie Verlinkung der verschiedenen Signale mit den entsprechenden PDUs.

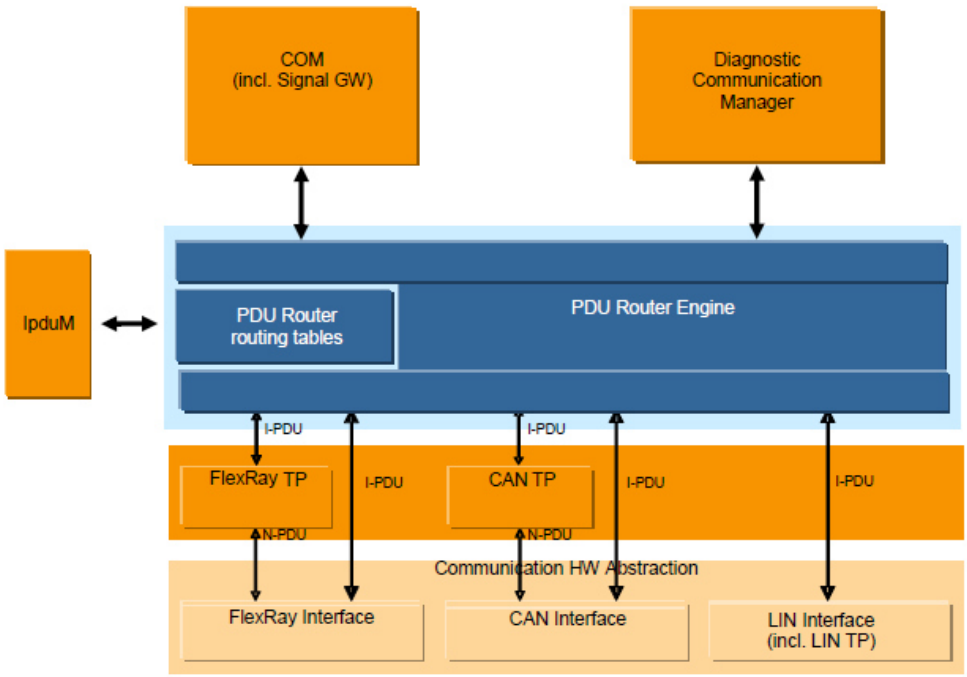

Abbildung 5.4: PduR-Struktur zur Buskommunikation (Quelle: [12])

Als übergeordnete Schicht zur BSW spielt die RTE bei der Konfiguration eines AUTOSAR konformen Systems eine zentrale Rolle. Denn die RTE entkoppelt die ASW von der BSW. Im Kontext dieser Arbeit steht vor allem die automatische Kommunikationssignal-Konfiguration der RTE im Fokus.
Nähere Details zur RTE können aus Kapitel 4 entnommen werden.

5.2.1 Controller Area Network (CAN)

Wie bereits erwähnt, befasst sich diese Arbeit in Bezug auf die Buskommunikation ausschließlich mit dem CAN-Bus. Daher wird nachfolgend näher auf diesen Standard eingegangen. Der Grund für diese Entscheidung liegt in der weiten Verbreitung des Standards, der guten Performance sowie den sehr geringen Kosten. [45]

Heutzutage ist CAN aus der Automotive Industrie nicht mehr weg zu denken. Dieser Kommunikationsstandard hat seinen Weg mittlerweile in unzählige weitere Branchen gemacht und zählt generell zu den wichtigsten Kommunikationsprotokollen in der Industrie. Die Entwicklung wurde von Robert Bosch um 1980 initiiert. Ziel war es ein effizientes Protokoll zur Kommunikation der verschiedenen Fahrzeugkomponenten zu schaffen, sowie die Länge der Kabelstränge in einem Fahrzeug zu minimieren. Mittlerweile sind unzählige Modifikationen und Weiterentwicklung am Markt verfügbar. [25]

Die Ursache für die Effizienz des CAN-Standards liegt in dessen technischen Details. Das Bussystem verfügt über ein serielles Interface. Je nach Länge des Kommunikationsweges sind Übertragungsraten von bis zu 1 Mbit/s möglich. Im Vergleich zu anderen Bussystemen, welche oftmals technisch aufwendige Koaxialkabel benötigen, kommt ein CAN-Netzwerk mit simplen, vergleichsweise billigen "Twisted Pair"-Kabeln aus. Im Gegensatz zu anderen Kommunikationssystemen wird mit sehr niedrigen Datenlängen gesendet, wodurch sich eine relativ geringe Latenzzeit ergibt. Einen Broadcast oder Multicast abzusetzen ist in CAN genauso einfach, wie eine simple Kommunikation zwischen zwei einzelnen Knoten. Natürlich sind auch entsprechende Sicherheitsmechanismen sowie Fehlererfassungs- und Korrekturstrategien Hardwareseitig implementiert. [25]
In einem CAN-Kommunikationsnetzwerk findet weder ein Verbindungsaufbau zwischen den Knoten, noch ein direktes Adressieren der Nachrichten statt. Jede Nachricht erhält eine Identifikationsnummer und wird vom CAN-Controller in Form eines "Frames" als Broadcast abgesetzt. Anschließend können alle anderen Knoten im Netzwerk, welche auf diese Identifikationsnummer konfiguriert sind, den Datenframe empfangen. Dies ermöglicht die Kommunikation mit mehreren Fahrzeugkomponenten mit nur einer Nachricht. Ein weiterer Vorteil ist, dass die Nachrichten nicht zyklisch abgesetzt werden müssen, sondern ein Event basiertes Senden möglich ist. [25]

5.3 Migration von bestehender Software

Der vollkommene Umstieg auf AUTOSAR konforme Ansätze kann nicht in einem Zug vollzogen werden. Dazu gibt es in der Praxis zu viele existierende Systemkomponenten. Das genaue Vorgehen hängt von internen Businessplänen sowie vorhandener Kapazitäten ab. Es wäre zu kostspielig und zeitaufwendig, bestehende Systeme von Grund auf neu als AUTOSAR konforme Systeme zu entwickeln. Daher begnügt man sich zum Beispiel mit der Migration bestehender Systeme oder damit neue Komponenten nach dem AUTOSAR-Standard zu entwickeln. Auf der technischen Ebene bedeutet dies, dass man älteren Systemen ein neues Interface aufsetzt. [28]
In den verschiedenen Migrationsphasen gibt es mehrere Herausforderungen. Um eine bestehende ASW AUTOSAR konform zu machen, müssen Softwareunterteilungen erfolgen, Ports und Events definiert und RTE konforme Interfaces aufgesetzt werden. Zusätzlich gilt es noch die bestehenden Performance Voraussetzungen zu erfüllen. Die BSW ist, aufgrund der strikten Standardisierung, eine noch größere Herausforderung bei der Migration. Daher ist eine Neuentwicklung oder die Nutzung bestehender BSW-Module am Markt meist effizienter. Eine weitere Schwierigkeit könnte die Kommunikation zwischen der neu integrierten standardkonformen ECU und anderen Netzwerkkomponenten, welchen ein unterschiedliches Netzwerkmanagement zu Grunde liegt, sein. [41]
Das Zerlegen des bestehenden Systems ist ein sehr guter Ansatz für eine erfolgreiche Transformation. Man versucht eine Unterteilung in ASW, Sensorsteuerung, Hardwareabstraktionen und andere BSW-Module zu finden, sowie vorhandene Teile, welche im

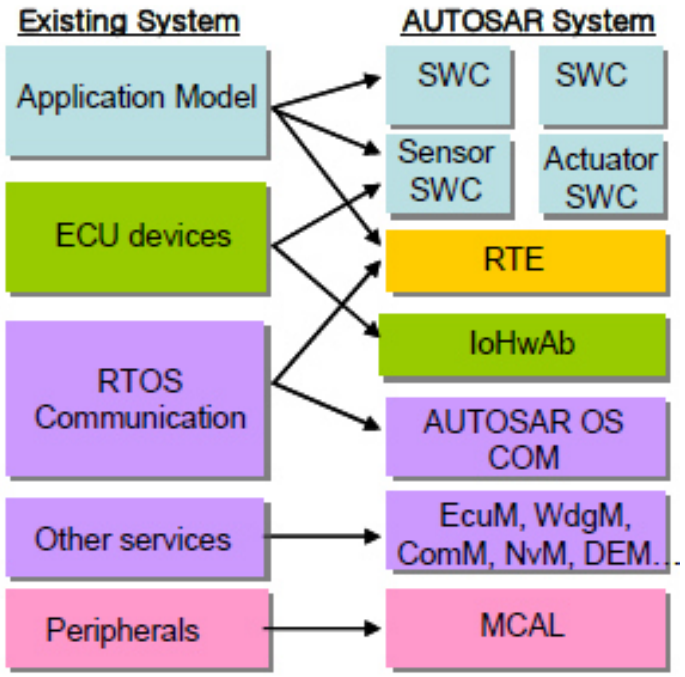

Abbildung 5.5: AUTOSAR-Migration (Quelle: [41])

neuen System nutzlos wären, zu entfernen. Eine exemplarische Darstellung für das eben beschriebene Mappen zu standardisierten AUTOSAR-Schichten und Modulen wird in Abbildung 5.5 dargestellt. Ein wesentlicher Schritt ist die Umsetzung konformer Interfaces, sowie auch der darauf aufbauende VFB, welcher in Kapitel 4.6 erklärt ist. Wenn diese Interfaces definiert sind, muss die RTE entsprechend konfiguriert werden. Unter Anderem werden hier Kommunikationssignale zugewiesen, Tasks und Events erstellt sowie Prioritäten zugeteilt. Bereits erstellte Kommunikationsmatrizen können meist ohne große Änderungen in das neue System übernommen werden. [41]

Der Ansatz der Unterteilung, welcher in diesem Kapitel beschrieben wird, wird auch im ASW-Integrationskonzept dieser Arbeit umgesetzt. Zur erfolgreichen Integration der beiden Demonstratoren in die AUTOSAR konforme Projektumgebung, muss das bisherige System vorerst in verschiedene Teile untergliedert werden. Für die spätere Systemkonfiguration werden zunächst alle relevanten Informationen bezüglich der BSW extrahiert. Die vorhandenen Signalkonfigurationen der Bussysteme werden ebenfalls entfernt und durch den, zu beginn dieses Kapitels beschriebenen, ARXML-Generator in ein kompatibles Dateiaustauschformat geladen. Zu diesem Zeitpunkt verfügt man über eine ASW-Komponente, welche zwar keine BSW spezifischen Informationen mehr enthält, jedoch auch noch nicht mit dem System verknüpft ist. Darum sollte der nächste Schritt die komplette Konfiguration der BSW sowie der Kommunikationsstrukturen sein. Dies kann entweder durch vorhandene Konfigurationsdateien, oder händische Eingaben

erfolgen. Durch das Importieren der bereits generierten ARXML-Kommunikationsdatei und konfigurieren der benötigten Tasks, Events, etc. erhält die RTE schlussendlich sämtliche Interfaces. Diese werden von der ASW für den vollständigen Funktionsumfang benötigt. Um die Integration abzuschließen, muss die ASW noch mit den erstellten Interfaces der RTE verknüpft werden.

5.4 Konzept des ersten Demonstrators

Die Entwicklung einer AUTOSAR konformen Mehrkernprojektumgebung beschäftigt sich großteils mit der Konzeption einer Struktur, welche das Einpflegen von älteren, bereits vor AUTOSAR umgesetzten Projekten erleichtert und effizient gestaltet. Um diese Struktur erstellen zu können, ist es wesentlich alle beteiligten Tools und Systemmodule zu verstehen und einen Arbeitsablauf zu Entwickeln. Im Zuge dieser Aufgabe ist eine funktionsfähige Grundkonfiguration der wesentlichen AUTOSAR BSW-Module zu erstellen. Die Komponenten, welche im Fokus dieser Arbeit liegen, wurden im Kapitel 5.2 bereits näher erläutert. Außerdem ist es notwendig die RTE-Schicht entsprechend vorzubereiten und mit der BSW zu verknüpfen. Aufgrund der Verschiedenartigkeit der zu migrierenden Projekte und deren Konfigurationsdokumentation, erleichtert ein Generator zur Erstellung von standardisierten Konfigurationsdateien für die automatische Kommunikationskonfiguration die Arbeit erheblich.
Bevor das erste größere Kundenprojekt eingepflegt wird, sollte das grundlegende Projekttemplate und der anzuwendende Migrationsablauf evaluiert werden. Um den grundlegenden Teil des Projekttemplates und die notwendigen Migrationsschritte zu evaluieren wird ein erster Demonstrator konzeptioniert. Dieses erste Migrationsprojekt umfasst die Grundfunktionen eines funktionierenden Systems. Inhaltlich gesehen, wird die Funktionsfähigkeit des Betriebssystem-Konfigurationsansatzes, des Task- und Eventmanagements, sowie der Kommunikationsstruktur bezüglich einfacher digitaler Ports sowie dem CAN-Bussystem getestet. Nach erfolgreichem Testen des Konzepts, werden anhand dieses Demonstrators bereits erste Konfigurationen, hinsichtlich einer Mehrkernimplementierung, umgesetzt und evaluiert.
Die in Abbildung 5.6 positionierten Komponenten, welche aus einem "AURIX TC 275 Board", einem handelsüblichen Schalter mit drei Positionen, einem Fahrzeugpedal sowie einer Drosselklappe bestehen, werden auf einem mobilen Versuchsaufbau montiert. Weiters ist in der Abbildung der genaue Aufbau der Kommunikationsstruktur des Demonstrators im AUTOSAR-Schichtenmodell mit roten Linien eingezeichnet.
In der obersten Schicht ist die existierende ASW, welche zuerst eingehende Signale von einem Pedal verarbeitet und anschließend entsprechende Werte für die Stellung einer Drosselklappe, sowie den Status der Blinklichter berechnet und abschickt. Die Positionsübertragung des Pedals erfolgt analog. Dazu wird es direkt an einen Eingangspin am Board gehängt. Dieser Pin wird mit dem PORT Modul entsprechend konfiguriert und an das ADC-Modul weiter gegeben. Im ADC-Modul findet eine Umwandlung in die tatsächlichen Dezimalwerte statt, welche während der Berechnung periodisch in eine vorkonfigurierte

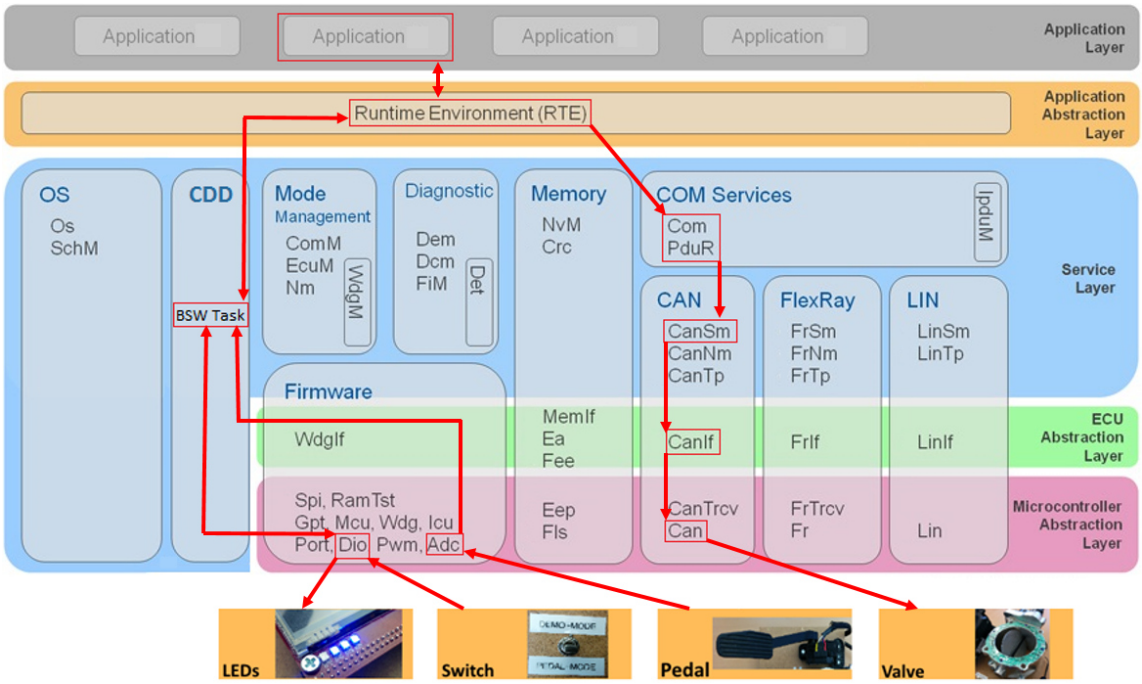

Abbildung 5.6: Kommunikationsaufbau des ersten Demonstrators

Inter Runnable Variable (IRV) in der RTE-Schicht zwischengespeichert werden. Die ASW kann diesen Wert anschließend mit dem entsprechenden RTE-Funktionsaufruf weiter verwenden. Sobald die neue Position der Drosselklappe berechnet ist, wird der neue Wert über einen RTE-Funktionsaufruf nach unten gegeben. Die RTE gibt den Positionswert an das COM-Modul weiter, welches diesen in das zugehörige Signal verpackt. Anschließend wird die gesamte betroffene PDU an den PduR weitergegeben. Von diesem Modul aus könnte die PDU nicht nur als CAN sondern auch als FlexRay, Local Interconnected Network (LIN) oder ETHERNET Nachricht weitergegeben werden. Eine Weiterleitung an das CAN-Modul, welches die Signalgruppe an die Drosselklappe überträgt, findet daher im PduR-Modul statt. Der Zustandswert des Schalters wird vom DIO-Modul direkt an eine IRV in der RTE weitergegeben. Dort wiederum hat die ASW einen lesenden Funktionsaufruf zur Verfügung. Genau gegengleich funktioniert die Ansteuerung der Leuchtdioden am Board, um den Gaspedal-Prozentwert anzuzeigen. Der Schalter erlaubt es zwischen dem manuellen und dem automatischen Modus umzuschalten. In diesem Automatikmodus wird die Stellung der Drosselklappe in einer Mehrkernumgebung berechnet.
Da sämtliche Modulkonfigurationen ohnehin vom jeweiligen Gesamtsystem abhängig sind, werden alle Module, welche in diesem Kontext nicht erwähnt sind, zu Testzwecke weitestgehend mit Standardwerten konfiguriert.

Kapitel 6

Softwareimplementierung

Auf Basis der bisher erarbeiteten Theorie rund um die Automotive Industrie, den verschiedenen Standards sowie dem entwickelten Konzept, wird nun die daraus entstandene Softwareimplementierung beschrieben. Zunächst wird ein Überblick über die verwendeten Entwicklungswerkzeuge gegeben und die Anforderungen an die Implementierung genauer beschrieben. Anschließend setzt sich diese Arbeit mit der technischen Umsetzung des entstanden Projekttemplates auseinander. In diesem Kontext wird der generelle Aufbau, die verwendeten BSW-Module und dessen Initialisierung sowie die Anbindung zu Hardware und RTE aufgearbeitet. Es wird auch auf den entwickelten ARXML-Generator, sowie dem integrierten Demonstrator eingegangen. Weitere Punkte dieses Kapitels stellen die Demonstrator Integration, der anschließende Funktionstest der Implementierung sowie das integrierte Kundenprojekt dar.

6.1 Entwicklungsumgebung, Werkzeuge und Komponenten

In der heutigen Zeit bildet die Grundlage zur effizienten Softwareentwicklung eine effiziente Toolchain sowie der Einsatz einer anwendungsgerechten Integrated Development Environment (IDE). Diese Arbeit beschäftigt sich vor allem mit der Entwicklung einer effizienten AUTOSAR-Projektumgebung, wodurch die Auswahl der Entwicklungswerkzeuge an erster Stelle steht. Hierbei ist zu beachten, dass ein Mehrkernprojekttemplate entstehen soll, mit welchem die Integration von nicht AUTOSAR konformer ASW effizient gestaltet werden kann. Für diese Anforderungen gibt es eine hohe Anzahl an Alternativen. Da am Ende dieser Arbeit allerdings die Evaluierung im Hinblick auf die Serienproduktion im Fokus steht, wird eine zertifizierbare, für die industrielle Serienproduktion einsetzbare AUTOSAR-Entwicklungsumgebung eingesetzt. Die dafür benötigten Werkzeuge umfassen eine IDE zur Konfiguration und Integration der verschiedenen AUTOSAR BSW-Module, einen Compiler, welcher eine mit dem Mikrocontroller kompatible Binärdatei erstellen kann, ein Mikrocontroller, welcher den aktuellen Standards und Leistungskriterien der Automotive Industrie entspricht, ein System zum Flashen und Debuggen sowie ein System zum Testen der CAN-Konfiguration.
Nachfolgend werden die verwendeten Werkzeuge und Komponenten detailliert beschrieben.

- Elektrobit Tresos Studio [23]

  Die Konfigurationsoptionen der AUTOSAR-BSW sind in unzähligen Dokumentationen detailliert beschrieben. Diese umfassen Spezifikationen zu über 40 unterschiedlich komplexen Modulen, welche über Interfaces untereinander vernetzt sind und für eine funktionierende Kommunikation entsprechend konfiguriert werden müssen. Mit etwa 100.000 konfigurierbaren Parametern, welche wiederum bis zu 5 unterschiedliche Einstellungen zulassen, ist eine gute Dokumentation allein nicht ausreichend, um einen effizienten Entwicklungsprozess zu unterstützen. Für diesen Zweck wird nicht nur ein gutes Tool, welches dem Entwickler die effiziente Konfiguration der unterschiedlichen Module ermöglicht, sondern auch ein Werkzeug benötigt. Jenes Werkzeug dient als Unterstützung um passende Parameter zu finden und Fehlermeldungen zu erkennen. Der Grund dafür sind die starken Abhängigkeiten zwischen den Konfigurationen der einzelnen Module in den verschiedenen Schichten. Um nun zu verhindern, dass ein AUTOSAR-Entwickler alle Details einer gültigen Konfiguration auswendig kennen muss, kann die eingesetzte IDE dafür sorgen, dass nur untereinander kompatible Kombinationen von Parametern konfiguriert werden können. Daraus ergibt sich der große Vorteil, dass zu jeder Zeit im Entwicklungsprozess nur semantisch korrekte Konfigurationen erstellt werden können. Der Umstand, dass keine fehlerhaften Konfigurationen aufgespürt werden müssen, die BSW-Codegröße vom Tool so gering wie möglich gehalten wird und die Kommunikation zwischen den Modulen, aufgrund der Standardisierungen, von vornherein reibungslos funktioniert, führt zu einer enormen Steigerung der Entwicklungseffizienz. [19]
  Es ist selbstverständlich, dass ein Entwicklungswerkzeugs, welches die soeben beschriebenen Funktionen und Möglichkeiten unterstützt, über ein wesentlich höhere Komplexitätslevel verfügt als ein normaler Texteditor. Was sich in der Anfangsphase als erheblichen Mehraufwand im Projekts äußert, ist im späteren Verlauf einer Serienproduktion die Grundlage für ein skalierbares, flexibles, effizientes und vor allem fehlerfreies System.
  Im Zuge dieser Arbeit wird das "Elektrobit Tresos Studio" als Entwicklungswerkzeug genutzt. Es besteht aus einer Eclipse basierenden, grafischen Entwicklungsumgebung welche die Möglichkeit besitzt, eigene Erweiterungswerkzeuge einzubinden. Diese IDE ermöglicht es, jedes BSW-Modul einer ECU in einem einzigen Programm zu konfigurieren, validieren sowie generieren. [19]
  In Abbildung 6.1 ist eine detaillierte Übersicht über die Systementwicklung mit "Elektrobit Tresos Studio" ersichtlich. Veranschaulicht wird vor allem die automatische Konfiguration der ECU und Software Component (SW-C), der Export der erstellten Konfigurationsdateien und die Generierung der konfigurierten BSW.
  Es werden mit diesem Softwarepaket von "Elektrobit" die wesentlichen AUTOSAR-BSW-Module geliefert, welche nachfolgend beschrieben werden.

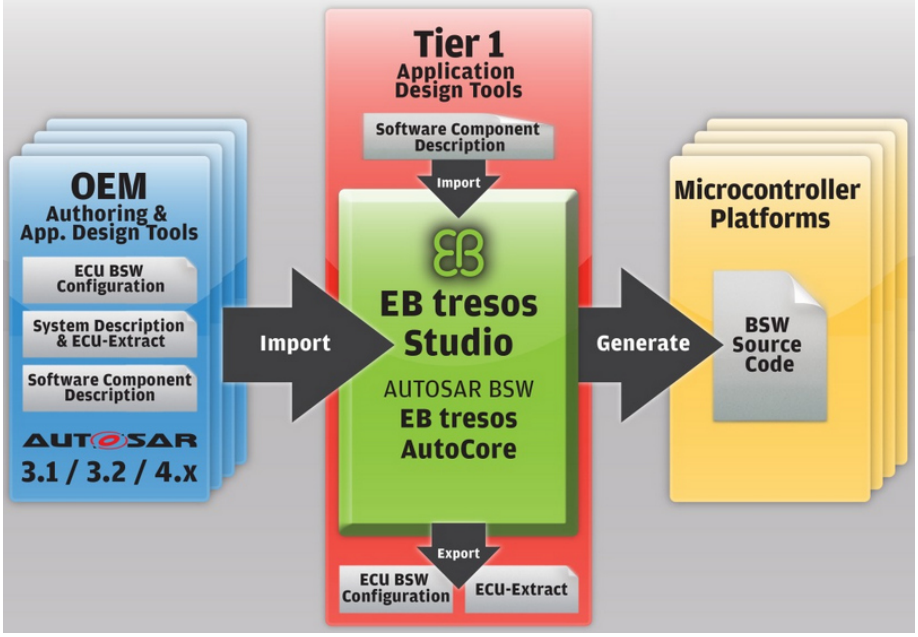

Abbildung 6.1: Konzept von EB Tresos Studio (Quelle: [23])

- BSW-Module [22, 35]

  Die AUTOSAR konforme BSW stellt den Kern des Projekttemplates dar. Für den Zugriff auf die Hardwarekomponenten sind die entsprechenden Hardwaretreibermodule, rund um ADC, CAN, DIO, GPT, MCU, PWM, welche in Kapitel 5.2 bereits näher erläutert wurden, zuständig. Die verwendeten Versionen der "Low Level Driver" stammen in dieser Implementierung vom Hersteller "Infineon" und stellen die direkte Schnittstelle zur Hardware dar [35]. Alle aufgesetzten Interface Module, sowie auch das AUTOSAR konforme Mehrkernbetriebssystem werden im Zuge des "EB Tresos AutoCore"-Pakets von "Elektrobit" bereitgestellt. Dieses BSW-Paket basiert auf den neuesten AUTOSAR-Spezifikationen und beinhaltet auch alle gängigen Kommunikationsprotokolle der Automotive Industrie. Es lässt sich in nachfolgende Teile gruppieren. Der Basisteil stellt die Hauptinfrastruktur für das "AutoCore"-Paket dar. Die RTE, abstrahiert die ASW von der BSW und sorgt somit für Flexibilität und Skalierbarkeit. Die Gruppe der Kommunikationsservices stellt sämtliche Interfaces für die Bus- und Netzwerkkommunikation zur Verfügung. Der Diagnose Stack beinhaltet alle Module welche für die externe und interne Überwachung verwendet werden. Das Real Time Operating System (RTOS) "AutoCore OS", führt unter anderem das Scheduling, die Ausführung der Tasks, die Ressourcenverwaltung und die Mehrkernimplementierung durch. Zuletzt wird die MCAL als zugehörige Gruppe gesehen, da "Elektrobit" auch die Mikrocontroller Module der verschiedenen Steuergerätehersteller im Rahmen dieses Pakets, mit anbietet. [22]

- AURIX Application Kit TC275 [34]

  Der "Aurix TriCore" gehört aktuell zu den stärksten Mehrkern-Mikrocontrollern am Markt. Er verfügt über die, für diese Arbeit wesentlichen, Hardwareunterstützungen für CAN, ADC, drei gesondert ansteuerbaren 32Bit CPU-Kerne mit je 200MHz und zugehörigen Speicherbereichen. Unterstützt werden unter anderem eine Zertifizierung bis ASIL-D, sowie aktuelle AUTOSAR-Spezifikationen.

- Altium Tasking Compiler 4.3r1 [2]

  Um aus dem von "EB Tresos Studio" generierten Programmcode eine, am "AURIX TriCore" ausführbare, Binärdatei zu erstellen, wird ein entsprechender Compiler benötigt. Dieser sollte den Industriestandards entsprechen, sauberen Code produzieren und in der Praxis einsetzbar sein. Wegen der Kompatibilität und vorhandener Erfahrungswerte wird im Zuge dieser Arbeit der "Altium Tasking Compiler" verwendet. Welcher ebenfalls für die aktuellen Sicherheitszertifizierungen der Industrie zugelassen ist.

- LAUTERBACH Trace32 [43]

  Um die vom Compiler generierte Binärdatei auf den "AURIX TriCore"-Mikrocontroller zu flashen, wird die Software "Trace32" sowie die dazugehörige entsprechende Hardware der Firma "Lauterbach" verwendet.
  Während der Implementierung wird dieses Werkzeug außerdem zum Debugging verwendet. Es verfügt über eine interaktive, grafische Benutzeroberfläche und ermöglicht das Auslesen sämtlicher relevanter System- und Kommunikationsregister.

- Vector CANalyzer [60]

  Da sich diese Arbeit, in Bezug auf die Buskommunikation ausschließlich mit dem CAN-Standard beschäftigt, ist die Auswahl eines entsprechenden Diagnosetools von hoher Wichtigkeit. Um die implementierte CAN-Kommunikation effizient testen zu können, wurde der "CANalyzer" der Firma "Vector" gewählt. Durch das, per USB verbundene "CAN-Case" ist die Überwachung sowie Simulation von CAN-Signalen ohne Einschränkungen möglich. Während der Implementierung wird dieses Werkzeug benutzt um CAN-Signale vom Mikrocontroller aufzuzeichnen, den Kommunikationsstatus zu überwachen sowie Testsignale zu Simulieren.

6.2 Anforderungen an das Projekttemplate

In dieser Arbeit wurden bereits einige Anforderungen an das entstandene Projekttemplate definiert. Diese werden nachfolgend zusammengefasst und ergänzt.
Das finale AUTOSAR konforme Projekttemplate soll ein einfaches und effizientes Integrieren von nicht AUTOSAR konformen Softwarekomponenten ermöglichen. Es darf dabei kein Unterschied zwischen Einkern oder Mehrkernanwendungen bestehen. In Letzterem muss das vorkonfigurierte Projekttemplate bereits über entsprechende, beispielhaft erstellte Anwendungsgruppen verfügen, um eine intuitive Zuteilung der Tasks zu ermöglichen. Der Systementwickler sollte sich in diesem Kontext hauptsächlich Gedanken über die Abhängigkeiten der Tasks untereinander und somit die genaue Zuteilung machen. Ein weiterer wichtiger Aspekt ist die automatische Signalkonfiguration und Skalierung. Der Systementwickler soll, ausgehend von der zu integrierenden ASW, Interfaces in der RTE aufrufen, um auf die entsprechenden externen Kommunikationskomponenten wie CAN, ADC, DIO und PWM zugreifen zu können. Diese Interfaces sollen automatisch generiert werden können, sowie bereits sämtliche Skalierungsberechnungen beinhalten. Um diese automatische Konfiguration der RTE zu ermöglichen, ist es notwendig den Treibermodulen, welche für die reibungslose Interaktion der Software mit der Hardware verantwortlich sind, eine lauffähige Standardeinstellung zu konfigurieren. Darauf aufbauend werden alle aufgesetzten Interface Module für die Interaktion untereinander vorkonfiguriert. Um die Konfiguration der BSW abschließen zu können, müssen noch die Einstellungen des RTOS in eine standardanwendungskompatible Form gebracht werden. Um besonders heiklen Echtzeitbedingungen gerecht zu werden, soll ein direkter Zugriff auf die Hardwareschicht möglich sein. Dadurch wird ein Kommunikationsoverhead vermieden und die Latenz deutlich verkürzt. Aufgrund sicherheitskritischer Anwendungsfälle ist auch die Zertifizierbarkeit, hinsichtlich des AUTOSAR-Standards und verschiedener Sicherheitsnormen, von großer Bedeutung.
Die Herausforderung im Hinblick auf die Toolchain besteht darin, die Implementierung für Änderungswünsche weitgehend offen zu halten. Der Austausch eines Werkzeugs oder eines Moduls darf zu keinen Problemen führen. Der Hauptgrund dafür ist, dass es ein großes Spektrum an potentiell zu integrierenden Softwaresystemen gibt, welche unter anderem für unterschiedliche Mikrocontroller oder Buskommunikationssysteme konzipiert sind.

6.3 Implementierung

In diesem Kapitel wird die genaue technischen Umsetzung des entstanden Projekttemplates erklärt. Dazu gehört der generelle Aufbau, die verwendeten BSW-Module, dessen Initialisierung sowie die Anbindung zu Hardware und RTE. In diesem Zuge wird auf den entwickelten ARXML-Generator und die Implementierung des ersten Demonstrators eingegangen.

6.3.1 Projekttemplate

Eine wesentliche Aufgabe ist die Erstellung eines Projekttemplates. Am Beginn dieses Projekts muss eine individuelle Struktur im "Elektrobit Tresos Studio" erstellt werden. Die Erstellung dieser zieht zahlreiche Herausforderungen mit sich, da die Herstellervariante verfeinert werden muss um den individuellen Anforderungen sowie der gegebenen Hardware zu entsprechen. Gleich zu Beginn des Projekterstellungsprozesses sind Änderungen in Makefiles wichtig. Diese werden hinsichtlich der verwendeten Entwicklungsumgebung, des Compilers, der Projektpfade und des jeweiligen Mikrocontrollers personalisiert. Im Hinblick auf die benötigten BSW-Module und zu konfigurierenden Abhängigkeiten existieren umfangreiche Dokumentationen. Das Anwendungsgebiet für diese Komponenten ist allerdings derart umfangreich ausgelegt, dass die Funktionsdokumentation sehr generisch gehalten wird. Um dieses Projekt umsetzen zu können, sowie sämtliche praxisrelevante Informationen aus den Dokumentationen zu extrahieren, wird eine erhebliche Einarbeitungszeit benötigt.
Das Projekttemplate soll sämtliche BSW-Module enthalten, welche für die grundsätzliche Lauffähigkeit einer Standardanwendung benötigt werden. Da es sich lediglich um ein ausführbares Grundgerüst handelt, wird noch keine ASW an die RTE angebunden. Sobald eine ASW integriert wird, sollen nur noch Kommunikations- sowie Task Konfigurationen erforderlich sein, um ein ausführbares System zu erhalten. Dieses Template soll bereits generierbar, kompilierbar und auf dem Mikrocontroller ausführbar sein.
Die dafür konfigurierten BSW-Module sind:

- OS
- EcuM
- DET
- DEM
- BswM
- MCU
- DIO
- PORT
- MemMap
- RTE (Kein BSW-Modul)

6.3.1.1 Modulinitialisierung

Nach dem Hinzufügen der benötigten Module existieren Standardkonfigurationen, welche allerdings nicht lauffähig sind. Um ein lauffähiges Template zu erhalten, sind Mikrocontroller abhängige Einstellungen zu tätigen, sowie semantische Fehler auszubessern. Diese Schritte erfordern ausgeprägtes Wissen über die AUTOSAR-Architektur, die Automotive Domäne, die einzelnen BSW-Module sowie die Entwicklungsumgebung "Elektrobit Tresos Studio" selbst. Nachfolgend werden einige wesentliche Anpassungen vorgestellt.

Das EcuM-Modul ist für die Initialisierung des Systems zuständig. Daher muss man die Initialisierungsfunktionen sämtlicher verwendeter Treibermodule zu der Initialisierungsliste hinzufügen. Wesentlich ist auch die Initialisierung der "Mcu Clock". Weitere erforderliche Einstellungen in diesem Modul sind die verschiedenen Ruhezustandsmodi sowie der Systemneustart.
Um eine, mit dem Mikrocontroller kompatible, Konfiguration erstellen zu können, ist es notwendig die "McuClockSettings" im MCU-Modul anzupassen. Ein Ausschnitt der verwendeten Einstellungen wird in Abbildung 6.2 gezeigt. Die GTM-Funktionalität ist auch im MCU-Modul enthalten. Sie werden gleich konfiguriert, um einer eventuellen Integration eines ADC-Moduls nicht im Weg zu stehen.

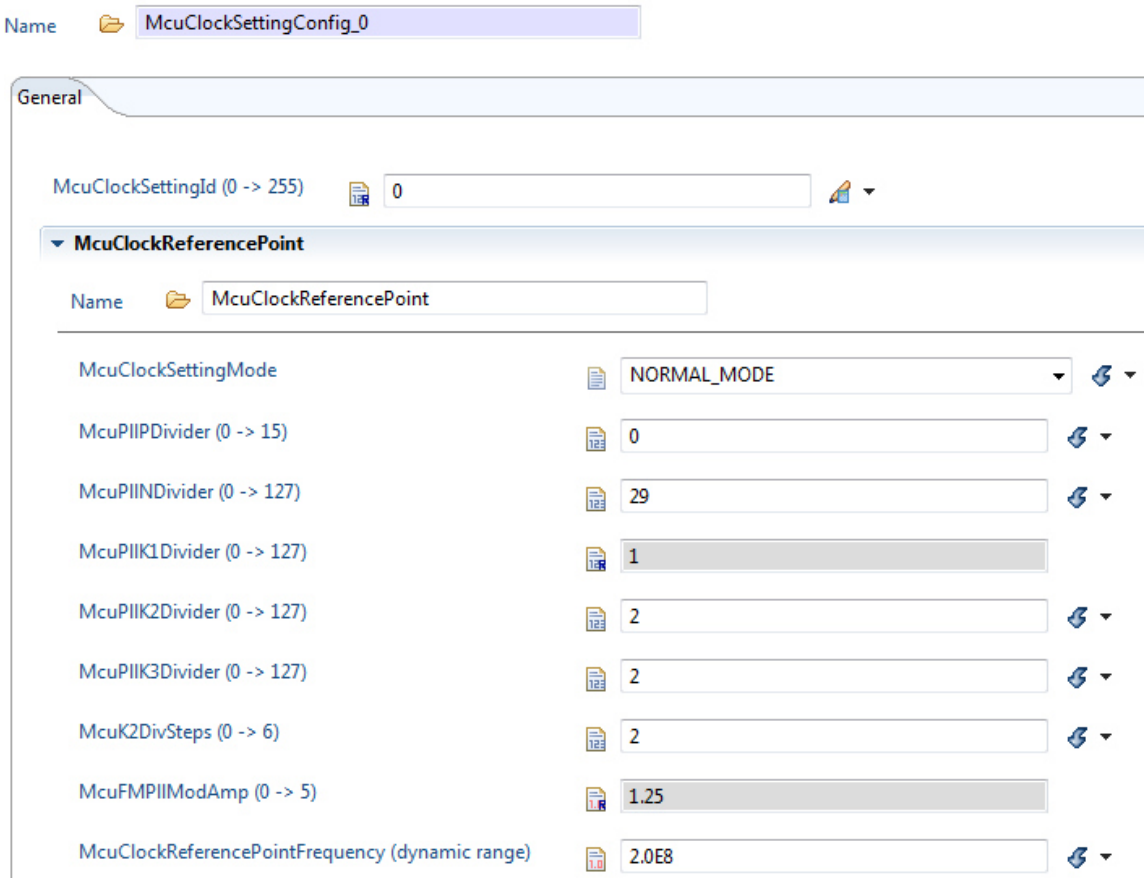

Abbildung 6.2: Konfiguration der MCU-Clock

Im OS-Modul werden zunächst relevanten BSW-Tasks sowie die benötigten Scheduletabellen erstellt. Dazu zählen der Initialisierungstask, Tasks für den Schedulemanager und die RTE-Funktionalität.
Die notwendigen Konfigurationen für das BswM-Modul kann man mit Hilfe der Assis-

tenten der Entwicklungsumgebung erstellen. In Abbildung 6.3 ist ersichtlich, dass die Systeminitialisierung grundsätzlich in den Modulen EcuM und BswM abläuft. Die erste Phase nennt sich "Startup One", welche vorrangig das Betriebssystem startet. Die Phase "Startup Two" initialisiert das Schedulemanager und das BswM Modul. Beim Start dieses Moduls werden unter anderem die Kommunikationstreiber, die zugehörigen Interfaces und Diagnoseservices initialisiert. Nach erfolgreicher Initialisierung wird das System für den Betrieb frei gegeben.

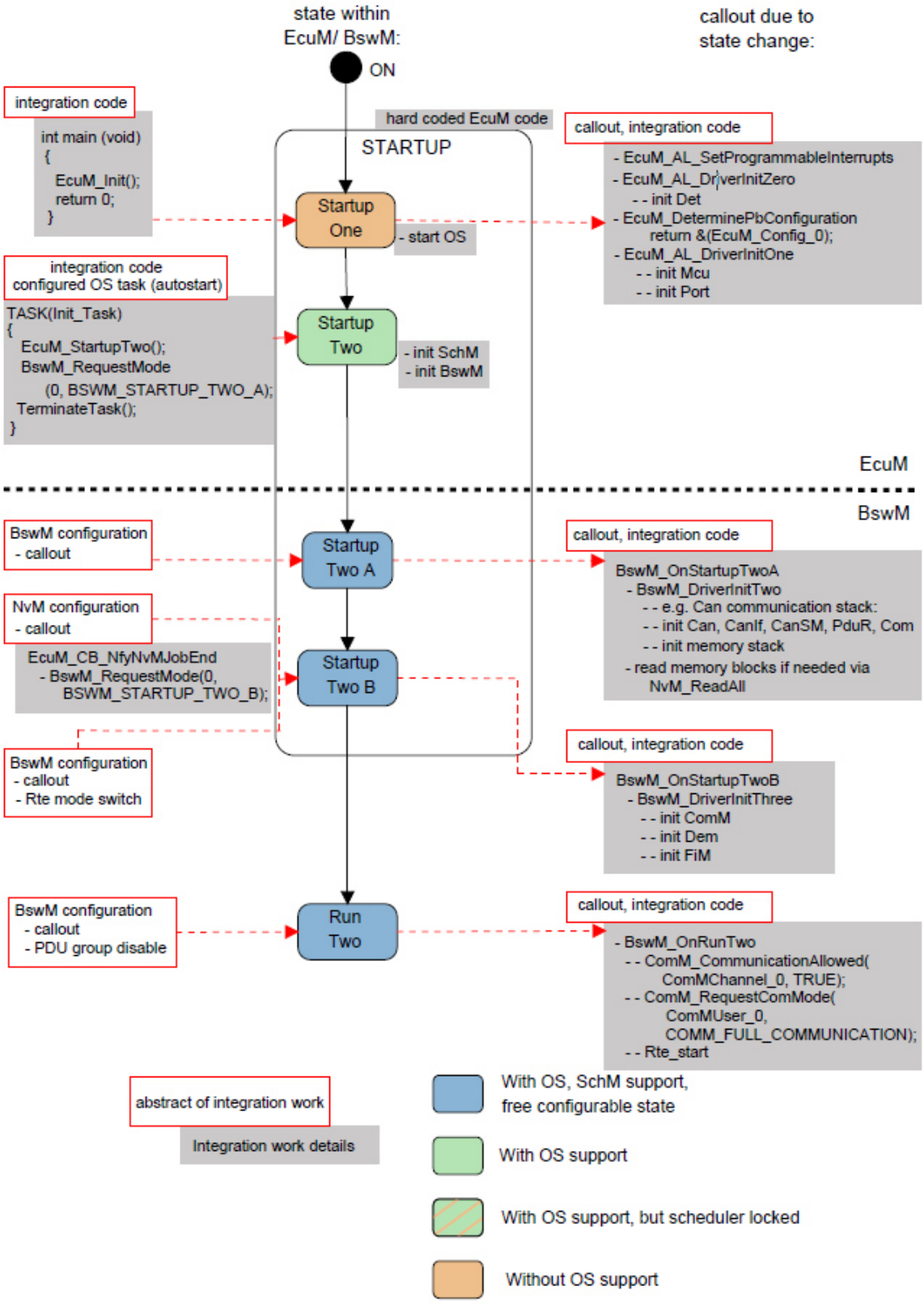

Abbildung 6.3: Ablaufdiagramm des Systemstarts / Initialisierung

6.3.1.2 Mehrkerninitialisierung

Um das Projekttemplate mehrkerntauglich zu machen, werden Anpassungen an der "main.c" durchgeführt. Jeder Kern erstellt eine eigene Instanz dieser Klasse. Daher muss man in dieser Klasse zwischen den aufrufenden Kernen differenzieren. Im Quelltext 6.1 ist diese Implementierung dargestellt.

Quelltext 6.1: Mehrkernversion der main Klasse

```
int main(void)
{
    CoreIdType act_core_id = GetCoreID();
    if(act_core_id == 0)
    {
        StatusType core1_status;
        StartCore(1, &core1_status);
        StatusType core2_status;
        StartCore(2, &core2_status);
        EcuM_Init();
    }
    else if(act_core_id == 1)
    {
        StartOS(0);
    }
    else if(act_core_id == 2)
    {
        StartOS(0);
    }
return 0;
}
```

6.3.2 ARXML-Generator

Das entstandene Projekttemplate soll ein zügiges und effizientes Einbinden von nicht AUTOSAR konformer ASW in ein AUTOSAR konformes Projekt ermöglichen. In der Praxis gibt es allerdings eine Vielzahl an potentiellen zu integrierenden Projekten, von denen längst nicht alle eine kompatible ARXML-Signaldefinitionsdatei zur Verfügung stellen. Diese wird benötigt um eine automatische RTE-Schnittstellenkonfiguration, inklusive Skalierungseinstellungen zu ermöglichen. In dieser Implementierung werden Interfaces für die CAN-Kommunikation, sowie auch für ADC-Werte bereitgestellt.
Ein Nebenziel dieser Arbeit ist somit die Erstellung eines entsprechenden Generators, welcher aus einer vorhandenen DBC-Datei und einer textuellen Signalbeschreibung in einer EXCEL Datei eine AUTOSAR konforme und für den Import kompatible ARXML-Datei erstellt. Dieser Ablauf ist auch in Abbildung 5.2 ersichtlich.
Umgesetzt wird die Implementierung dieses Hilfswerkzeugs mit der Programmiersprache C#, dem .NET Framework 4 und der XML-Spezifikation. Nachfolgend werden die technischen Details zur Programmierung, zur Erweiterbarkeit sowie zum Einsatz dieses neuen Werkzeugs beschrieben.

Dieses Projekt unterliegt aus seiner Natur heraus ständigen Anpassungen. Aus diesem Grund ist das Ziel dieses Generators, in seiner Funktionsweise, leicht veränderbar zu sein. Hierfür wird ein sehr dynamischer Programmieransatz gewählt und die ARXML-Datei aus einer Klasse mit XML definierten Elementen aufgebaut. Diese Klassenstruktur ist jederzeit, ohne vorherige Änderungen durchführen zu müssen, mit weiteren gewünschten ARXML-Feldern erweiterbar. Nach dem Einlesen aus der DBC und der Kombination dieser Daten, mit den Werten aus der EXCEL Datei, wird eine Instanz der XML-Klasse erstellt und mit den berechneten Resultaten befüllt. Anschließend startet man die Dateierstellung, wobei zwei unterschiedliche Dateien erstellt werden. Erstens wird die geforderte ARXML-Datei, welche beim Importieren in das Projekttemplate die nötigen RTE-Schnittstellen definiert. Als zweites wird eine, zur C-Programmiersprache kompatible, Header-Datei, welche zusätzlich in das Projekttemplate eingebunden wird, kreiert. Diese Datei umfasst alle nötigen Berechnungen bezüglich der Signalskalierung sowie der Offset Werte. Technisch umgesetzt wird dieser Anspruch durch Makrodefinitionen, welche die erstellten RTE-Interfaces um die relevanten Berechnungen erweitert.

6.3.2.1 Komponenten der Grafische Benutzeroberfläche (GUI)

Da der Generator verschiedene Benutzerinteraktionen, wie etwa die Pfaddefinitionen zum Lesen und Schreiben benötigt sowie die manuelle Überprüfung der gelesenen Inputdateien ermöglicht, wird aus Gründen der Bedienbarkeit eine interaktive, grafische Benutzeroberfläche entwickelt. Die Abbildung 6.4 zeigt die GUI des Generators. Zu sehen ist die Auswahl der beiden Inputdateien, sowie die anschließend herausgelesenen Signale, inklusive der jeweiligen Signalname, Längendefinitionen, Datentypen und Signalnummern. In der Praxis müssen nicht alle aus der textuellen EXCEL Datei gelesenen Signale über eine vollständige Definition in der DBC-Datei verfügen. Dies gilt zum Beispiel für eventuell erstellte Debugsignale. Für diese Anforderung werden die Signale in zwei getrennten Listen dargestellt. Nach der manuellen Überprüfung der Resultate kann man die Erstellung der Dateien über den "Generate" Button bestätigen. Für eventuell auftretende Fehlerfälle sind im Hilfemenü verschiedene Hinweise bezüglich der Anforderungen an die Excel Datei hinterlegt.

6.3.2.2 Implementierung der XML-Klasse

Das Herzstück des Generators stellt die serialisierbare XML-Klasse dar. Diese Klasse ist ein genaues Abbild der generierten ARXML-Datei. Änderungen an dieser Struktur werden daher direkt in die Ausgabe übernommen. Der Quelltext 6.2 zeigt einen Ausschnitt, genauer gesagt wird das ARXML-Feld "SWCImplementation" dargestellt. In der aktuellen Version des Programms werden derzeit 49 solcher Felder serialisiert.

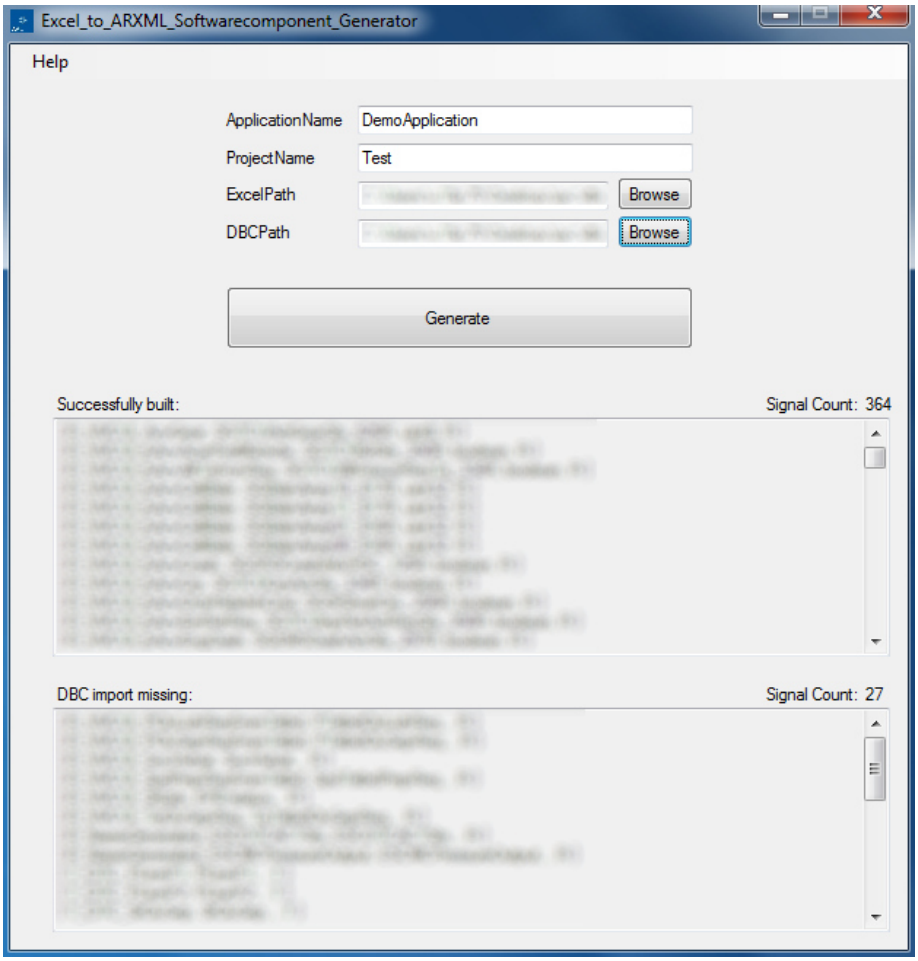

Abbildung 6.4: GUI des ARXML-Generators

Quelltext 6.2: Definition eines Elements der XML Klasse

```
public class SWCImplementation
{
    [XmlElement("SHORT-NAME")]
    public string implName;
    [XmlElement("CODE-DESCRIPTORS")]
    public CodeDescriptors codeDescriptors = new CodeDescriptors();
    [XmlElement("PROGRAMMING-LANGUAGE")]
    public string programmingLanguage = "C";
    [XmlElement("RESOURCE-CONSUMPTION")]
    public ResourceConsumption resourceConsumption = new ResourceConsumption();
    [XmlElement("SW-VERSION")]
    public string swVersion = "1.0.0";
    [XmlElement("VENDOR-ID")]
    public int vendorId = 1;
    [XmlElement("BEHAVIOR-REF DEST=\"SWC-INTERNAL-BEHAVIOR\"")]
    public string behaviorRef;

    public SWCImplementation()
    {
    }
    public SWCImplementation(string applicationName, string projektName)
    {
        implName = NameGenerators.getIMPLName(projektName);
        behaviorRef = "/" + applicationName + "/SwComponentTypes/" +
            NameGenerators.getSWCName(projektName) + "/" + NameGenerators.
            getIBName(projektName);
    }
}
```

6.3.2.3 Resultierende ARXML-Datei

Das Ergebnis des Generators ist eine AUTOSAR konforme ARXML-Datei, welche für den Import in das Projekttemplate geeignet ist. Nach dem Import der Datei, verfügt die RTE über alle notwendigen Konfigurationen bezüglich der Kommunikationsschnittstellen. Ab nun stehen die entsprechenden Lese- und Schreibschnittstellen zum COM-Modul sowie auch für andere Hardwaremodule, welche Werte in die ASW liefern, zur Verfügung. Ein solches Hardwaremodul wäre zum Beispiel das ADC-Modul. Der Zugriff auf diese Werte wird über eine sogenannte IRV der RTE umgesetzt.
Die resultierende ARXML-Datei, für die in Abbildung 6.4 gezeigten 364 validen Signale umfasst ca. 14000 Zeilen. Der Quelltext 6.3 zeigt das entstandene Element aus der vorangegangenen XML-Definition 6.2 nun in der ARXML-Datei.

Quelltext 6.3: Definition eines Elements der ARXML Datei

```
<AR-PACKAGE>
    <SHORT-NAME>SwcImplementations</SHORT-NAME>
    <ELEMENTS>
        <SWC-IMPLEMENTATION>
        <SHORT-NAME>Impl_JAC</SHORT-NAME>
            <CODE-DESCRIPTORS>
                <CODE>
                    <SHORT-NAME>SRC</SHORT-NAME>
                    <ARTIFACT-DESCRIPTORS>
                        <AUTOSAR-ENGINEERING-OBJECT>
                            <SHORT-LABEL>EngObject</SHORT-LABEL>
                            <CATEGORY>SWSRC</CATEGORY>
                        </AUTOSAR-ENGINEERING-OBJECT>
                    </ARTIFACT-DESCRIPTORS>
                </CODE>
            </CODE-DESCRIPTORS>
            <PROGRAMMING-LANGUAGE>C</PROGRAMMING-LANGUAGE>
            <RESOURCE-CONSUMPTION>
                <SHORT-NAME>Resources</SHORT-NAME>
            </RESOURCE-CONSUMPTION>
            <SW-VERSION>1.0.0</SW-VERSION>
            <VENDOR-ID>1</VENDOR-ID>
            <BEHAVIOR-REF DEST="SWC-INTERNAL-BEHAVIOR">/DemoApplication/
                SwComponentTypes/SWC_JAC/IB_JAC</BEHAVIOR-REF>
        </SWC-IMPLEMENTATION>
    </ELEMENTS>
</AR-PACKAGE>
```

6.3.2.4 Resultierende Makro Datei

Die Anforderungen der automatischen Skalierung und Offsetberechnung für eingehende und ausgehende Signale, werden durch die, ebenfalls aus dem Generationsprozess entstehende, Makro Datei erfüllt. Die dazu benötigten Werte werden bereits im Zuge des Einlesens aus der DBC-Datei herausgelöst. Die Makrodatei enthält die definierten RTE-Schnittstellen und erweitert sie um die entsprechenden Berechnungen. Im Quelltext 6.4 ist sowohl ein Makro für einen Lesenden sowie für einen Schreibenden RTE-Aufruf dargestellt.

Quelltext 6.4: Makro zur automatischen Skalierung für R/W Signale in RTE

```
#define Rte_Scaled_Rte_Read_R_ControllerX_PduX_SignalX_123R(result) {uint8
    calc_help; Rte_Read_R_ControllerX_PduX_SignalX_123R(&calc_help); result = (
    uint8)((calc_help *10.F)+30.F);}
#define Rte_Scaled_Rte_Write_T_ControllerY_PduY_SignalY_321T(float_value) {uint8
    calc_help = (uint8)((float_value +0.F)/0.1F);
    Rte_Write_T_ControllerY_PduY_SignalY_321T(calc_help);}
```

6.3.3 Integration des Demonstrators

Um die erste Funktionsfähigkeit des Projekttemplates zu überprüfen wird ein Demonstrator integriert, welcher alle notwendigen Basisfunktionen abdeckt. Der Demonstratoraufbau besteht aus einem Gaspedal, einer Drosselklappe und einem "TriCore"-Mikrocontroller.

Evaluiert wird damit unter anderem die Funktion des Taskschedulings, die CAN-Kommunikation und die Verarbeitung analoger Signale in einer Mehrkernumgebung.
Die ASW-Funktionen werden auf 3 Prozessorkerne Aufgeteilt. Zu Testzwecken wird dem erste Kern die komplette BSW-Funktionalität sowie das Einlesen der Analogen Werte zugewiesen. Der Zweite Kern ist für sämtliche Berechnungen der ASW zuständig. Der dritte Kern ist für den, in Kapitel 5.4 beschriebenen, automatischen Beschleunigungsmodus verantwortlich.
Die Basis für die Integration der Demonstrator ASW bildet das bereits definierte, vollständig vorkonfiguriert AUTOSAR-Projekttemplate, welches in der Arbeitsmappe im "Elektrobit Tresos Studio" vorbereitet ist. Zusätzlich existiert noch eine DBC-Datei, welche das Kommunikationsverhalten des CAN-Controllers abbildet und eine entsprechende EXCEL Datei, welche die RTE-Kommunikation textuell beschreibt. Aus diesen beiden Dateien erstellt der in Kapitel 6.3.2 beschriebene Generator, die benötigte ARXML-Datei für die Signalkonfiguration.
Die Demonstrator-Konfiguration läuft in verschiedenen Stufen ab und beginnt bei der Anpassung der relevanten BSW-Module. In den darauffolgenden Kapiteln werden die aufeinanderfolgenden Schritte näher erläutert.

6.3.3.1 OS-Konfiguration

Zunächst ist es notwendig das vorkonfigurierte Betriebssystem um die benötigten Tasks, Events, Zähler und der Mehrkernkonfiguration zu erweitern.
Der erste Schritt ist die entsprechenden von der ASW benötigten, Taskeinträge im Betriebssystem Modul hinzuzufügen. Die dazugehörige Übersicht aller Demonstrator Tasks ist in Abbildung 6.5 ersichtlich.

Name Os

General | OsAlarm | OsAppMode | OsSpinlock | OsApplication | OsCounter | OsEvent | OsIoc | OsIsr | OsCoreConfig | OsOS | OsResource | OsScheduleTable | OsTask | Published Information

OsTask

Index	Name	OsTaskActivation	OsTaskPr...	OsTaskSc...	OsStacksi...
0	Init_Task	1	4	NON	1024
1	SchMDiagStateTask_20ms	1	50	FULL	512
2	Rte_Time_Task	1	52	FULL	1024
3	Rte_Event_Task	1	51	FULL	1024
4	SchMComTask_1ms	1	150	FULL	1024
5	SchMComTask_5ms	1	150	FULL	1024
6	ASWTask0	1	49	FULL	512
7	ASWTask1	1	48	FULL	512
8	ASWTask2	1	47	FULL	512

Abbildung 6.5: Auflistung aller Demonstrator Tasks

Anschließend werden die nutzbaren Prozessorkerne auf drei, dem "TriCore"-Mikrocontroller entsprechend, erhöht. Da jeder Kern eigene Tasks besitzt, werden zunächst die drei Hardwarezähler in der Software abgebildet. In Abbildung 6.6 ist die genaue Konfiguration des Hardware Zählers für den ersten Kern abgebildet.

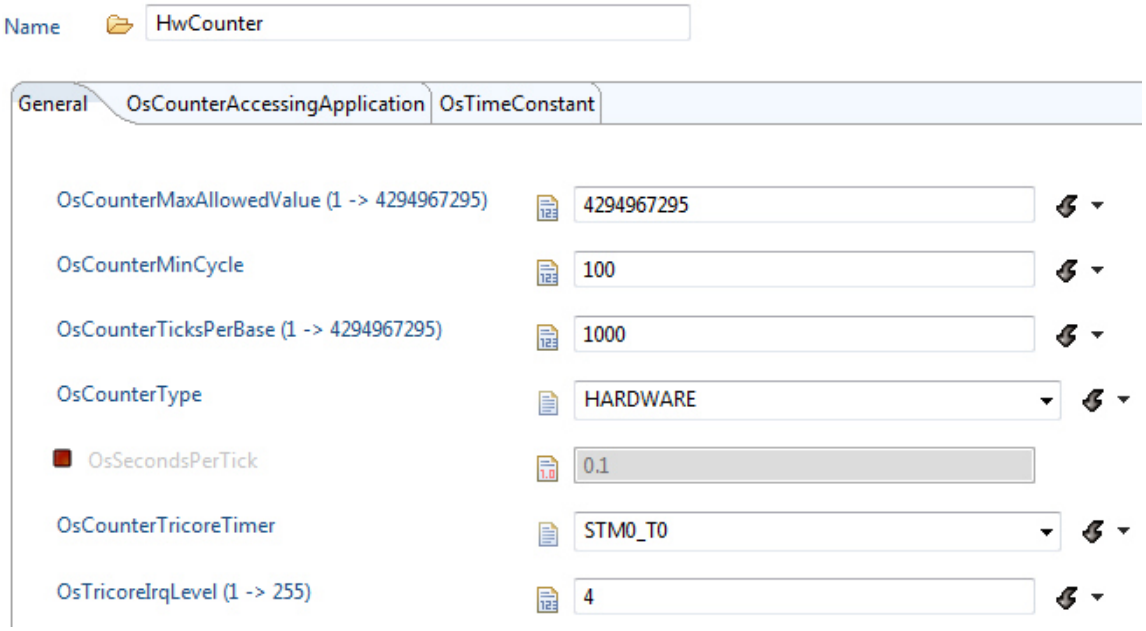

Abbildung 6.6: Konfiguration von Hardware Zähler 0

Basierend auf den eben definierten Zählern ist, der nächste Schritt die Definition von Alarmen, welche für das Triggern der ASW-Tasks zuständig sind. Da einer der Tasks über ein eigenes Event ausgelöst wird, sind nur zwei solcher Alarme notwendig. In Abbildung 6.7 wird die Konfiguration des Alarms, welcher das Einlesen steuert, abgebildet.

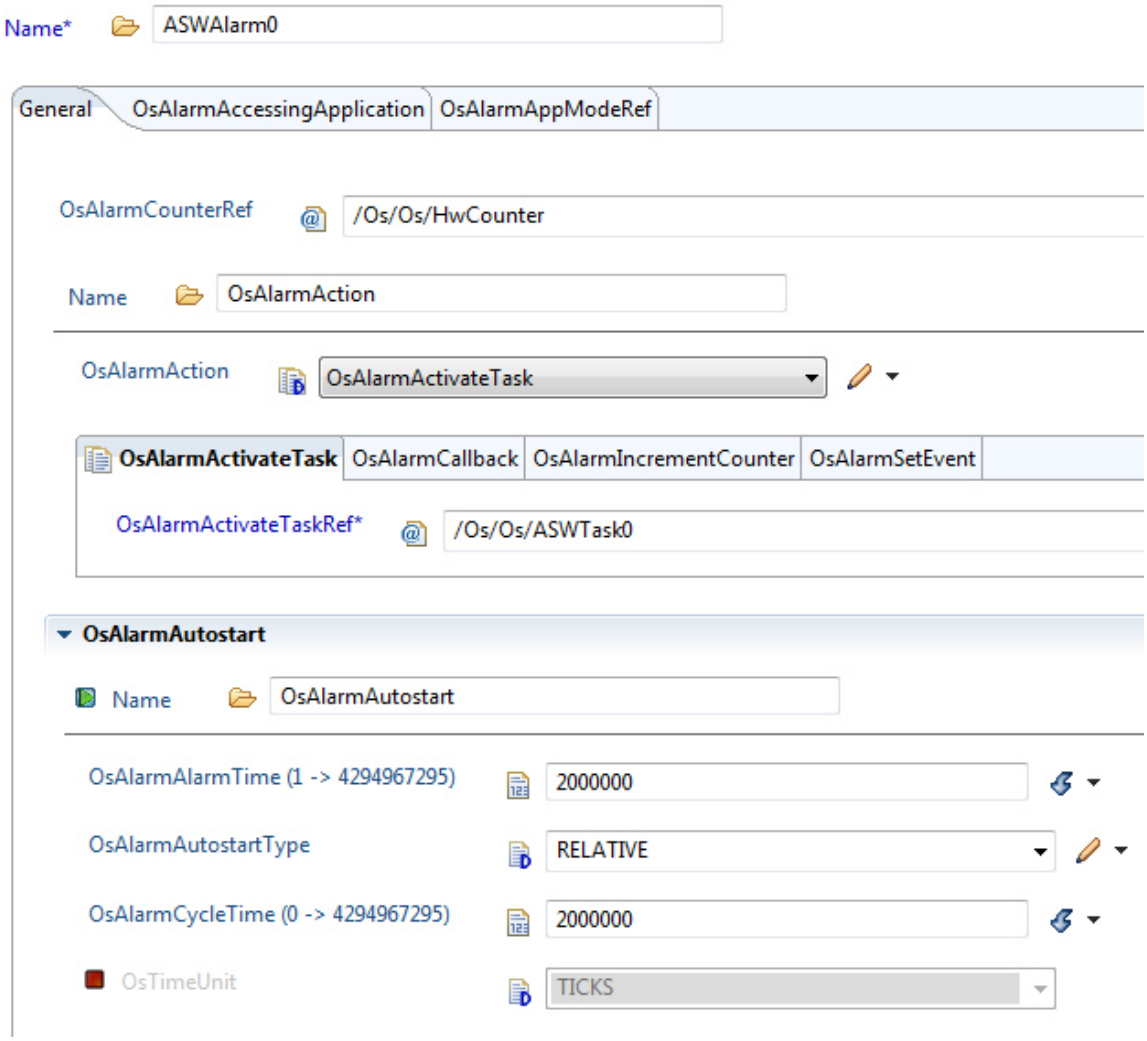

Abbildung 6.7: Konfiguration von ASWAlarm 0

Je nach Definition der ASW-Tasks werden verschiedene Spinlocks benötigt, um einen reibungslosen Zugriff auf die, zwischen Anwendungen und Kernen verteilte, Ressourcen zu gewährleisten. Im Anschluss werden verschiedene Anwendungsgruppen erstellt, welche den einzelnen Kernen zugeordnet werden. Diese Gruppen sind der Grundstein für ein funktionierendes Mehrkernsystem. Im nächsten Schritt werden sämtliche Tasks, Events, Zähler und Alarme der gewünschten Anwendungsgruppe und somit einem Kern zugeordnet. Im Hinblick auf die Demonstrator Konfiguration wird der "ASWTask0" der Anwendungsgruppe für "Kern0" zugeordnet. Das gleiche Vorgehen gilt ebenfalls für die übrigen Tasks.

6.3.3.2 Kommunikationsbus Konfiguration

Bezüglich der Buskommunikation beschäftigt sich diese Arbeit ausschließlich mit der CAN-Kommunikation. Um ein reibungsloses Zusammenspiel der Module zu ermöglichen, muss man die Module CAN, CanIf, Controller Area Network State Manager (CanSM), PduR und COM konfigurieren. Basierend auf dem vorkonfigurierten Projekttemplate erfolgt der überwiegende Teil der Konfiguration, durch das Importieren der DBC-Datei in das "Elektrobit Tresos Studio" automatisch. Zusätzlich notwendige Einstellungen, bezüglich der Baudrate und Port Konfiguration sowie die automatisch generierten Konfigurationen werden nachfolgend näher erläutert.
Die Baudrate gibt die Signalrate an, und gehört entsprechend den CAN-Controllern eingestellt. In Abbildung 6.8 sind die verwendeten Einstellungen ersichtlich.

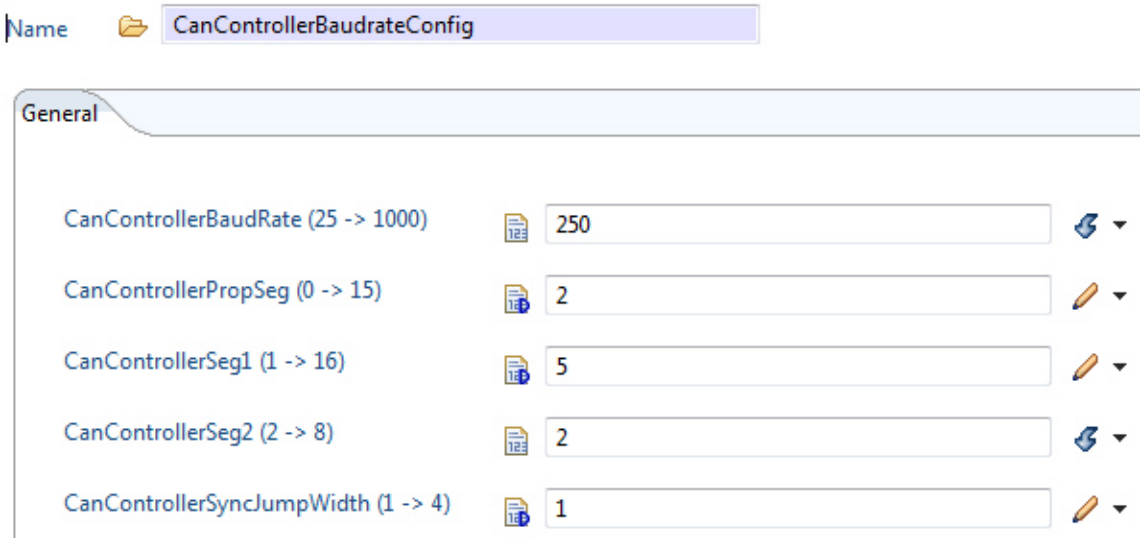

Abbildung 6.8: Konfiguration der Baudrate

Die vom CAN-Controller belegten Pins müssen aus der Hardwaredefinition ausgelesen und anschließend im PORT Modul definiert werden.
Durch den Import der DBC-Datei wird das CAN-Modul bereits mit initialen Werten konfiguriert, welche allerdings manuell zu vervollständigen sind. Abbildung 6.9 zeigt die vollständige Konfiguration.

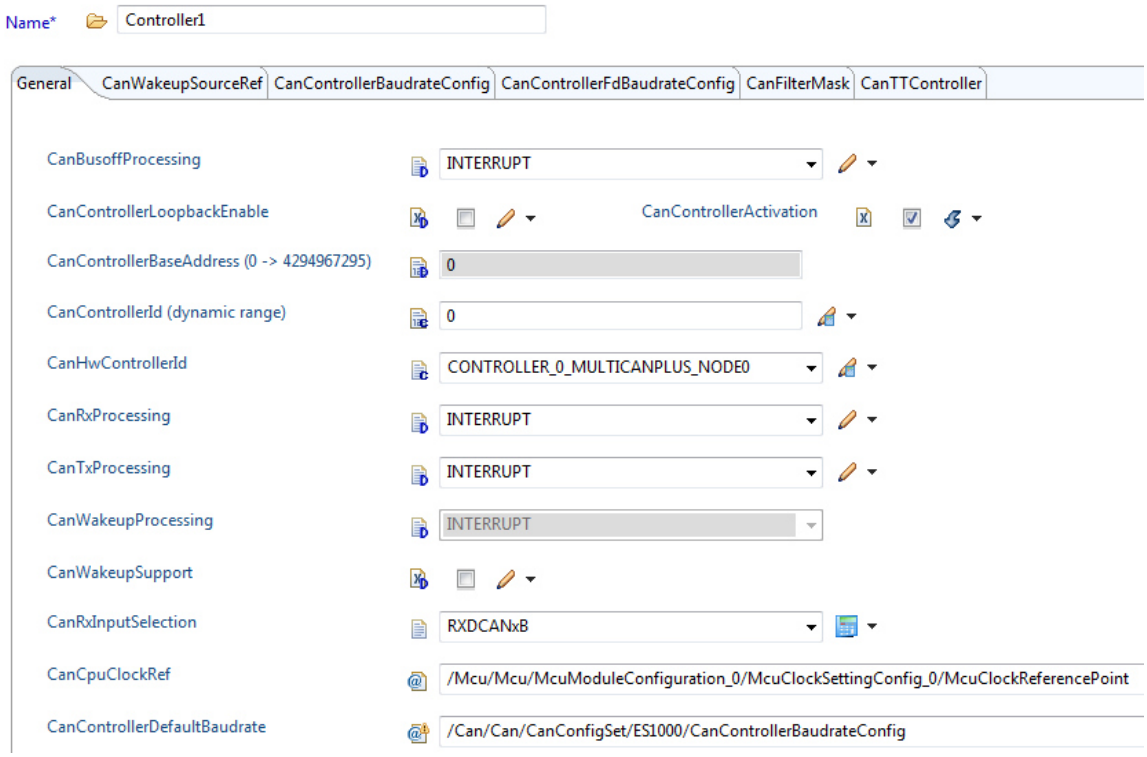

Abbildung 6.9: Konfiguration des CAN-Controllers

Die unterste Speicherebene stellt ein CAN-Nachrichtenobjekt dar. In diese werden eintreffende CAN-Signale initial gespeichert sowie ausgehende übertragen. Zudem ist es an dieser Stelle möglich Filtermasken zu definieren, welche sich auf die Nachrichtenannahme beziehen. In Abbildung 6.10 sind die automatisch erstellten Einträge ersichtlich. Es wird je ein Eintrag zum Senden und Empfangen erstellt.

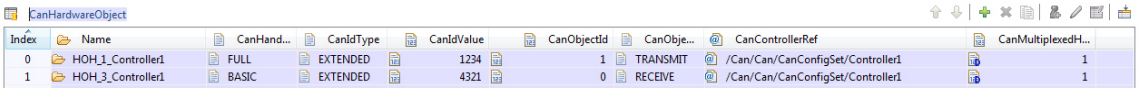

Abbildung 6.10: CAN-Nachrichtenobjekte

Als nächster Schritt folgt das CanIf-Modul. Auch dieses ist bereits automatisch konfiguriert. Interface Definitionen bezüglich des Controllers sowie den Lese- und Schreibbuffern erfolgen in diesem Modul. Es existieren die in Abbildung 6.11 automatisch generierten Definitionen, bezüglich der PDUs.

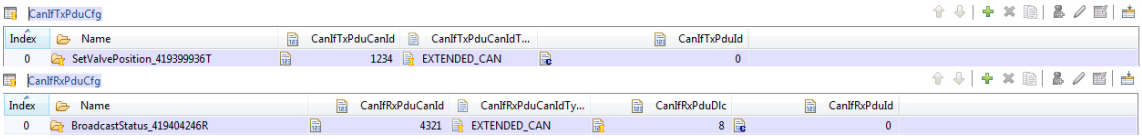

Abbildung 6.11: PDU-Konfigurationen im CanIf-Modul

Das CanSM-Modul wird in der generierten Standardkonfiguration belassen. Die vordefinierten Werte, bezüglich des "Controller1" Netzwerks, sind für die generelle Funktion ausreichend.
Das PduR-Modul verteilt die PDUs zwischen dem CanIf und dem COM-Modul. Diese könnten allerdings auch für LIN, FlexRay, Ethernet, sowie zur Diagnostik verwendet werden. Die manuellen Einstellungen beschränken sich in diesem Kontext darauf, das COM-Modul als "UpperModule" und das CanIf-Modul als "LowerModule" zu konfigurieren. Alle anderen verfügbaren Moduleinträge werden deaktiviert.
Da das COM-Modul eine Busprotokoll unabhängige PDU-Verwaltung ermöglicht, stellt es das Kernstück innerhalb der AUTOSAR-Buskommunikation dar. In diesem Modul werden die einzelnen Signale zu PDUs zusammengefasst. Diese wiederum können zu PDU-Gruppen gekapselt werden und sind auch entsprechend konfigurierbar. In Abbildung 6.12 sind die verschiedenen CAN-Signale, welche aus der DBC-Datei exportiert werden, dargestellt.

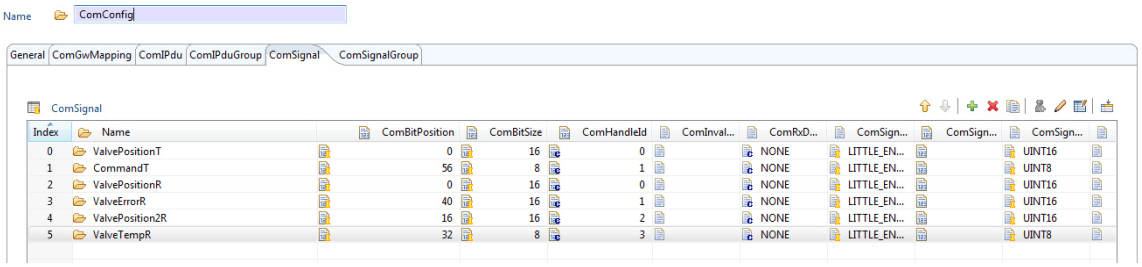

Abbildung 6.12: Signalkonfiguration im COM-Modul

Um die Buskommunikation für die ASW zugreifbar zu machen, erfolgt nun der Import der generierten ARXML-Kommunikationsdatei. Dadurch werden die entsprechenden, von der ASW benötigten Interfaces in der RTE bereitgestellt und müssen lediglich manuell im RTE-Editor des "Elektrobit Tresos Studio" den jeweiligen COM-Signalen zugewiesen werden. Als letzten Schritt zur funktionierenden Buskommunikation erfolgt noch die Integration der, in Kapitel 6.3.2.4 beschriebenen, Makrodatei zur automatischen Signalskalierung.

6.3.3.3 ADC-Konfiguration

Das verbundene Gaspedal liefert die Werte als Analogsignal. Für diesen Umstand wird das ADC-Modul konfiguriert. Zuerst wird im PORT Modul ein verfügbarer analoger Eingangsport definiert und im ADC-Modul referenziert. Auf Basis dieser Referenz wird ein "AdcChannel" definiert, welcher einer "AdcGroup" hinzugefügt wird. Dieser Gruppe kann man verschieden Konvertierungsmodi, wie zum Beispiel einmalig und zyklisch, zuweisen. Um die Konvertierung automatisch beim Systemstart auszuführen, wird der Start API-Aufruf in die Initialisierungssequenz und der Stop Aufruf in die Dnitialisierungssequenz des Systems eingebunden. Um aktuelle Resultate erhalten zu können, müssen die Interrupts der Konvertierung manuell in einem zyklischen BSW-Task abgearbeitet werden. Hierfür wird die Klasse "IoHwAb" implementiert. Wie im Quelltext 6.5 zu erkennen ist, existiert darin eine Start, Stop und MainFunction API.

Quelltext 6.5: Implementierung des Steuerungsinterfaces des ADC Moduls

```
#include "IoHwAb.h"

uint32 ADC_IoHwAb_StartConversion(Adc_GroupType Group, Adc_ValueGroupType *
    DataBufferPtr)
{
    while( (Adc_17_GetStartupCalStatus()) != E_OK);
    Std_ReturnType AdcBufferSetupStatus = Adc_SetupResultBuffer(Group,
        DataBufferPtr);

    if(AdcBufferSetupStatus == E_OK)
    {
        Adc_StartGroupConversion(Group);
        Adc_EnableGroupNotification(Group);
        return E_OK;
    }
    return E_NOT_OK;
}

void ADC_IoHwAb_StopConversion(Adc_GroupType Group)
{
    Adc_StopGroupConversion(Group);
    Adc_DisableGroupNotification(Group);
}

void ADC_IoHwAb_MainFunction()
{
    /*Handles the interrupts from Group that uses Request Source 0 of ADCx.*/
    Adc_IsrSrn0AdcRS0(0);
}
```

Um der ASW-Zugriff auf die ADC-Werte zu verschaffen, werden im ARXML-Generator IRV-Variablen definiert. Somit wird dem ADC-Task in der BSW, über einen RTE-Funktionsaufruf, das Abspeichern der Resultate in Feldern der RTE ermöglicht. Im Anschluss kann die ASW aus diesen Variablen über einen lesenden RTE-Aufruf die berechneten Werte auslesen lassen. Die beiden Funktionsaufrufe, für den schreibenden und lesenden Zugriff, auf die RTE-Variable sind im Quelltext 6.6 dargestellt.

Quelltext 6.6: Zugriff auf IRV in der RTE

```
#define Rte_Scaled_Rte_IrvRead_ADC_CurrentPedalValue
    Rte_IrvRead_ADC_CurrentPedalValue
#define Rte_Scaled_Rte_IrvWrite_ADC_CurrentPedalValue
    Rte_IrvWrite_ADC_CurrentPedalValue
```

6.3.3.4 ASW-Integration

Nachdem nun das Projekttemplate für dieses Demonstrator System vollständig konfiguriert ist, folgt als letzter Schritt die Integration der ASW selbst. Die Aufgaben sind nun sowohl das Einbinden der ASW-Programmdateien, als auch die Verknüpfung dieser mit den, bereits erstellten, RTE-Interface Funktionen.
Die ASW besteht aus mehreren Klassen, welche sich in drei unterschiedliche Gruppen einteilen lassen. Die erste Gruppe umfasst die Funktionen zum Einlesen, Aufbereiten und Ausgeben der Signale zur RTE. Die zweite Gruppe regelt die Berechnungen bezüglich der Drosselklappenposition, welche von der Gaspedalposition abhängig sind. Die letzte Gruppe ist für die Berechnungen der automatischen Positionierung zuständig.
Äquivalent zu den gerade beschriebenen Gruppen werden in der ASW drei Tasks erstellt, welche bereits in der BSW konfiguriert, durch die Zuordnung zu den entsprechenden Anwendungsgruppen auf die gewünschten Kerne verteilt und von den jeweiligen Alarmen aufgerufen werden. Im Sequenzdiagramm 6.13 wird der ASW-Programmablauf des Demonstrators erklärt. Aufgrund der Länge der generierten Methodennamen werden im Diagramm Abkürzungen verwendet.
Im Sequenzdiagramm werden drei Tasks, ein Alarm, die RTE und eine Positions-Klasse dargestellt. Der "ASWATask0" wird zyklisch von einem Alarm aktiviert und ist für die Kommunikation zur RTE verantwortlich. Von diesem Task aus wird, abhängig vom gesetzten Modus, entweder der "ASWTask1" für den manuellen Modus oder der "ASWTask2" für den automatischen Modus aktiviert. Diese beiden Tasks kommunizieren mit der Positions-Klasse und berechnen die zu setzende Drosselklappenposition bevor sie an den wartenden "ASWTask0" zurück geben. Dieser holt sich abschließend den aktuellen Positionswert aus der Spinlock geschützten Positions-Klasse und gibt ihn an die RTE weiter.

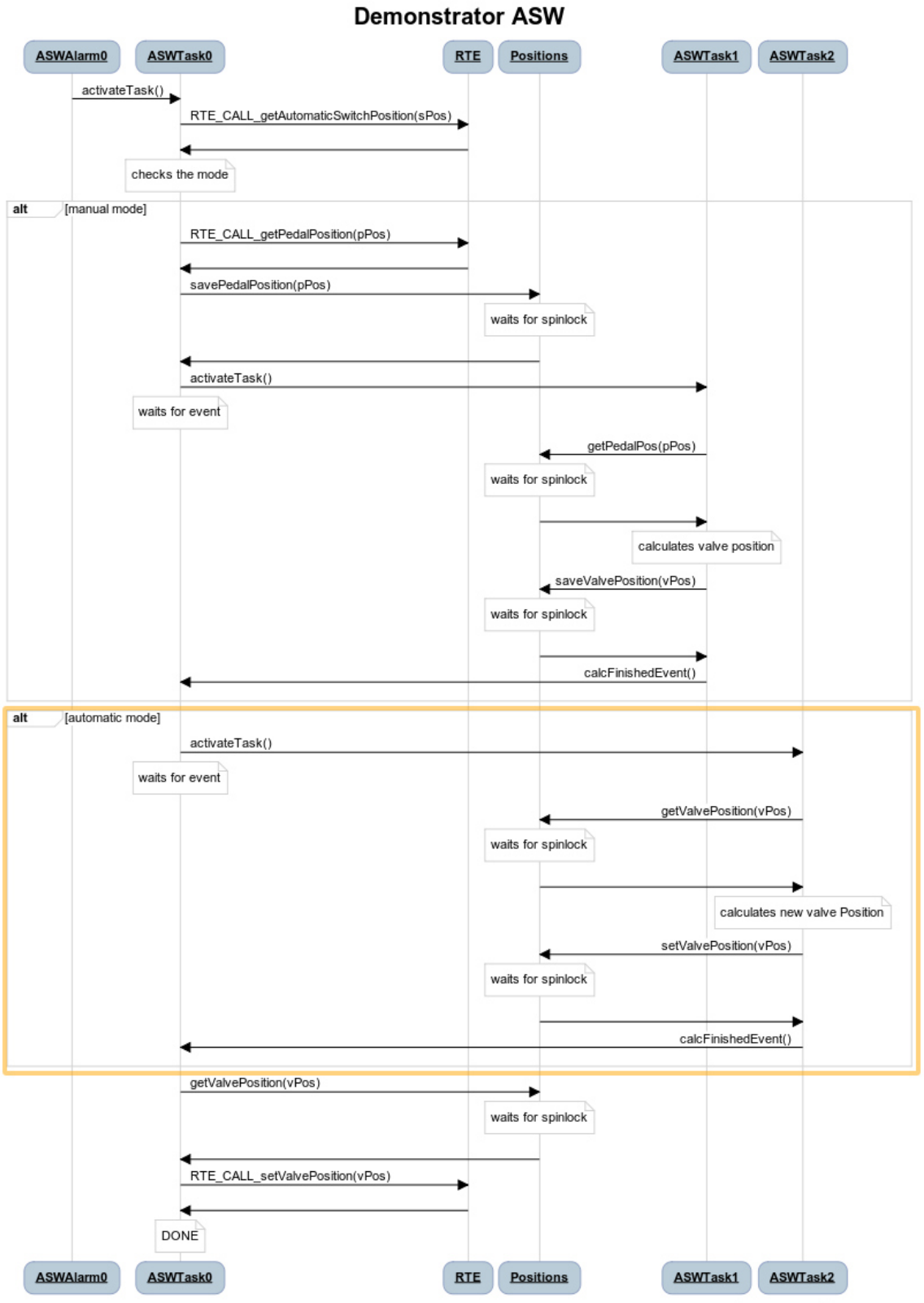

Abbildung 6.13: Demonstrator Sequenzdiagramm

6.4 Demonstrator Funktionstests

Zur Evaluierung der Funktion des Projekttemplates sowie des Demonstrators werden, für die verschiedenen verwendeten Komponenten, Funktionstests implementiert. Das im Rahmen dieser Arbeit entstandene Projekttemplate implementiert eine Standardkonfiguration. Im Zuge der Integration eines realen Systems, muss diese im Bereich der Konfigurationsparameter sehr stark angepasst werden. Daher soll lediglich die generelle Funktionsfähigkeit, nicht jedoch die Ausführungszeiten oder das Langzeitverhalten des Projekttemplates geprüft werden.

- Alarm & Task Test

  Es wird ein Demo Task erstellt, welcher von einem Alarm in zyklischen Abständen von 250ms aufgerufen wird. In der Quellcodedatei werden zwei Funktionalitäten implementiert. Erstens wird bei jedem Aufruf ein Zähler inkrementiert. Zweitens wird, abhängig davon ob der Zähler gerade oder ungerade ist, eine verbundene Leuchtdiode aktiviert oder deaktiviert.

  Das Programm wird für exakt 15 Minuten, auf dem "AURIX TriCore" ausgeführt. In dieser Zeit ist ein rhythmisches Blinken der Diode sichtbar. Nach Ablauf der Testzeit, wird mit Unterstützung des "Trace32"-Tools der Wert des Zählers ausgelesen. Der Ergebniswert wird, unter Berücksichtigung der Messtoleranz, mit dem Sollwert verglichen.
  Der Demonstrator bestand den Test erfolgreich.

- CAN-In Test

  Es wird ein CAN-Netzwerk, inklusive RTE-Interface aufgesetzt. Anschließend wird ein Demo Task erstellt, welcher von einem Alarm in zyklischen Abständen von 20ms aufgerufen wird. In der Quellcodedatei wird der Wert mittels Interface Aufruf aus der RTE ausgelesen und anschließend in eine globale Variable der ASW gespeichert.

  Das Programm wird auf dem "AURIX TriCore", an dem der "CANalayzer" angeschlossen ist, ausgeführt. Mit dem "CANalyzer" werden fortlaufend unterschiedliche CAN-Signale an den "TriCore" gesendet. Das "Trace32"-Tool überwacht die Änderungen der zuvor definierten globalen Variable. Die neuen Werte sollen zyklisch in diese Variable geschrieben werden, abhängig von den Werten die der "CANalyzer" überträgt.
  Der Demonstrator bestand den Test erfolgreich.

- CAN-Out Test

  Der Versuchsaufbau ist ähnlich wie beim "CAN-In Test" und unterscheidet sich durch schreibende Zugriffe.

Der Demo Task schickt im 20ms-Zyklus verschiedene CAN-Signale. Die eintreffenden Signale werden im "CANalayzer" überwacht und mit den Sollwerten des Tasks verglichen.
Der Demonstrator bestand den Test erfolgreich.

- CAN-Timing Test

  Der Versuchsaufbau entspricht dem des "CAN-Out Tests".

  Mit dem "CANalayzer" ist es möglich die genauen Zeitabstände zwischen den eintreffenden Nachrichten zu messen. Ob der Zeitabstand mit dem definierten Zyklus von 20ms übereinstimmt wird dabei verglichen.
  Der Demonstrator bestand den Test erfolgreich.

- ADC Test

  Es wird ein analoges Gaspedal an die Steuerplatine angeschlossen. Das ADC-Modul wird so konfiguriert, dass es den Analogwert ab dem Systemstart zyklisch in einen Digitalwert konvertiert.

  Das "Trace32"-Tool überwacht die Änderungen des digitalen Werts im Programm. Es werden Maximum- und Minimumwerte wiederholt ausgelesen und verglichen.
  Der Demonstrator bestand den Test erfolgreich.

- Mehrkern Test

  Es werden zwei gleiche Demo Tasks erstellt, welche von zwei Alarmen in zyklischen Abständen von 250ms aufgerufen werden. In der Quellcodedatei werden je zwei Funktionalitäten implementiert. Erstens wird bei jedem Aufruf ein Zähler inkrementiert. Zweitens wird, abhängig davon ob der Zähler gerade oder ungerade ist, die zum Task gehörende Leuchtdiode aktiviert oder deaktiviert. Dabei werden beide Tasks auf unterschiedlichen Kernen ausgeführt.

  Das Programm wird auf dem "AURIX TriCore" ausgeführt. Mit dem "Trace32"-Tool wird ein Kern während der Laufzeit angehalten. Ab diesem Zeitpunkt darf ein Taskzähler nicht mehr inkrementiert werden, und die zugehörige Leuchtdiode nicht mehr blinken. Dies darf allerdings keine Auswirkungen auf den Zähler und die Leuchtdiode des Tasks auf dem anderen Kern haben. Durch diesen Versuchsaufbau wird die Unabhängigkeit der beiden Tasks vom jeweils anderen Kern gezeigt.
  Der Demonstrator bestand den Test erfolgreich.

- Spinlock Test

  Es werden zwei gleiche Demotasks erstellt, welche von zwei Alarmen in zyklischen Abständen von 250ms aufgerufen werden. In der Quellcodedatei werden zwei Funktionalitäten implementiert. Es wird auf die globale Funktion "setPosition(int)" zugegriffen. Diese Funktion holt sich den Spinlock "spin0", ändert anschließend eine globalen Variable und gibt "spin0" frei. Die zweite Funktionalität ist ein Zähler der nach jedem Aufruf der "setPosition(int)" Funktion inkrementiert wird. Es werden beide Tasks auf unterschiedlichen Kernen ausgeführt.

  Das Programm wird auf dem "AURIX TriCore" ausgeführt. Mit dem "Trace32"-Tool wird der Kern, auf welchem der Task ausgeführt wird, welcher "spin0" blockiert, angehalten. Der andere Task wird dadurch indirekt blockiert, da er in der "setPosition(int)" Funktion warten muss, bis "spin0" frei gegeben wird. Aus diesem Grund wird sich der Zähler bei keinem der Tasks mehr erhöhen.
  Der Demonstrator bestand den Test erfolgreich.

6.5 Kundenprojekt Integration

Im Kontext dieser Arbeit wurde bisher ein Demonstrator integriert, konfiguriert und somit die Funktionsfähigkeit des Projekttemplates validiert. Der zügige und effiziente Integrationsprozess des Demonstrators, gepaart mit den erfolgreichen Funktionstests, hat die hohe Qualität und Effizienz der aufgesetzten Toolchain bewiesen. Die bis dato geleistete Arbeit und das erworbene Know-How dienen nun als Basis für die Integration eines firmeninternen Kundenprojekts, welches im Vergleich zum bereits integrierten Demonstrator ein Großprojekt darstellt.
Für die Integration eines mehrkernfähigen Großprojekts, welches unter nicht AUTOSAR konformen Umständen entwickelt wurde, fallen nun einige zusätzliche, mitunter sehr aufwendige, Arbeitsschritte an. Die ASW dieses Projekts verfügt über hunderte Funktionen, ca. 500 Eingangs- und Ausgangssignale, mehrere Buscontroller und verschiedene ADC und PWM-Konfigurationen. Einen großen Teil des Projekts stellt die Analyse der Abhängigkeiten zwischen den benötigten ASW-Tasks dar. Diese bieten die Grundlage für eine effiziente Verteilung der Tasks auf die verschiedenen Kerne. Zusätzlich existieren noch eine große Anzahl an zeitkritischen Voraussetzungen, verteilten Ressourcen, unterschiedlichen Systemmodi und Fehlerbehandlungssysteme, welche alle über die BSW-Konfiguration gesteuert werden müssen. Da dieses Projekt ursprünglich in einer nicht AUTOSAR konformen Entwicklungsumgebung entstanden ist, existieren dazu bisher keine Konfigurationsdateien, welche eine automatische Konfiguration ermöglichen würden.

Die erste Revision der Integration dieses Projekts wurde schrittweise umgesetzt und bestätigte die bisherige Implementierung fortlaufend. Das vorkonfigurierte Projekttemplate ließ sich, zusammen mit dem erlangten Wissen zur Funktionalität der Toolchain, den Standards und den notwendigen Arbeitsschritten, als zuverlässige Basis für die erfolgreiche Großprojektintegration verwenden.
Nachfolgend werden einige Beispiele für die starke Vereinfachung der Migration erklärt. Als Paradebeispiel kann man die knapp 400 CAN-Signale heranziehen, welche durch die ca. 14000 Zeilen umfassende, automatisch generierte ARXML-Datei in der RTE zugreifbar gemacht wurden. Darauf aufbauend konnten sämtliche CAN-Skalierungsberechnungen und CAN-Hardwarezugriffe aus der ASW entfernt werden und durch die entsprechenden RTE-Aufrufe ersetzt werden. Weiters trug das vorkonfiguriert Projekttemplate dazu bei, dass die CAN-Controller Konfiguration weitgehend durch das Importieren der DBC-Datei abgeschlossen werden konnte. Die verschiedenen Tasks mussten nur noch den Kernen zugeordnet werden und die ADC- und PWM-Treiber mussten lediglich noch auf das Einsatzgebiet angepasst und mit den relevanten RTE-Interfaces verbunden werden.

Kapitel 7

Bewertung der Erfolgsfaktoren

Dieses Kapitel nutzt die gewonnen Erfahrungen, um das generelle Konzept und das Framework zu Bewerten, mit dem Ziel einen zusätzlichen Mehrwert für zukünftige Projekte zu generieren.
Für die Umsetzung eines AUTOSAR konformen Projekts stehen am Markt zahlreiche Tools und Toolchains zur Verfügung. Im Verlauf dieser Arbeit wurden verschiedene Tools zu einer effektiven Toolchain abgestimmt und mit dieser eine grundlegende Basisstruktur für die Integration von nicht AUTOSAR konformer ASW erstellt. Im Rahmen der Entwicklung und Demonstrator-Migration wurden weitere Tools benötigt und erstellt. Nach Abschluss der Funktionstests des Demonstrators, zur Evaluierung des Projekttemplates und den ersten Eindrücken in der Großprojektintegration, kann man ein erstes Fazit ziehen. Das erstellte Projekttemplate und die verwendeten Tools stellen, in Kombination mit dem erworbenen Know-How, eine ideale Basis für die Erstellung von serienproduktionsnahen Fahrzeugsystemen dar. Dieser modulare Entwicklungsansatz verspricht eine sehr hohe Wiederverwertbarkeit, Austauschbarkeit von einzelnen Modulen und ein enormes Skalierungspotential.
Die Evaluierung der entstandenen Implementierung wurde bereits in Kapitel 6 behandelt. In diesem Kapitel wird das entwickelte Konzept, das erarbeitete Know-How und die verwendete Toolchain anhand verschiedener Kriterien bewertet. Die Beurteilungsgruppen umfassen folgende Kriterien: Vorbedingungen, Anbieter, Kosten, Standards, Anwendungsgebiete, Benutzerfreundlichkeit, Konfigurierbarkeit sowie unterstützende Tools. Bei der nachfolgenden Bewertung wird nicht nur auf das durchgeführte Projekt, sondern auch auf die Marktsituation und die Toolchain eingegangen. Dieses Vorgehen ist wesentlich, da für Projekte in der Fahrzeugindustrie die unterschiedlichsten Vorgaben und Ziele definiert sind. Somit existiert kein universeller Entwicklungsansatz und ein genauer Einblick in die Projektumgebung kann die Definition zukünftiger Projekte stark erleichtern.

7.1 Vorbedingungen

Neben der entstandenen Softwareimplementierung ist die Einarbeitung in die Automotive Domäne ein wesentlicher Projektbestandteil. Um die notwendigen Zusammenhänge zu verstehen und die Entwicklungsumgebung zu beherrschen, reichen Programmierfähigkei-

ten alleine nicht aus. Die Auseinandersetzung mit den wesentlichen Industriestandards, verschiedenen Werkzeugen zur Evaluierung und Buskommunikationssystemen war von großer Bedeutung.

Programmierfähigkeit	Sämtliche der, von "Elektrobit Tresos" bereitgestellten, BSW-Module sind in der C-Programmiersprache implementiert. Daher sind sehr gute Fähigkeiten in diesem Bereich unabdingbar. Um das ARXML-Datenaustauschformat zu verstehen sind grundlegende Kenntnisse von XML erforderlich. Der damit verbundene ARXML-Generator wurde in der C#-Programmiersprache erstellt.
Automotive Domäne	Das durchgeführte Projekt beschäftigte sich intensiv mit Hardware, Software, Standards, einer Entwicklungsumgebung und den Systementwicklungsprozessen welche in der Automotive Industrie üblich sind. Daher ist die Einarbeitung in diese komplexe Systemumgebung, sowie in die Welt der eingebetteten Softwareentwicklung unverzichtbar.
AUTOSAR	Die Grundvoraussetzung, um die gegenständliche Toolchain und modulare Projektumgebung zu verstehen, ist ein vollständiges Wissen der AUTOSAR-Architektur. Der modulare Aufbau und die standardisierten Interfaces bieten die Basis für die stark erhöhte Wiederverwertbarkeit, Wartbarkeit und Skalierbarkeit. Bereits bei der Konfiguration der BSW ist es wichtig die Unterschiede und Möglichkeiten der einzelnen AUTOSAR-Schichten umsetzen zu können. Vor allem die Migration eines nicht AUTOSAR konformen Softwareprojekts erfordert viele Änderungen, welche ohne fortgeschrittene AUTOSAR-Erfahrung schwer durchgeführt werden können.
Toolchain	Die Entwicklung eines AUTOSAR-Systemprojekts setzt genaue Kenntnisse über die verwendete Toolchain voraus. Es wird eine Entwicklungsumgebung für die Softwareentwicklung, ein entsprechend dimensionierter Mikrocontroller, ein kompatibler Compiler, sowie Werkzeuge zum Flashen der Software und Debuggen des Systems und der Buskommunikation benötigt.

Elektrobit Tresos	Im Kontext dieser Arbeit stellt, das auf dem AUTOSAR-Standard basierende "Elektrobit Tresos Studio", die Hauptentwicklungsumgebung für die Softwarekomponenten dar. Es dient zur Konfiguration der BSW, zum Aufsetzen der Systemvariablen und Integrieren von ASW. Als grafische Entwicklungsumgebung unterstützt es bei der Konfiguration, zeigt ungültige Kombinationen auf und generiert das erstellte System. Das "Elektrobit Tresos AutoCore"-Paket beinhaltet dabei alle notwendigen Module für ein vollständiges Projekt. Aus diesen Vorteilen ergibt sich eine erhebliche Einarbeitungszeit in die notwendigen Konfigurationsabläufe, den existierenden Pool an Fehlermeldungen sowie in das bereitgestellte Arbeitsinterface.
Kommunikationsprotokolle	Das Verständnis für die Funktionsweise der internen Systemkommunikation ist ebenso erforderlich wie jenes für die externe Buskommunikation. Das Hauptaugenmerk liegt auf dem CAN-Bus, welcher sehr umfangreich umgesetzt wurde und die Drosselklappe des Demonstrators steuert.

Tabelle 7.1: Notwendiges Know-How

Vor Beginn eines solchen Projekts ist es entscheidend sich entsprechendes Vorwissen bezüglich AUTOSAR und der zu verwendenden Toolchain anzueignen. Die Vorteile der Flexibilität, Wiederverwertbarkeit, Austauschbarkeit und Wartbarkeit eines AUTOSAR konformen Softwareprojekts werden durch unsachgemäßes Vorgehen minimiert.

7.2 Anbieter

Ein stabiler Partner ist bei der Entwicklung eines Softwareprojekts für die Serienproduktion entscheidend. Langjährige Markterfahrung und Entwicklungskooperationen mit den Marktteilnehmern der anderen Automotive Sparten, sind Grundvoraussetzung für eine qualitative Entwicklung. Eine stetige Weiterentwicklung der verwendeten Werkzeuge, sowie geringe Einschränkungen in Bezug auf die Kompatibilität und Möglichkeiten zur Zertifizierung sind ebenfalls wesentlich für ein Projekt in diesem Kontext.
Die nachfolgend aufgelisteten Bewertungskriterien sind unabhängig von einem spezifischen Anbieter gewählt. Aufgrund der Projektvorgaben wurden die Bewertungen anhand von "Elektrobit" durchgeführt.

Elektrobit [20]

Die Grundlage für den in dieser Arbeit verfolgt Softwareentwicklungsansatz, bildet eine von der Firma "Elektrobit" bereitgestellte Entwicklungsumgebung.

Entwicklung	Die "Elektrobit Automotive Group" wurde 1985 gegründet und ist seither in der Fahrzeugindustrie tätig. "Elektrobit" wurde 2015 vom "Continental-Konzern" übernommen. Aktuell werden über 1400 Mitarbeiter in über acht Ländern beschäftigt.
Marktanteil	"Elektrobit" hat sich in der Serienproduktion von Fahrzeugen weltweit etabliert. Über 70 Millionen Fahrzeuge sind mit ihren eingebetteten Systemen ausgestattet.
Produktportfolio	Neben dem in dieser Arbeit verwendeten "Elektrobtit Tresos", werden noch viele weitere Produkte angeboten. Es existieren Lösungen zur Fahrzeuginfrastruktur, Mensch-Maschine Kommunikation, Navigationsservices, Fahrassistenzsysteme, ECUs und generelle Entwicklungsservices.
Partner	"Elektrobit" pflegt enge Partnerschaften mit vielen Zulieferern und Fahrzeugherstellern am Markt und ist als Premium Mitglied im AUTOSAR-Konsortium tätig. Dies ermöglicht den Zugriff und die rasche Umsetzung der neuesten Hardware und Software Standardspezifikationen.

Tabelle 7.2: Eigenschaften von "Elektrobit" [20]

Elektrobit Tresos [21]

Das "Elektrobit Tresos Studio" ist eine Eclipse-basierte Entwicklungsumgebung für AUTOSAR konforme Softwareentwicklungen. Es vereint den Systemerstellungsprozess mit dem Systemkonfigurationsprozess, sowie dem Generierungsprozess. Im Softwareentwicklungsprozess erleichtert das Werkzeug die Implementierung und Konfiguration von Softwareprojekten, durch die Bereitstellung von Entwicklungsunterstützungen und Validierungsmechanismen. Die eingesetzten BSW-Module stammen aus dem zugehörigen Paket "Elektrobit Tresos AutoCore", welches mit den neuesten AUTOSAR-Definitionen kompatibel ist.

Entwicklung	"Elektrobit Tresos Studio" und "Elektrobit Tresos AutoCore" werden analog zu AUTOSAR weiterentwickelt. Die aktuellen Softwareversionen unterstützen die AUTOSAR-Versionen 4.x, 3.x sowie alle gängigen Buskommunikationsprotokolle. Durch die Zusammenarbeit mit führenden Hardwarezulieferern, sowie dem AUTOSAR-Konsortium, finden ständig Aktualisierungen statt.
Qualitätssicherung	Um die Qualität der angebotenen Software zu gewährleisten, werden zyklisch über 14000 Tests auf 20 unterschiedlichen Architekturen durchgeführt. Diese Tests erfüllen die höchsten Qualitätsstandards für die Serienproduktion und decken 100 Prozent des Programmcodes der verkauften Software ab.
Funktionen	Das "Elektrobit Tresos AutoCore"-Paket beinhaltet das RTE-Modul sowie auch sämtliche benötigten BSW-Module, inklusive dem mehrkernfähigen "AutoCore-OS" RTOS. Zu den BSW-Modulen gehören Kommunikationsservices, Diagnosestacks, Interfacemodule und auch die MCAL-Module der verschiedenen Mikrocontroller Hersteller. Das "Elektrobit Tresos Studio" bietet einen Konfigurationseditor, Funktionen zur Validierung der Konfiguration, die Möglichkeit Konfigurationen zu importieren und exportieren, unterstützende Mechanismen zum Arbeitsablauf und die Generierung von Programmcode.
Aufbau	Das "Elektrobit Tresos Studio" ist als flexible grafische Entwicklungsumgebung aufgebaut. Es handelt sich um eine Eclipse-basiertes Tool welches das Importieren, Exportieren und Erweitern von Konfigurationen stark erleichtert. Bei "Elektrobit Tresos AutoCore" handelt es sich um ein modular aufgebautes BSW-Modulpaket. Die einzelnen Module werden regelmäßig nach den neuesten AUTOSAR-Standards updatet und sind untereinander kompatibel. Gemeinsam bieten diese beiden Grundwerkzeuge einen uneingeschränkten Zugriff auf Quelltext und Konfigurationen.
Lizenzierung	Es werden unterschiedlichste Lizenzierungsmodelle angeboten, welche sämtliche Bedürfnisse eines Projekts in der Fahrzeugindustrie abdecken. Von der kostengünstigeren Evaluierungslizenz bis hin zu maßgeschneiderten Systemkonfigurationen, für die Serienproduktion ist alles erhältlich.

Systemanforderungen	Bei der verwendeten Software handelt es sich ausschließlich um eine Eclipse-basierte Entwicklungsumgebung und einem Modulpaket. Voraussetzung für die Nutzung, sind ein kompatibler Mikrocontroller und ein Werkzeug zum Flashen der ausführbaren Datei. Der Entwickler benötigt dazu lediglich einen durchschnittlichen Windows Computer.
Hardware / Software Einschränkungen	Das verwendete Softwaresystem basiert auf AUTOSAR und ist deshalb mit vielen gängigen Werkzeugen der Fahrzeugindustrie kompatibel. Der modulare Aufbau führt dazu, dass für eine Hardwareänderung lediglich einzelne Module, meist der MCAL, ausgetauscht werden müssen. Auch die BSW-Module, einschließlich dem "AutoCore-OS" selbst, sind aufgrund der Standardisierung völlig austauschbar und abwärtskompatibel. Bezüglich der Kommunikationsbusunterstützung, werden sämtliche gängigen Protokolle angeboten.

Tabelle 7.3: Eigenschaften von "Elektrobit Tresos" [21]

Bei "Elektrobit Tresos" handelt es sich um eine vollständige Entwicklungsumgebung, inklusive der notwendigen BSW-Module, welche alle Prozesse vom Beginn des Entwicklungsprozesses über Unterstützungswerkzeuge bei der Konfiguration bis hin zur Erstellung der ausführbaren Programmdatei in sich vereint. Daher ist dieses Werkzeug eine ideale Grundlage für ein serienproduktionsnahes Projekt in der Fahrzeugindustrie.

7.3 Kosten, Verfügbarkeit und Service

Insbesondere bei der Softwareentwicklung für die Serienproduktion sind eine qualitative Entwicklungsumgebung und aktuelle Softwarepakete Pflicht, um Effizienz und professionelle Resultate zu erreichen. Allerdings ist der Wert eines Softwareentwicklungssystems nicht nur anhand der technischen Details zu beziffern. Die Verfügbarkeit von Entwicklerunterstützung, Dokumentation, Schulungen und angebotene Trainings sind notwendige Voraussetzungen für einen effizienten Entwicklungsprozess.
Im Rahmen dieser Arbeit wurden Evaluierungslizenzen verwendet. Die Anschaffungskosten hängen von der gewählten Systemkonfiguration ab. Daher lässt sich keine Auskunft über die tatsächlichen Kosten geben. Würden die Werkzeuge in der Serienproduktion eingesetzt werden, wären, abhängig vom gewählten Lizenzmodell, sehr wohl Anschaffungskosten und Supportkosten zu leisten. Der Hersteller bietet sowohl Schulungen, Trainings, als auch Unterstützungen quer durch den ganzen Entwicklungsprozess an.

7.4 Standards

Standardisierung ist ein wesentlicher Bestandteil der heutigen Fahrzeugindustrie. Dies gilt vor allem, wenn die Vorteile von Austauschbarkeit, Wartbarkeit, Wiederverwendbarkeit und Skalierbarkeit genutzt werden sollen. Mit AUTOSAR wurde ein Standard entwickelt, der diese Vorteile vereint. Die Zusammenarbeit bezüglich der Standardisierung wird zum starken Qualitätsvorteil gegen unbeteiligte Unternehmen. In der Serienproduktion kommt es allerdings nicht nur auf Portierbarkeit und Integrierbarkeit an, sondern auf die Unterstützung von verschiedenen Sicherheitsstandards.

Qualitätsstandard	Die Toolchain muss auf höchste Qualitätsstandards setzen, um für den Einsatz in einer industriellen Serienproduktion geeignet zu sein.
Safety	Die eingesetzte Entwicklungsumgebung, sowie die BSW-Module müssen mit den neuesten Definitionen der ISO 26262 konform sein und sich bis ASIL-D zertifizieren lassen.
AUTOSAR	AUTOSAR muss die Basis für die verwendete Toolchain sein. Zusätzlich müssen laufend neue Aktualisierungen des Standards eingepflegt werden.
Konformitätsklassen	Es müssen verschiedene "AUTOSAR Scalability Classes" unterstützt werden.
Implementierungssprache	Die zugrunde liegende Software sollte unter Berücksichtigung aktueller Standards, zum Beispiel "MISRA-C" implementiert sein.
Entwicklungsprozess	Es müssen standardisierte Entwicklungsprozesse, wie zum Beispiel der Automotive-SPICE Standard unterstützt werden.

Tabelle 7.4: Standards

Der Entwicklungspartner "Elektrobit" erfüllt alle oben angeführten Kriterien. Es werden höchste Qualitätsstandards angestrebt und eine ständig Aktualisierungen der verwendeten Werkzeuge auf Basis der neuesten AUTOSAR-Definitionen durchgeführt.

7.5 Produktreifegrad

In der Fahrzeugindustrie gibt es unterschiedlichste Entwicklungsszenarien. Je nach Vorgaben und Zielen des Systems sind unterschiedliche Entwicklungsprozesse notwendig um

ein adäquates System zu erhalten. Für die Erstellung von Prototypen wird, zum Beispiel kaum Wert auf Zertifizierbarkeit und standardisierte Abläufe gelegt. Dafür müssen sich allerdings unter geringen Kosten schnelle Ergebnisse präsentieren lassen. Im Zuge dieses Projekts wurde die gegenständliche Toolchain im Hinblick auf die Serienproduktion evaluiert. In diesem Kontext treten andere Faktoren in den Vordergrund. Die entstehende Lösung muss den aktuellen Industriestandards und Sicherheitszertifizierungen genügen. Komponenten müssen austauschbar und wiederverwendbar sein. Außerdem muss entsprechende Entwicklungsunterstützung und gute Dokumentation vorhanden sein.

Einarbeitungszeit	Neben der aktiven Unterstützung bei der Softwareentwicklung und Modulkonfiguration, bietet die verwendete Toolchain auch Validierungsmechanismen an und vereint alle Schritte des Entwicklungsprozesses in einem Werkzeug. Zusätzlich werden noch eine Vielzahl an Demonstrationsprojekten, sowie Beschreibungen zu Arbeitsabläufen angeboten. Somit muss, neben der Einarbeitungszeit in den AUTOSAR-Standard und die damit einhergehenden Definitionen der BSW-Module, auch eine erhebliche Eingewöhnungszeit in die Toolchain einkalkuliert werden.
Produktivität	Ist der Einstieg in die AUTOSAR-Thematik und die innovative Entwicklungsumgebung erst einmal geschafft, ermöglicht die verwendete Toolchain eine äußerst effiziente Systementwicklung. Aufgrund der unterstützenden Konfigurationsvalidierung können ausschließlich semantisch korrekte Systeme erstellt werden. Dies verhindert langwieriges Fehlersuchen und erhöht die Produktivität. Der modulare Softwareaufbau ermöglicht im späteren Entwicklungsprozess ein rasches und unkompliziertes Austauschen und Erweitern des erstellten Systems.
Anwendungsgebiet	Um ein überdurchschnittlich hohes Maß an Produktivität und Professionalität zu garantieren, wird eine längere Einarbeitungszeit benötigt. Daher eignet sich dieser Entwicklungsansatz vor allem für die Serienproduktion von Fahrzeugsystemen.

Weiterentwicklung	Die Herstellerfirma der "Elektrobit Tresos"-Werkzeuge ist Premium Mitglied im AUTOSAR-Konsortium und pflegt enge Partnerschaften zu bedeutenden Hardwareherstellern der Fahrzeugindustrie. Aus diesem Grund unterliegt die verwendete Software den neuesten AUTOSAR-Standards und unterstützt die aktuellsten Hardwareplattformen. Daher kann man davon ausgehen, dass diese Entwicklung auch in Zukunft stattfindet.
Mehrkerntauglichkeit	In der heutigen Zeit sind Mehrkernsysteme auch in der Fahrzeugindustrie nicht mehr wegzudenken. Daher wurde auch das gegenwärtige Projekt in einer mehrkernkompatiblen Projektumgebung durchgeführt. Die Herstellerfirma bietet neben einem Einkernbetriebssystem- auch noch ein Mehrkernbetriebssystem-Modul an.
Sicherheitsmechanismen	Obwohl aktuelle Sicherheitsstandards nicht Thema dieser Arbeit sind, haben sie in der Industrie einen hohen Stellenwert. Die Herstellerfirma stellt zusätzlich ein "Safety OS" zur Verfügung.
Mikrocontroller Architektur	Das Einsatzgebiet von "Elektrobit Tresos" ist hinsichtlich der vorgegebenen Architekturen sehr weitläufig. Beispiele für die Zielarchitekturen sind: PowerPC, ARM (Cortex), Renesas (V850), Freescale und TriCore.

Tabelle 7.5: "Elektrobit Tresos"-Anwendungsgebiet

Wie aus der Tabelle 7.5 ersichtlich, ist "Elektrobit Tresos" ein ideales Werkzeug für Projekte in der Serienproduktion.

7.6 Benutzerfreundlichkeit

Benutzerfreundlichkeit ist von hoher Wichtigkeit für den Softwareentwickler. Als benutzerfreundlich wird dabei ein System definiert, welches eine hohe Nutzungsqualität in der Interaktion mit dem System gewährleistet. Intuitive Bedienbarkeit, klar wahrnehmbare Strukturen und Abläufe, sowie der Verzicht auf verwirrende Darstellungen und Menüführungen zählen zur Basis dieses Kriteriums.
Vor allem in der Einarbeitungszeit kann man somit viel Zeit und Frust ersparen. Dieses Kriterium wird nachfolgend für die drei Punkte, nämlich Entwicklungsumgebung, Entwicklungssupport und Dokumentation bewertet.

7.6.1 Entwicklungsumgebung: Tresos

Die Entwicklungsumgebung "Elektrobit Tresos Studio" und das BSW-Modulpaket "Elektrobit Tresos AutoCore" gestaltet, dank der benutzerfreundlichen Menüführung, den zahlreichen Arbeitsassistenten und Demonstrationsprojekten, auch für Einsteiger in die Fahrzeugsoftwareentwicklung ein benutzerfreundliches Entwicklungswerkzeug.

Zusatzfunktionen	"Elektrobit Tresos Studio" ist eine Eclipse-basierte Entwicklungsumgebung, für die entsprechende Zusatzfunktionen existieren. Ein Beispiel dafür ist die enthaltene SVN Versionsverwaltung. Die Entwicklungsumgebung unterstützt maßgeblich beim Konfigurationsprozess. Es werden Standardwerte vorgeschlagen, Konfigurationen validiert, Einstellmöglichkeiten dokumentiert und der gesamte Entwicklungsprozess in ein Werkzeug vereint.
Schnittstellen	Da die eingesetzte Toolchain auf dem AUTOSAR-Standard basiert, ist sie mit den meisten gängigen Werkzeugen in der Fahrzeugindustrie kompatibel. Sowohl der Import und Export von Konfigurationen und Projekten, als auch die Wiederverwertbarkeit von erstellten Komponenten gehört zu den Standardfunktionen des Tools.
Updates	Die Versionen von "Elektrobit Tresos" sind stark an die AUTOSAR-Veröffentlichungen angelehnt. Um laufend über die aktuellsten Module zu verfügen, werden neue Versionen der BSW unabhängig vom Gesamtpaket veröffentlicht.

Tabelle 7.6: "Elektrobit Tresos"-Benutzerfreundlichkeit

7.6.2 Entwicklungssupport

Zur Benutzerfreundlichkeit gehört nicht nur der Programmaufbau und die Bedienbarkeit, sondern in der heutigen Zeit auch immer mehr der Entwicklungssupport im gesamten Projektzyklus. In der Einarbeitungszeit können Schulungen, Trainings und Demonstrationsprojekte von entscheidender Bedeutung sein. Allerdings kann es auch im laufenden Industrieprojekt zu Unklarheiten kommen. In dieser Phase ist meist keine Zeit zu verlieren und der Hersteller muss kurzfristig entsprechend geschultes Personal bereitstellen können.

Schulungen	Die Firma "Elektrobit" bietet laufend alle benötigten Trainings und Schulungen an. Zusätzlich gibt es noch die Möglichkeit diese selbst zu vereinbaren. Beginnend bei AUTOSAR über die Buskommunikation bis hin zu gezielten Produktschulungen ist alles vorhanden.
Support im Entwicklungsprozess	Je nach Lizenzmodell gibt es unterschiedliche Unterstützungsmöglichkeiten während der laufenden Entwicklung. In jedem Fall steht ein Kundenportal für schriftliche Anfragen sowie eine Telefonhotline zur Verfügung.

Tabelle 7.7: "Elektrobit Tresos"-Entwicklersupport

Gerade beim Support in der Projekteingangsphase kann die verwendete Entwicklungsumgebung "Elektrobit Tresos Studio" sehr stark punkten. Über den Support im Softwareentwicklungsprozess können im Kontext dieser Arbeit keine Angaben gemacht werden, da lediglich eine Evaluierungslizenz benutzt wurde.

7.6.3 Dokumentation

In einer komplexen Domäne, wie der Fahrzeugindustrie, reicht intuitive Bedienbarkeit und ausreichender Support durch den Hersteller bei weitem nicht aus, um effizient zu qualitativen Ergebnissen zu kommen. Eine umfangreiche und qualitative Dokumentation ist die Grundlage für eine gut einsetzbare und verständliche Software.

Verfügbarkeit	Die Dokumentation von "Elektrobit Tresos Studio" wird mit der Entwicklungsumgebung mitgeliefert und ist sowohl in PDF-Form, als auch im Tool selbst interaktiv integriert. Hinsichtlich des BSW-Pakets "Elektrobit Tresos AutoCore" verfügt jedes Modul über mindestens eine eigene Dokumentationsdatei inklusive einem Änderungsverlauf. Die Dokumentation wird zusammen mit der Software selbst ausgeliefert.
Vollständigkeit	Die Dokumentation von "Elektrobit Tresos" ist für den weitläufigen Einsatz der Software aufgebaut und generell gehalten. Zusätzlich existieren noch spezifische Anwendungsbeispiele und Demonstrationsprojekte. Im Zuge der Systementwicklung dieser Arbeit, ist die Dokumentation durch Vollständigkeit und gute Strukturierung aufgefallen.

Aktualität	Sämtliche Dokumentationen sind auf dem neuesten Stand und enthalten zusätzlich einen Änderungsverlauf der Updates.
Mehrsprachigkeit	Die im Zuge dieser Arbeit verwendete Dokumentation ist in englischer Sprache gehalten.
Fehlerdokumentation	Das "Elektrobit Tresos Studio" bietet eine enorme Anzahl von unterstützenden Funktionen und Konfigurationsparametern an. Aus diesem Grund gibt es viele unterschiedliche Fehlermeldungen, mit denen man sich im Laufe des Entwicklungsprozesses auseinandersetzen muss.
Quellcode	Der Quellcode der BSW-Module ist vorbildlich dokumentiert und nach dem "MISRA-C"-Standard implementiert.
Demos / Arbeitsabläufe	In den unterschiedlichen Modul-Dokumentationen stehen eine enorme Anzahl an praxisnahen Implementierungen zur Verfügung. Die Entwicklungsumgebung bietet verschiedene Kataloge von vorgegebenen Arbeitsschritten an, welche als Unterstützung bei der Konfiguration dienen können. Zusätzlich existieren noch unterschiedliche Demonstrationsprojekte, welche in die Entwicklungsumgebung importiert werden können. Letztere sind allerdings eher undurchsichtig gehalten.

Tabelle 7.8: "Elektrobit Tresos"-Dokumentation

Während des gesamten Entwicklungsprozesses dieser Arbeit waren die verfügbaren Dokumentationen zu AUTOSAR, "Elektrobit Tresos" sowie dem eingesetzten Mikrocontroller die ersten Anlaufstellen. Sowohl zur Entwicklungsumgebung als auch zu den eingesetzten BSW-Modulen existiert eine hochqualitative, vollständige und aktuelle Dokumentation. Um den Einstieg in die Entwicklung zu erleichtern, werden Arbeitsabläufe vorgegeben, Integrationsbeispiele erklärt sowie Demonstrationsprojekte angeboten.

7.7 Konfiguration

Es existieren mehrere Entwicklungsumgebungen, welche eine AUTOSAR konforme Softwareentwicklung ermöglichen. Allerdings bieten nur wenige eine grafische Benutzeroberfläche zur Konfiguration der BSW-Module oder Validierung der erstellten Konfiguration an.

Konfigurationsaufwand	Nach erfolgreich überstandener Einarbeitungszeit lassen sich die Vorteile dieser, etwas komplexeren, Entwicklungsumgebung voll ausnutzen. Sie zeigen sich vor allem im stark minimierten Aufwand der Konfiguration. Das Setzen simpler Auswahlfelder, Importieren von Kommunikationskonfigurationen und Selektieren von gewünschten Objekten aus Listen, erspart Unmengen an geschriebenem Programmcode. Dieser wird jedoch im Zuge der Systemgenerierung erzeugt und lässt sich nachvollziehen.
Konfigurationsunterstützung	Im Zuge dieser Arbeit konnte das volle Potential des "Elektrobit Tresos Studio" nicht ausgenutzt werden. Dieses lässt sich erst ausschöpfen, wenn für die zu erstellende ECU die vollständige ARXML-Konfiguration aus den vorherigen Systemdesigntools verfügbar ist. Trotzdem konnte die Toolchain vollkommen überzeugen. Es konnte auf die Demonstrationsprojekte, eine ausführliche Dokumentation sowie die bereits ausführlich beschriebenen Features des Werkzeugs zurück gegriffen werden.
Komplexität	Ein Werkzeug mit dem Funktionsumfang von "Elektrobit Tresos" ist von Natur aus eher als komplex anzusehen. Nach erfolgreicher Einarbeitungszeit nimmt diese allerdings ab und beschränkt sich sehr schnell auf die Komplexität der Fahrzeugsoftwareentwicklung selbst. Unnötig komplex gestaltet sich lediglich die Implementierung von dem C-Programmcode innerhalb der Entwicklungsumgebung, da keine Syntax Hervorhebung implementiert ist.
Erweiterbarkeit	Durch den modularen Systemaufbau und dem zugrunde liegenden AUTOSAR-Standard, lässt sich die Konfiguration eines Projekts, ohne größere Umstände, beliebig erweitern.
Austauschbarkeit	Der Entwicklungsumgebung liegt der AUTOSAR-Standard zugrunde. Dieser ist grundsätzlich ein Garant für Austauschbarkeit und Wiederverwertbarkeit. Der AUTOSAR-Standard stellt nicht nur die Austauschbarkeit von Systemmodulen innerhalb eines Projekts oder eines Unternehmens, sondern generell für alle beteiligten Marktteilnehmer in den Mittelpunkt.
Kompatibilität	Die verwendete Entwicklungsumgebung kann in Verbindung mit anderen Systemdesignwerkzeugen verwendet werden und generell AUTOSAR kompatible Datenformate wie ARXML importieren.

ASW Voraussetzungen	Die AUTOSAR konforme Umsetzung von ASW setzt eine strikte Trennung von ASW und BSW voraus. Die Grundvoraussetzung dafür ist, dass die ASW nur über die Schnittstellen der RTE auf die BSW zugreift.
Skalierbarkeit	Ein Hauptargument für den AUTOSAR-Standard ist die enorme Skalierbarkeit im Entwicklungsprozess. Da die verwendete Entwicklungsumgebung auf AUTOSAR basiert, ist dies selbsterklärend.
Wiederverwendbarkeit	Durch die klar definierten Interfaces der verschiedenen Softwaremodule, lassen sich sowohl ASW wie auch die einzelnen Module und Konfigurationen der BSW in unterschiedlichen Einsatzgebieten wiederverwenden.

Tabelle 7.9: "Elektrobit Tresos"-Konfiguration

Einer der Hauptvorteile der verwendeten "Elektrobit Tresos"-Lösung ist die übersichtliche Konfiguration der BSW in der Entwicklungsumgebung "Elektrobit Tresos Studio". Es werden verschiedene Konfigurationsunterstützungen, beginnend bei voreingestellten Standardkonfigurationen über Vorschläge zu alternativen Konfigurationen bis hin zur Validierung der konfigurierten Systeme, bereitgestellt. Besonders hervorzuheben ist die großteils grafisch unterstützte Systemkonfiguration. So ist es selten notwendig, Konfigurationen der BSW über C-Programmcode durchzuführen. Die Entwicklungsumgebung unterstützt in jedem Bereich des Systemerstellungsprozesses. Zu Beginn der Hardwarekonfiguration, BSW-Konfiguration, beim Aufsetzen der RTE-Schicht und gegen Ende bei der Integration der ASW.

7.8 Integration von externen Tools

Die verwendete Toolchain besteht nicht nur aus den "Elektrobit Tresos"-Tools, sondern beinhaltet auch noch die nachfolgend aufgelisteten Werkzeuge.

Tasking Compiler	Wurde mit Unterstützung von entsprechend konfigurierten "makefiles" verwendet, um aus dem von "Elektrobit Tresos Studio" generierten Programmcode, die auf dem Mikrocontroller ausführbare Programmdatei zu erstellen. Er verursachte im Entwicklungsprozess keine Probleme und erfüllte alle Anforderungen hinsichtlich Geschwindigkeit und Kompatibilität.

Trace32	Dieses Tool wurde zusammen mit der dazugehörigen "Lauterbach"-Hardware verwendet, um die erstellte und ausführbare Programmdatei auf den Mikrocontroller zu flashen. Zusätzlich wurde es noch für das Debugging eingesetzt. Diese Software bietet eine hervorragende grafische Benutzerschnittstelle und erlaubt die genaue Überwachung aller Register, sowie das Schritt-für-Schritt Debugging im Quellcode.
CANalyzer	Auch dieses Werkzeug besteht aus einer Hardware und einer Softwarekomponente. Es dient zur Überwachung und Simulation der CAN-Buskommunikation. Der "CANalyzer" verfügt dazu über alle nötigen Funktionen, inklusive der Möglichkeit die Kommunikation aufzuzeichnen.
ARMXL Generator	Dieses Tool wurde aufgrund fehlender Konfigurationsdateien als Unterstützung zur Erstellung von ARXML-Dateien, welche die CAN-Kommunikation beschreiben, erstellt. Das Softwaredesign sieht eine einfache Erweiterbarkeit hinsichtlich der ausgegebenen Datei vor.

Tabelle 7.10: Zusätzliche Tools

7.9 Zusammenfassung der Bewertung

In diesem Kapitel wurden die Eigenschaften der verwendeten Entwicklungswerkzeuge mit den Erkenntnissen aus der durchgeführten Arbeit kombiniert. Diese Bewertung soll eine Grundlage zur Entwicklungswerkzeugauswahl für zukünftige Projekte bilden.
Am Beginn eines Softwareprojekts für die Fahrzeugindustrie, ist es notwendig sich bestimmte Vorkenntnisse anzueignen. Programmierfähigkeiten sind zwar entscheidend, jedoch allein nicht ausreichend. Wesentlich sind vor allem Kenntnisse über die Automotive Industrie, den gängigen Standards wie AUTOSAR und ein vorhandenes Grundwissen bezüglich der Buskommunikation. Sind diese Voraussetzungen erfüllt, müssen die Rahmenbedingungen des Projekts evaluiert werden und in diesem Zuge Vorbedingungen und Ziele für die Software definiert werden. Auf Basis dieser Definitionen lässt sich eine geeignete Entwicklungsumgebung und Toolchain auswählen. Neben dem Einsatzgebiet sind allerdings auch noch Anschaffungskosten und Supportkosten ein wichtiger Faktor für die Auswahl der richtigen Werkzeuge. Das gewählte Werkzeug soll, je nach Projekt in dem es eingesetzt wird, verschiedene Standards und Zertifizierungen unterstützen. Da es sich bei dem durchgeführten Projekt um eine Evaluierung bezüglich der Serienproduktion handelt, waren im Rahmen dieses Projekts vor allem der AUTOSAR-Standard und die

Zertifizierbarkeit nach verschiedenen ISO-Normen und dem ASIL-Standard, entscheidend. Die Einsatzfähigkeit in den vielen unterschiedlichen Anwendungsgebieten der Fahrzeugindustrie lässt sich nach verschiedenen Kriterien beurteilen. Eine hohe Einarbeitungszeit, gefolgt von großem Produktivitätspotential sind zum Beispiel eine gute Basis für ein Projekt in der Serienproduktion. Um ein qualitativ hochwertiges Programm entwickeln zu können, werden effektive und benutzerfreundliche Entwicklungswerkzeuge benötigt. Dazu zählen eine Entwicklungsumgebung welche im gesamten Entwicklungsprozess unterstützen kann und effizient zu verwenden ist, entsprechender Support vom Hersteller der Entwicklungswerkzeuge quer durch den ganzen Projektablauf und außerdem eine hochqualitative, aktuelle und vollständige Dokumentation der eingesetzten Software. Wenn die Entwicklungsumgebung passend gewählt wurde, hält sich der Konfigurationsaufwand der BSW, den Umständen entsprechend, in Grenzen. Zusätzlich wird dadurch ein hohes Maß an Kompatibilität, Wiederverwendbarkeit und Austauschbarkeit erzielt. Im Hinblick auf diese Arbeit bildet die Entwicklungsumgebung eine sehr gut abgestimmte Einheit mit der restlichen Toolchain, welche aus dem Tasking Compiler, dem Trace32 Tool und dem "CANalyzer" besteht.

Kapitel 8

Zusammenfassung und Ausblick

Dieses letzte Kapitel gibt einen Überblick über die behandelten Themengebiete und fasst die Ergebnisse dieser Arbeit zusammen. Zusätzlich gibt es sowohl einen Ausblick auf geplante sowie auch auf zukünftige notwendige Entwicklungsschritte.

8.1 Zusammenfassung

Das Ziel dieser Arbeit war ein intensives Studium, der für die Softwareentwicklung in der Fahrzeugindustrie notwendigen Wissensgebiete. Nach der erfolgreichen Einarbeitung in die Domäne der Fahrzeugindustrie wurde die Konzeptphase eingeleitet. Zuerst wurde ein intensives Studium der nötigen BSW-Module durchgeführt. Daraus entwickelte sich im Anschluss ein Konzept für das grundlegende Projekttemplate. Für dieses Template wurde ein passender Demonstrator entworfen, welcher in der Implementierungsphase in das entstandene Projekttemplate integriert wurde. Dieser verwendet die CAN-Buskommunikation und evaluiert die grundlegenden Funktionen des Projekttemplates. Zur Unterstützung der Integration wurde, wie bereits erwähnt ein ARXML-Generator entwickelt. Dieser sollte die nötigen Konfigurationsdateien für die CAN-Kommunikationssignale erstellen.
Im Kapitel der Softwareimplementierung wurde die Implementierung selbst beschrieben. Zuerst wurde auf die verwendete Entwicklungsumgebung und die Toolchain eingegangen. Anschließend wurden die Anforderungen an das Projekttemplate definiert. Im Mittelpunkt dieses Kapitels stand die genaue technische Umsetzung des Projekttemplates, des Demonstrators sowie des ARXML-Generators. Um die Funktionsfähigkeit des Demonstrators zu evaluieren, wurden entsprechende Testfälle erstellt und angewendet.
Aufbauend auf diese Erfahrungen wurde die Integration eines ersten praktischen Kundenprojekts durchgeführt. Auch in diesem Kontext konnten sehr gute Ergebnisse erzielt werden. Diese zweite Migration ist allerdings nicht mehr Thema dieser Arbeit.
Auf Basis der bis dahin gesammelten Erfahrungen wurde eine ausführliche Bewertung der Entwicklungsumgebung, Toolchain und des gesamten Projekts durchgeführt. Dabei wurde noch einmal gezielt auf viele unterschiedliche Kriterien bezüglich der genutzten Werkzeuge in Kombination mit dem erstellten Projekt eingegangen. Dieses Kapitel soll bei der Auswahl von Entwicklungsumgebungen in weiteren Projekten unterstützen.

Um die Wichtigkeit dieser Domäne noch einmal in den Vordergrund zu rücken, ist es bedeutend zu wissen, dass seit der ersten Erscheinung von Mikrocontrollern in der Fahrzeugindustrie die potenziellen Einsatzgebiete enorm gesteigert wurden. Von unterstützenden Hilfssystemen, über komplette Fahrzeugüberwachungssysteme und der vollkommen elektronischen Regeln der Fahrzeugkomponenten, bis hin zum autonomen Fahren wird den eingesetzten Mikrocontrollern alles abverlangt. Aus diesem Grund ist es nicht verwunderlich, dass die Fahrzeugindustrie in kurzer Zeit sämtliche Evolutionssprünge der klassischen Desktop Computer vollzogen hat. Zu Beginn haben Mehrkernsysteme, aufgrund der komplexen Ressourcenverwaltung und Echtzeit kritischen Anwendungen, ein erhebliches Problem dargestellt. Diese Entwicklung ist mittlerweile jedoch umgesetzt.
Durch immer komplexer werdende, vernetzte Systeme kam, wie in Kapitel 3 ausführlich beschrieben wurde, der Wunsch nach Standardisierung auf. OSEK gehört zu den ersten wichtigen Standards der Fahrzeugindustrie. Das Ziel war, einen offenen Industriestandard für verteilte Steuergeräte zu entwickeln. Schon damals erkannte man das Potential einer modularen Struktur. Der Fokus sollte auf eindeutig definierten Kommunikationsstrukturen und Schnittstellen liegen. Im Jahr 2003 wurde das AUTOSAR-Konsortium gegründet, welches einen der wichtigsten Standards der Gegenwart darstellt. Das Motto von AUTOSAR: "Cooperate on standards, compete on implementation" beschreibt schon sehr genau was mit dessen Einführung erreicht werden sollte. Die BSW sollte weiter von der ASW entkoppelt werden, was vor allem durch die Einführung der RTE ermöglicht wird. Der AUTOSAR-Standard ermöglicht die Wiederverwertbarkeit, vereinfachte Integration und Austauschbarkeit von Softwarekomponenten innerhalb eines Projekts sowie auch unter den verschiedenen Marktteilnehmern. Mit zunehmender Wichtigkeit der angesteuerten Systeme ist auch der Wunsch nach Standardisierung in Sicherheitsfragen gestiegen. Der ISO 26262 Standard sowie die ASIL Zertifizierungen stellen die wesentlichen Konzepte bezüglich dieses Bereichs der Automotive Industrie dar.
Die theoretische sowie auch praxisorientierte Auseinandersetzung mit einer AUTOSAR konformen Multicore-Serienproduktions-Plattform soll eine fundierte Grundlage für zukünftige Projekte bieten.

8.2 Ausblick

Im Zuge dieser Arbeit wurde ein AUTOSAR konformes Projekttemplate für die Migration von nicht AUTOSAR konformen Projekten erstellt und durch die Integration eines Demonstrators evaluiert.
AUTOSAR ist der gängige und global akzeptierte Entwicklungsstandard in der Fahrzeugindustrie. Dies wird durch den enormen Zustrom an führenden Unternehmen der Automotive Industrie eindeutig belegt. Die fortlaufende Standardisierung ist die Grundlage für Wiederverwertbarkeit, Austauschbarkeit, Wartbarkeit und Flexibilität in der gesamten Branche. Daher wird AUTOSAR maßgeblich dazu beitragen, dass sich die Entwicklung in Zukunft immer stärker auf Endverbraucherbedürfnisse fokussieren kann.
In der heutigen Zeit, wo autonome Fahrzeuge bereits um ihre Zulassung für den Straßen-

verkehr kämpfen, werden längst alle wesentlichen Fahrzeugsysteme elektronisch gesteuert. Darauf begründet sich die Wichtigkeit der Sicherheitsstandardisierungen rund um die ISO 26262 sowie der ASIL-Zertifizierung, welche auch in Zukunft von immer höherer Bedeutung werden.
Im Hinblick auf das entwickelte Projekttemplate und die, bereits beschriebene, Integration des Kundenprojekts liegt trotzdem noch einiges an Arbeit vor dem Projektteam. Um ein stabiles System zu gewährleisten, müssen die Abhängigkeiten der ASW-Tasks genau analysiert werden und anschließend eine effiziente Verteilung auf die vorhandenen Prozessorkerne durchgeführt werden. Des Weiteren werden Konfigurationen hinsichtlich der Kommunikationsnetzwerke erstellt und die Hardwarekonfigurationen angepasst. Um einen reibungslosen Programmablauf zu gewährleisten werden abschließend noch Funktionstests und Sicherheitstests durchgeführt.
Um in der Fahrzeugsoftwareentwicklung konkurrenzfähig bleiben zu können, wird man in Zukunft immer stärker auf Standardisierung und Zertifizierung eingehen müssen. Der in diesem Projekt erarbeitete Ansatz stellt einen realistischen Entwicklungsprozess der nahen Zukunft dar.

Verzeichnisse

Abkürzungsverzeichnis

AUTOSAR	AUTomotive Open System ARchitecture
ABS	Antiblockiersystem
ARXML	AUTOSAR XML
ADC	Analog-Digital Konverter
API	Application Programming Interface
ASIL	Automotive Safety Integration Level
ASW	Anwendungssoftware
ASAM	Association for Standardisation of Automation and Measuring Systems
BSW	Basis-Software
BswM	Basis-Software Mode Manager
CCs	Communication Controllers
CAN	Controller Area Network
CanIf	Controller Area Network Interface
CanSM	Controller Area Network State Manager
COM	Communication
CPU	Central Processing Unit
CPUs	Central Processing Units
DBC	CAN DataBase Container
DIO	Digital Input Output
DET	Development Error Tracer
DEM	Diagnostic Event Manager
E/E	Elektrisch/Elektronisch
ECU	Electric Control Unit
EcuM	ECU State Manager
ECUs	Electric Control Units
FIFO	First In – First Out
GTM	Generic Timer Module
GUI	Grafische Benutzeroberfläche

GPT General Purpose Timer
HIS Herstellerinitiative Software
ID Identifikator
I/O Input/Output
IDE Integrated Development Environment
IOC Inter-OS-Application Communication
ISO Internationale Organisation für Normung
ISR Interrupt Service Routine
IRQ Interrupt Request Module
IRV Inter Runnable Variable
I-PDU Interaction Layer Protocol Data Unit
I-PDUs Interaction Layer Protocol Data Units
LOKI List of Known Issues
LIN Local Interconnected Network
MCAL MicroController Abstraction Layer
MCU Micro Controller Unit
MCUs Micro Controller Units
MC Mehrkern
MISRA Motor Industry Software Reliability Association
OBD On-Board-Diagnostic
OMG Meta Object Facility
OS Operating System
OSEK Offene Systeme für die Elektronik im Kraftfahrzeug
OSEK/VDX Offene Systeme für die Elektronik im Kraftfahrzeug / Vehicle Distributed eXecutive
OSEK COM OSEK Kommunikation
OSEK NM OSEK Netzwerk Monitoring
OIL OSEK Implementation Language
PDU Protocol Data Unit
PDUs Protocol Data Units
PduR PDU Router
PCP Priority Ceiling Protocol
PWM Pulsweitenmodulation
RTE Runtime Environment
RTOS Real Time Operating System

SW-C	Software Component
TOM	Timer Output Module
VDX	Vehicle Distributed Executive
VFB	Virtual Functional Bus
WCET	Worst Case Execution Time
XML	Extensible Markup Language

Abbildungsverzeichnis

Quelltextverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] Luis Almeida, Jakob Axelsson, Manuel Barranco, Patrice Bodu, Mirko Conrad, Joaquim Ferreira, Ines Fey, Ulrich Freund, Thomas M. Galla, Michael Golm, Michael Gonschorek, Mathieu Grenier, Hans A. Hansson, Bernd Hardung, Lionel Havet, Thorsten Kölzow, Andreas Krüger, Christian Kühnel, and Henrik Lönn. *Automotive embedded systems handbook*. Industrial information technology series. CRC Press, Boca Raton, 2009.

[2] Altium. *TASKING VX-toolset for TriCore User Guide*, v4.3 edition, 07 2013.

[3] AUTOSAR Development Cooperation. AUTOSAR AUTomotive Open System ARchitecture, 2009.

[4] AUTOSAR Development Cooperation. Guide to Multi-Core Systems. online, 2013.

[5] AUTOSAR Development Cooperation. Specification of ADC Driver, 2015.

[6] AUTOSAR Development Cooperation. Specification of CAN Driver, 2015.

[7] AUTOSAR Development Cooperation. Specification of Communication. http://www.engineersgarage.com/articles/autosar-automotive-open-systems-architecture, 10 2015.

[8] AUTOSAR Development Cooperation. Specification of DIO Driver, 2015.

[9] AUTOSAR Development Cooperation. Specification of GPT Driver, 2015.

[10] AUTOSAR Development Cooperation. Specification of MCU Driver, 2015.

[11] AUTOSAR Development Cooperation. Specification of Operating System, 2015.

[12] AUTOSAR Development Cooperation. Specification of PDU Router, 2015.

[13] AUTOSAR Development Cooperation. Specification of Port Driver, 2015.

[14] AUTOSAR Development Cooperation. Specification of PWM Driver, 2015.

[15] Jakob Axelsson. AutoSAR Overview. FESA Workshop at KTH, April 2010.

[16] Alessandra Mitidieri C. Autosar automotive open system architecture, cooperate on standards, compete on implementation. http://www.automotive-spin.it/uploads/4/mitidieri-4W.pdf, 2008.

[17] K. Chaaban, P. Leserf, and S. Saudrais. Steer-By-Wire system development using AUTOSAR methodology. In *Emerging Technologies Factory Automation, 2009. ETFA 2009. IEEE Conference on*, pages 1–8, Sept 2009.

[18] Manjusree S. PG Scholar Embedded Systems Specialist Automotive Electronics Assistant Professor. Department of Electronics Sree Buddha College of Engineering Kerala India Transportation Business Unit Tata Elxsi Ltd Kerala India Dhanamjayan P.R.,

Kuruvilla Jose. Ecu state manager module development and design for automotive platform software based on autosar 4.0. In *International Journal of Technical Research and Applications e-ISSN: 2320-8163*, pages 230–235, Jul 2015.

[19] D. Diekhoff. Autosar basic software for complex control units. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, pages 263–266, March 2010.

[20] Elektrobit. Elektrobit. https://www.elektrobit.com/, September 2015.

[21] Elektrobit. Elektrobit tresos. https://www.elektrobit.com/products/ecu/eb-tresos/, September 2015.

[22] Elektrobit. Elektrobit tresos autocore. https://www.elektrobit.com/products/ecu/eb-tresos/autocore/, September 2015.

[23] Elektrobit. Elektrobit tresos studio. https://www.elektrobit.com/products/ecu/eb-tresos/studio/, September 2015.

[24] Fabrizio Fabbrini, Mario Fusani, Giuseppe Lami, and E. Sivera. Software engineering in the european automotive industry: Achievements and challenges. In *Computer Software and Applications, 2008. COMPSAC '08. 32nd Annual IEEE International*, pages 1039–1044, July 2008.

[25] M. Farsi, K. Ratcliff, and M. Barbosa. An overview of controller area network. *Computing Control Engineering Journal*, 10(3):113–120, June 1999.

[26] Helmut Fennel, Stefan Bunzel, Harald Heinecke, Juergen Bielefeld, Simon Fuerst, Bmw Group, Klaus peter Schnelle, Walter Grote, Nico Maldener, Thomas Weber, Florian Wohlgemuth, Jens Ruh, Lennart Lundh, Tomas S, Ford Motor Company, Peter Heitkaemper, Robert Rimkus, General Motors, Jean Leflour, Alain Gilberg, Psa Peugeot Citroen, Ulrich Virnich, Stefan Voget, Siemens Vdo, Klaus Lange, Thomas Scharnhorst, and Bernd Kunkel. Achievements and exploitation of the autosar development partnership. technical report, society of automotive engineers, 2006.

[27] Christoph Ficek, Nico Feiertag, and Kai Richter. Applying the AUTOSAR timing protection to build safe and efficient ISO 26262 mixed-criticality systems, 2012.

[28] S. Furst. Challenges in the design of automotive software. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, pages 256–258, March 2010.

[29] Simon Fürst, Jürgen Mössinger, Stefan Bunzel, Thomas Weber, Frank Kirschke-Biller, Peter Heitkämper, Gerulf Kinkelin, Kenji Nishikawa, and Klaus Lange. Autosar–a worldwide standard is on the road. In *14th International VDI Congress Electronic Systems for Vehicles, Baden-Baden*, volume 62, 2009.

[30] Thomas Galla. Beyond AUTOSAR - Optimized AUTOSAR Compliant Basic Software Modules. http://www.researchgate.net/publication/228408165_Beyond_AUTOSAROptimized_AUTOSAR_Compliant_Basic_Software_Modules, 1 2008.

[31] Alain Gilberg, Steffen Lupp, Simon Fuerst, Demetrio Aiello, Stefan Schmerler, Frank Kirschke-Biller, Robert Rimkus, Kenji Nishikawa, and Andreas Titze. AUTOSAR: Achievements, roll-out, perspectives. In *ERTS 2012*, 2012.

[32] P.-E. Hladik, A. Deplanche, S. Faucou, and Y. Trinquet. Adequacy between autosar os specification and real-time scheduling theory. In *Industrial Embedded Systems, 2007. SIES '07. International Symposium on*, pages 225–233, July 2007.

[33] U. Honekamp. The autosar xml schema and its relevance for autosar tools. *Software, IEEE*, 26(4):73–76, July 2009.

[34] Infineon Technologies AG. *Application Kit TC2X5 Users Manual*. Infineon Technologies AG, 81726 Munich, Germany, 2013-05 edition, 05 2013.

[35] Infineon Technologies AG. *Infineon AUTOSAR Software*. Infineon Technologies AG, 81726 Munich, Germany, 2015-09 edition, 09 2015.

[36] ISO. ISO 26262-1:2011, 2011.

[37] S.K. Jena and M.B. Srinivas. On the suitability of multi-core processing for embedded automotive systems. In *Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2012 International Conference on*, pages 315–322, Oct 2012.

[38] Daniel Kaestner, Marc Schlickling, Markus Pister, Christoph Cullmann, Gernot Gebhard, Reinhold Heckmann, and Christian Ferdinand. Meeting real-time requirements with multi-core processors. In Frank Ortmeier and Peter Daniel, editors, *Computer Safety, Reliability, and Security*, volume 7613 of *Lecture Notes in Computer Science*, pages 117–131. Springer Berlin Heidelberg, 2012.

[39] F. Kluge, M. Gerdes, and T. Ungerer. Autosar os on a message-passing multicore processor. In *Industrial Embedded Systems (SIES), 2012 7th IEEE International Symposium on*, pages 287–290, June 2012.

[40] Balaji Kulkarni. Autosar automotive open systems architecture. http://www.engineersgarage.com/articles/autosar-automotive-open-systems-architecture, 10 2015.

[41] Daehyun Kum, Gwang-Min Park, Seonghun Lee, and Wooyoung Jung. Autosar migration from existing automotive software. In *Control, Automation and Systems, 2008. ICCAS 2008. International Conference on*, pages 558–562, Oct 2008.

[42] Giuseppe Lami, Fabrizio Fabbrini, and Mario Fusani. Is automotive spice suitable to assess product lines-based software process? In *Engineering of Computer Based Systems (ECBS-EERC), 2011 2nd Eastern European Regional Conference on the*, pages 157–158, Sept 2011.

[43] Lauterbach. Lauterbach. http://www.lauterbach.com/, September 2015.

[44] Patrick Leteinturier, Simon Brewerton, and Klaus Scheibert. MultiCore Benefits & Challenges for Automotive Applications. In *SAE Technical Paper*. SAE International, 04 2008.

[45] Fang Li, Lifang Wang, and Chenglin Liao. Can(controller area network) bus communication system based on matlab/simulink. In *Wireless Communications, Networking and Mobile Computing, 2008. WiCOM '08. 4th International Conference on*, pages 1–4, Oct 2008.

[46] Georg Macher, Muesluem Atas, Eric Armengaud, and Christian Kreiner. Automotive Real-time Operating Systems: A Model-Based Configuration Approach. In Jalil Boukhobza, Jean Philippe Diguet, Pierre Ficheux, Jose Rufino, and Frank Singhoff, editors, *ACM SIGBED Review Special Interest Group on Embedded Systems*, volume 1291 of *CEUR Workshop Proceedings*. Association for Computing Machinery. Special Interest Group on Embedded , 2014.

[47] Alexander Much. Removing Run-Time Errors from AUTOSAR Components Using Polyspace Code Verifiers. http://www.mathworks.com/tagteam/69796_91933v00_runtime_errors_autosar.pdf, 2011.

[48] Nico Naumann. AUTOSAR Runtime Environment and Virtual Function Bus. http://hpi.de/fileadmin/user_upload/fachgebiete/giese/Ausarbeitungen_ AUTOSAR0809/NicoNaumann_RTE_VFB.pdf, 2009.

[49] N. Navet, A. Monot, B. Bavoux, and F. Simonot-Lion. Multi-source and multicore automotive ecus - os protection mechanisms and scheduling. In *Industrial Electronics (ISIE), 2010 IEEE International Symposium on*, pages 3734–3741, July 2010.

[50] Farhang Nemati, Moris Behnam, and Thomas Nolte. Efficiently Migrating Real-Time Systems to Multi-Cores, 2009.

[51] M. Niklas, S. Voget, and J. Mottok. Safety-relevant development by adaptation of standardized safety concepts in AUTOSAR 4.0, 2012.

[52] OSEK/VDX Steering Committee. OSEK/VDX Binding Specification . http://portal.osek-vdx.org/files/pdf/specs/binding142.pdf, 2004.

[53] OSEK/VDX Steering Committee. OSEK/VDX Communication. http://portal.osek-vdx.org/files/pdf/specs/osekcom303.pdf, 2004.

[54] OSEK/VDX Steering Committee. OSEK/VDX Network Management: Concept and Application Programming Interface . http://portal.osek-vdx.org/files/pdf/specs/nm253.pdf, 2004.

[55] OSEK/VDX Steering Committee. OSEK/VDX System Generation OIL: OSEK Implementation Language. http://portal.osek-vdx.org/files/pdf/specs/oil25.pdf, 2004.

[56] Mike Pagel and Mark Broerkens. Definition and Generation of Data Exchange Formats in AUTOSAR, process independent model. *lNCS 4066*, pages pp. 52–65, 2006.

[57] Helmut Schelling. AUTOSAR - Equipped for Everything. *VECTOR Technical Article*, pages 1–4, March 2014.

[58] Stefan Schmerler and Robert Rimkus. Autosar weichenstellung in die zukunft. *ATZ-elektronik*, 8(1):58–63, 2013.

[59] OSEK/VDX steering committee. Osek/vdx operating system specification 2.2.3. February 2005.

[60] Vector. Canalyzer. http://vector.com/vi_canalyzer_de.html, September 2015.

[61] Stefan Voget. Autosar and the automotive tool chain. In *Proceedings of the Conference*

on Design, Automation and Test in Europe, DATE '10, pages 259–262, 3001 Leuven, Belgium, Belgium, 2010. European Design and Automation Association.

[62] Qiang Wang, Baiyu Xin, Chao Li, and Hong Chen. The realization of reusability in vehicular software development under autosar. In *Conference Anthology, IEEE*, pages 1–6, Jan 2013.

[63] Yuefei Wang, Xiang Wu, Liang Hou, and Zhiqiang Tang. Reliability analysis and improvement of osek automotive network management. In *Computer Science and Network Technology (ICCSNT), 2013 3rd International Conference on*, pages 742–746, Oct 2013.

[64] D.D. Ward. Misra standards for automotive software. In *Automotive Electronics, 2006. The 2nd IEE Conference on*, pages 5–18, March 2006.

[65] A. Zahir. Oil-osek implementation language. In *OSEK/VDX Open Systems in Automotive Networks (Ref. No. 1998/523), IEE Seminar*, pages 8/1–8/3, Nov 1998.

[66] Werner Zimmermann and Ralf Schmidgall. AUTOSAR-Softwarearchitektur für Kfz-Systeme. In *Bussysteme in der Fahrzeugtechnik*, ATZ/MTZ-Fachbuch, pages 367–413. Springer Fachmedien Wiesbaden, 2014.

[67] Werner Zimmermann and Ralf Schmidgall. *Bussysteme in der Fahrzeugtechnik*. Springer Fachmedien Wiesbaden, Wiesbaden, 2014.