# Von Funktionsblöcken bis AUTOSAR

## Komponentenmodelle im Entwicklungsprozess von Automobiltechnik

Steve Colling
228 463

Betreut von Dipl.-Inform. Christian Fuß

### Zusammenfassung

Steigende Anforderungen an Hard- und Software führen zu immer komplexeren Architekturen. Dies gilt besonders im Bereich der Softwareanwendungen im Automobilbereich. Um diese immer größer werdenden Architekturen überhaupt noch überschauen zu können, vor allem bei stark verteilten Systemen wie sie etwa im Automobil vorliegen, sind Entwickler, Entwerfer, Implementierer als auch Qualitätssicherer seit Jahren dazu angehalten solche großen Architekturen in übersichtliche Teilsysteme (Komponenten) zu unterteilen. Ein solches Vorgehen bei der Programmerstellung führte zum Paradigma der Komponenten basierten Softwareentwicklung und in der Automobilindustrie zur Einführung des AUTOSAR-Komponentenmodells. Diese Arbeit soll den Wert und die Wichtigkeit solcher Modelle, vor allem des AUTOSAR-Komponentenmodells veranschaulichen und die wichtigsten Eigenschaften der Komponentenmodelle Funktionsblöcke und AUTOSAR vorstellen.

# Inhaltsverzeichnis

# 1 Einleitung

Technischer Fortschritt führt zu immer komplizierter realisierbaren Produkten. Arbeitsteilung und Spezialisierung haben sich ab dem 19. Jahrhundert, in der industriellen Güterherstellung, als unerlässlich zur effizienten Planung und Produktion erwiesen. Dies führte zu komponentenbasierten Entwicklungsmodellen. So stellen in der Automobilindustrie Zulieferer Automobilkomponenten her (z.B. Auspuff, Getriebe, Stoßdämpfer), welche von den eigentlichen Herstellern zusammengebaut werden.

Mittlerweile stellen Softwareanwendungen die Basis eines hohen Anteils der Wertschöpfungskette der Automobilindustrie dar. Ein um 1996 gebautes Automobil verfügt über etwa ein halbes Dutzend verteilter Kontrolleinheiten und damit bereits über mehr Rechenleistung und Speicherplatz als die 1969 auf dem Mond gelandete Raumkapsel. Aktuelle Automobile verfügen über dutzende von Kontrolleinheiten, tausende von Funktionen und zehntausende von potentiellen Vernetzungsmöglichkeiten. Zur Realisation von anforderungserfüllenden Architekturen bedarf es eines von dieser Komplexität abstrahierenden Modells. Ein solches stellt AUTOSAR dar. AUTOSAR ist ein komponentenbasiertes Entwicklungsmodell für Anwendungen im KFZ. Es soll Anwendungsarchitekturen bei Entwicklung, Wartung, Skalierung, Erweiterung oder sonstiger Modifikation durch Abstraktion von Kommunikationsdetails übersichtlich halten und eine parallele Entwicklung durch klar definierte Arbeitsteilung (Erstellung einzelner Komponenten) ermöglichen.

In dieser Arbeit wird zunächst der Begriff der Komponente und des Komponentenmodells in der Informatik erläutert werden. Des Weiteren soll die Erstellung von Software mithilfe des Modells „Funktionsblöcke“ vorgestellt werden, welches ein frühes Komponentenmodell für eingebettete Systeme darstellt. Anschließend werden die wichtigsten Eigenschaften des AUTOSAR-Komponentenmodells, wie mehrschichtige Hardwareabstraktion oder maßgeschneiderte Laufzeitumgebung präsentiert. Schließlich werden die einzelnen Schritte des Erstellungsprozesses eines Systems aus Komponenten mit Hilfe des Modells, sowie die Realisierung einer einzelnen Komponente im Modell erklärt.

# 2 Komponenten

Eine Komponente ist definiert als eine in sich vollständige, wiederverwendbare Softwareeinheit mit eindeutig definierter Schnittstelle zur Integration in einer Umgebung, meist bestehend aus anderen Komponenten [12]. Das Innenleben ei-

ner Komponente ist daher für seinen Verwender irrelevant. In der Literatur existieren verschiedene Komponententypen, die wichtigsten hiervon sind zum einen Softwarekomponenten, welche meist Anwendungs-, Betriebssystem- oder Hardwareabstraktionskomponenten sind, und zum anderen Hardwarekomponenten wie etwa Kontrolleinheiten, Sensoren oder Aktoren. Middleware ist meistens Software welche eine Laufzeitumgebung als Abstraktion der zugrunde liegenden Hardware realisiert. Daher werden Betriebssystem- oder Hardwareabstraktionskomponenten oft als Middlewarekomponenten bezeichnet [5]. Diese stellen keine Anwendungsfunktionalitäten sondern Basisdienste für Anwendungskomponenten bereit. Zur Portierung einer komplexen Architektur auf eine neue Basismaschine müssen daher nur Middlewarekomponenten ausgetauscht werden.

Anwendungen werden meist durch Komposition mehrerer Anwendungskomponenten realisiert. Einzelne Komponenten realisieren Teilanwendungen und stellen somit ein Teilsystem des Anwendungssystems dar. Da für den Verwender einer Komponente nur deren Schnittstelle interessant ist, kann eine Anwendungskomponente durch eine andere ersetzt werden, wenn diese die gleichen Schnittstellen zur Verfügung stellt. Auf diese Weise können im Idealfall geänderte Anforderungen durch Ersetzung weniger Komponenten im System erfüllt werden.

In dieser Arbeit wird im Folgenden eine Komponente als atomar betrachtet, womit ausgedrückt sei dass sie nicht auf mehrere Kontrolleinheiten verteilt werden darf. In der Literatur werden oft jedoch auch größere Konstrukte mehrerer Komponenten als wiederum solche bezeichnet. Um jegliche Verwechslungen auszuschließen werden solche zusammengesetzten Komponenten hier jedoch immer als Kompositionen bezeichnet.

## 2.1 Komponentenmodelle

Um konsistente Anwendungen mit Hilfe von Komponenten realisieren zu können müssen klare Richtlinien für deren Kommunikation festgelegt werden. Diese Standards werden von den einzelnen Komponentenmodellen festgelegt. In allen Komponentenmodellen ist Kommunikation allein über Schnittstellen möglich. Diese Schnittstellen werden von den Modellen standardisiert. Auf diese Weise weiß ein Entwickler einer Komponente genau mit welcher Syntax Zugriffe auf die Komponente im Modell erfolgen und somit auch wie er benötigte Funktionalitäten an der Schnittstelle (im Modell) anbieten muss. Solche Modelle sollen die Entwicklung von Anwendungen vereinheitlichen und vereinfachen. Je nach Komponentenmodell und dessen Abstraktionsgrad, werden hoch entwickelte Softwarewerkzeuge benötigt um so erstellte komponentenbasierte Architekturmodelle auf Maschinencode abzubilden. Die Transformationsschritte, welche mit solcher

Werkzeugunterstützung realisiert werden, müssen somit auch vom Modell spezifiziert werden.

Eines der ersten weit verbreiteten Komponentenmodelle stellte CORBA (Common Object Request Broker Architecture) dar. Aus ihm gingen später die heute meist verwendeten Komponentenmodelle COM/DCOM von Microsoft und Enterprice Java Beans (EJB) aus dem Hause Sun Microsystems hervor. Während EJB von Anfang an zur Modellierung verteilter Architekturen fähig war, musste COM hierzu auf DCOM erweitert werden. In dieser Arbeit werden Funktionsblöcke und das an ihnen bei der Entwicklung orientierte AUTOSAR-Modell behandelt. Der prinzipielle Unterschied in AUTOSAR zu Funktionsblöcken nach IEC 61131-3 ist die Modellierbarkeit von einzelnen auf verschiedenen Kontrolleinheiten laufenden Funktionsblöcken, hier Komponenten genannt. Bei neueren Funktionsblockmodellen nach (IEC 61499) ist dies jedoch auch möglich. Die schichtenartige Hardwareabstraktion, das maßgeschneiderte Betriebssystem und das zusammenfassen jeglicher Vernetzung zu einem virtuellen Bus bleibt jedoch dem AUTOSAR-Modell vorbehalten. Ähnlich dem maßgeschneiderten OSEK-Betriebssystems in der AUTOSAR-Architektur kommen heute immer häufiger komponentenbasierte Betriebssysteme wie TinyOS [10] oder Contiki [1] zum Einsatz; dies in verteilten Anwendungen auf extrem ressourcenschwachen Kontrolleinheiten. Wenn letztere Betriebssysteme auch nicht Thema dieser Ausarbeitung sind, so zeigt ihr immer häufiger werdender Einsatz, dass das Vorurteil, modellbasierte Konstrukte wiederverwendbarer Softwareeinheiten seien ineffizient, nicht der Realität entspricht.

## 2.2 Interaktion von Komponenten

In modernen Komponentenmodellen wird eine strikte Trennung von Datenfluss und Kontrollfluss vorgenommen. Der Kontrollfluss zwischen Komponenten geschieht ereignisbasiert. Es werden Ereignisse von Komponenten an die Schnittstellen anderer geschickt, um deren Aktionen anzustoßen. Dies kann sowohl auf Client-Server als auch auf dem Sender/Receiver Prinzip basieren. Da bei Client/Server-Kommunikation ein Ereignis, welches eine Operation anstößt, oft auch gleich die Argumente enthält, und die darauf folgende Bestätigung der erfolgten Operation oft auch ein Resultat verkapselt muss auch von dieser Kommunikation abstrahiert werden. Im Modell erscheinen als Ereignisse, und somit als reiner Kontrollfluss, lediglich „Operation anstoßen“ und „Operation abgeschlossen“. Die Parameterübergabe und das Zurückliefern von Resultaten wäre hier als Datenfluss modelliert. Bei Sender/Receiver-Kommunikation wäre die Abstraktion

im Fall von parameterhaltigen send-Ereignissen analog. (Resultatrückgabe würde ganz wegfallen.)

# 3 Funktionsblöcke

Funktionsblöcke, nach IEC 61131 [8], gibt es seit den 90er Jahren. Sie haben sich seither als ein standardisiertes Komponentenmodell für Automatisierungsanwendungen durchgesetzt. Abbildung 1 zeigt die fünf von IEC 61131 festgelegten Spezifikationsdarstellungen. Eine erweiterte Form des IEC 61131 ist der IEC 61499 [9] welcher zur Modellierung verteilter und somit parallel laufender Systeme erweitert wurde. Laut IEC 61499 stellen Funktionsblöcke ein „generisches Modell für verteilte Systeme“ dar, in welchem Anwendungen als Netzwerk von funktionalen Modulen mit Zustandsspeicher, dargestellt werden.

Das Entwicklungsparadigma erlaubt sowohl symbolisches Programmieren, die Visualisierung topologischer Verteilung und Vernetzung als auch die allgemeine geräteübergreifende Konfiguration. Es existieren mittlerweile mehrere Software-Werkzeuge welche eine Erstellung IEC 61499 verträglicher Anwendungen ermöglichen. Hierbei handelt es sich etwa um ISaGRAF [1], L-Force Engineer [2] und IndraWorks [3] welche zur Realisierung von verteilten Automatisierungsanwendungen besonders geeignet sind.

## 3.1 Spezifikation

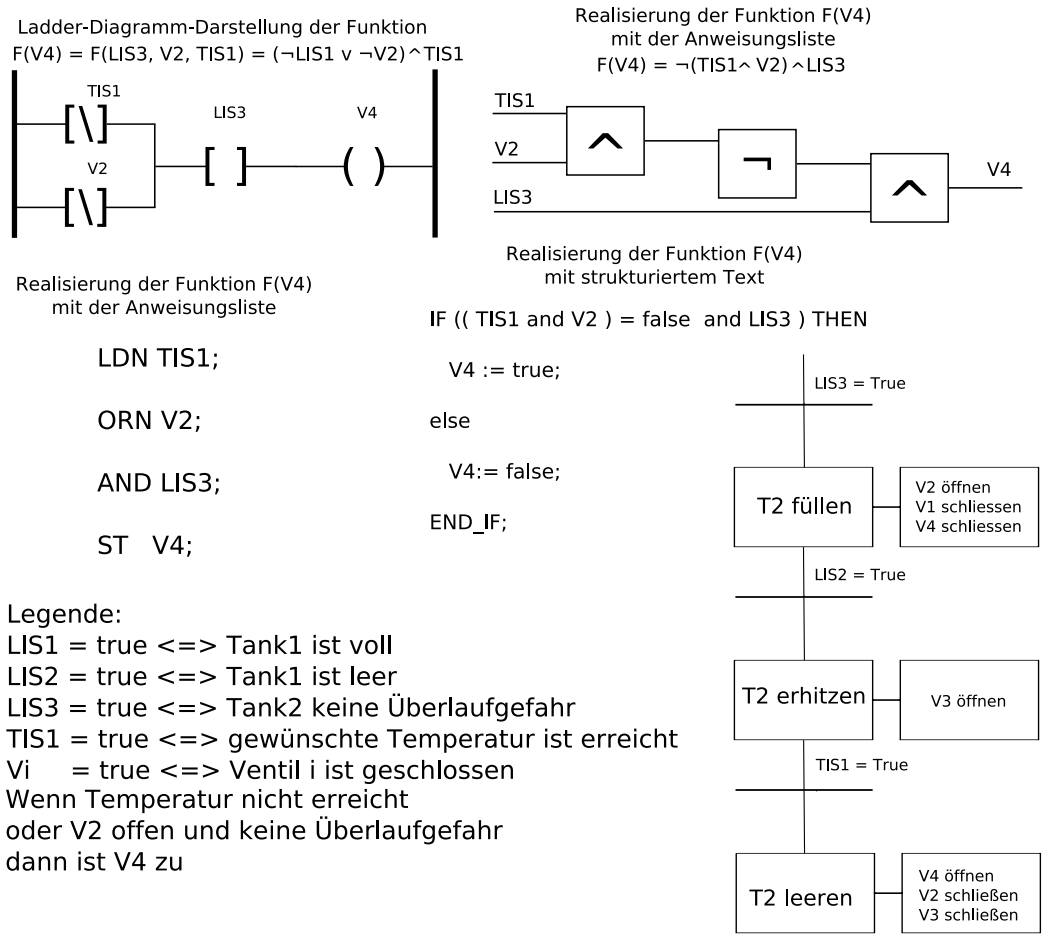
Ein Funktionsblock nach IEC 61131 stellt eine Kapsel für Daten und Algorithmen dar. Die Algorithmen wandeln dabei die Eingangssignale in Abhängigkeit der internen Daten in Ausgangssignale um. Funktionsblöcke werden intern oft mit Hilfe der *Sequential Funktion Chart* (SFC) realisiert.

IEC 61131-3 spezifiziert neben SFC und *Function Block Diagram* (FBD) auch noch *Ladder Diagram* (LD), *Instruction List* (IL) so wie *Structured Text* (ST). Eine LD-Spezifikation entspricht dabei der eines Schaltplans; zwischen den beiden äusseren Schienen liegt eine Spannung an welche je nach Wahrheitswerten der einzelnen durch „[ ]“ oder „[/]“ dargestellten Eingangsvariablen die Spannung an

[1] www.isagraph.com

[2] http://www.software-kompetenz.de/?28498

[3] http://www.boschrexroth.com/business_units/brc/subwebsites/product_catalogue/de/steuerungstechnik_de/Software_de/IndraWorks_de/index.jsp;jsessionid-=aVXxxMCnxvOhgqNCeb




Legende:
LIS1 = true <=> Tank1 ist voll
LIS2 = true <=> Tank1 ist leer
LIS3 = true <=> Tank2 keine Überlaufgefahr
TIS1 = true <=> gewünschte Temperatur ist erreicht
Vi = true <=> Ventil i ist geschlossen
Wenn Temperatur nicht erreicht
oder V2 offen und keine Überlaufgefahr
dann ist V4 zu

Abbildung 1: Realisierung einer Ventilsteuerung mit IEC 61131 Funktionsblöcken.

die durch „( )“ dargestellten Ausgangsvariablen weiterleitet oder nicht. Liegt an „( )“ Spannung an, so ist der Wert der hierdurch dargestellten Variable „1“ und sonst „0“. IL realisiert eine Funktion ähnlich wie Assembler. „LD x“ lädt dabei den Wert der Variable x ins aktuelle Ergebnis (AE) und „OR x“, respektive „AND x“ schreibt den Wert „AE+x“ respektive „AE*x“ ins AE. Wird dem Befehl ein „N“ an gehangen wird die Operation mit dem negierten Wert ausgeführt. Schliesslich kann mit dem „ST“-Befehl das AE in einem Register abgespeichert werden. *Structured Text* stellt eine prozedurale Programmiersprache dar und ist intuitiv zu verstehen, während die Semantik von SFC weitgehend mit der von Petri-Netzen einher geht.

Funktionsblöcke können zu so genannten *Composite Funktion Blocks* zusammengebaut werden um komplexere Anforderungen zu realisieren. Ein Automatisie-

rungsprogramm kann sowohl aus einem einfachen, als auch aus zusammengeschalteten Funktionsblöcken, bestehen. Man unterscheidet zwei Erstellungsarten für Funktionsblöcke, die Typdeklaration und die Instantiierung.

Bei der Typdeklaration werden die Schnittstelle und die Algorithmen des Funktionsblockes festgelegt. Somit steht fest, auf welche Weise Eingangssignale auf Ausgangssignale abgebildet werden, und wieviel Speicher der Block hierzu benötigt. Der so gewonnene Funktionsblock, oder genauer die Funktionsblocktypbeschreibung, kann dann in einer Funktionsblocktypenbibliothek abgelegt werden. Soll der Funktionsblock nun in einer konkreten Anwendung Verwendung finden, so muss er instantiiert werden.

Instantiierung kann für einen Block mehrfach durchgeführt werden. Bei jeder Instantiierung wird lediglich ein weiterer Speicherplatz für den Block reserviert und ein Verweis auf die Funktionsblocktypbeschreibung des entsprechenden Blocks angelegt. Da die reservierten Speicherbereiche disjunkt sind, können Seiteneffekte zwischen verschiedenen Instanzen gleichen Typs ausgeschlossen werden. Diese Speicherreservierung stellt somit auch den wesentlichen Unterschied zwischen Funktionsblöcken und Funktionen dar; im Gegensatz zu Funktionen oder auch Prozeduren bleiben Funktionsblöcke auch nach abgeschlossener Operation für weitere Verwendung im Speicher.

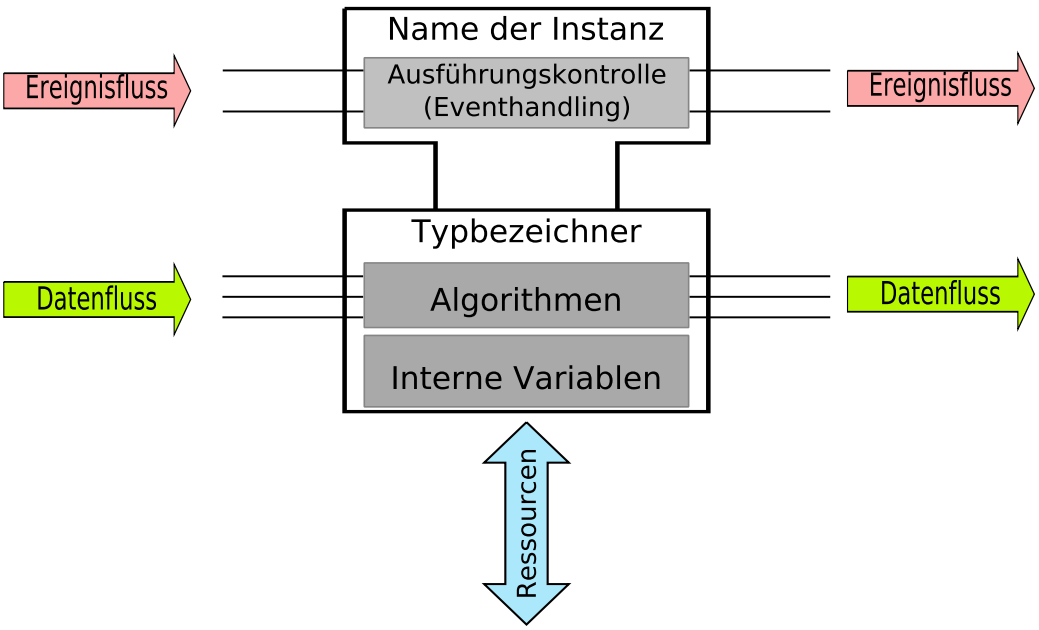



Abbildung 2: Funktionsblöcken nach IEC-61499. Kontroll- und Datenfluss werden getrennt modelliert werden.

Im Laufe der Zeit haben sich diverse Funktionblockbibliotheken von verschiedenen Anbietern auf dem Markt angesammelt und es mussten weitere Standardisierungsschritte durchgeführt werden. Dies war die Arbeit der OPENPLC, welche z.B. *motion control*-Funktionsblöcke einführte [11]. Funktionsblockbasierte Anwendungen wurden anfänglich hauptsächlich in speicherprogrammierbaren Steuerungen (SPS) und Industriellen PCs eingesetzt; später wurden auch Feldgeräte für ihre Verwendung ausreichend leistungsstark.

## 3.2 Interaktion

Bei der Entwicklung von Architekturen für vernetzte Feldgeräte zeigte sich der IEC 61131 Standard als unzureichend. Eine einheitliche Modellierung der Kommunikation zwischen Funktionsblöcken, die auf verschiedenen Felgeräten ausgeführt werden, war in dem Standard nicht vorgesehen. In der Praxis versuchte man mit Proxy-Funktionsblöcken die Ressourcenverteilung vor dem Systemprogrammierer zu verbergen. Die Verwendung von Proxys war in dem Standard aber auch nicht vorgesehen und daher wurde der IEC 61499 Standard eingeführt.

Der IEC 61499 Standard definiert ein modernes Komponentenmodell und erlaubt die Modellierung verteilter Funktionsblöcke. Die einzelnen Funktionsblöcke nach IEC 61499 unterscheiden sich von denen nach IEC 61131 dadurch, dass sie über eine Ausführungskontrolle verfügen, welche die einem eintreffenden Ereignis entsprechende Methode ausführt. Ein Funktionsblock nach IEC 61499 besteht daher aus einem Zustandsspeicher, Algorithmen und der Ausführungskontrolle.Zur Erstellung verteilter Architekturen gemäß dem IEC 61499 Standard wurden eine Reihe von Entwicklungstools geschaffen. Abbildung 2 visualisiert ein Anwendungsmodell gemäß der IEC 61499 Funktionsblöcke.

Ein typischer Aufruf einer Funktion eines Blockes geschieht nun wie folgt:

1. Die Eingangswerte werden gesetzt
2. Die Ereignisse für die Ausführung treffen ein
3. Die Ausführungskontrolle wird aktiviert
4. Diese aktiviert den Algorithmus
5. Die Algorithmen bestimmen aufgrund der Eingangsdaten und der internen Daten die neuen Ausgangsdaten
6. Die Ausführungskontrolle erkennt dass der Algorithmus fertig ist
7. Die Daten werden an die Ausgänge weitergeleitet
8. Das Ausgangsereignis wird erzeugt

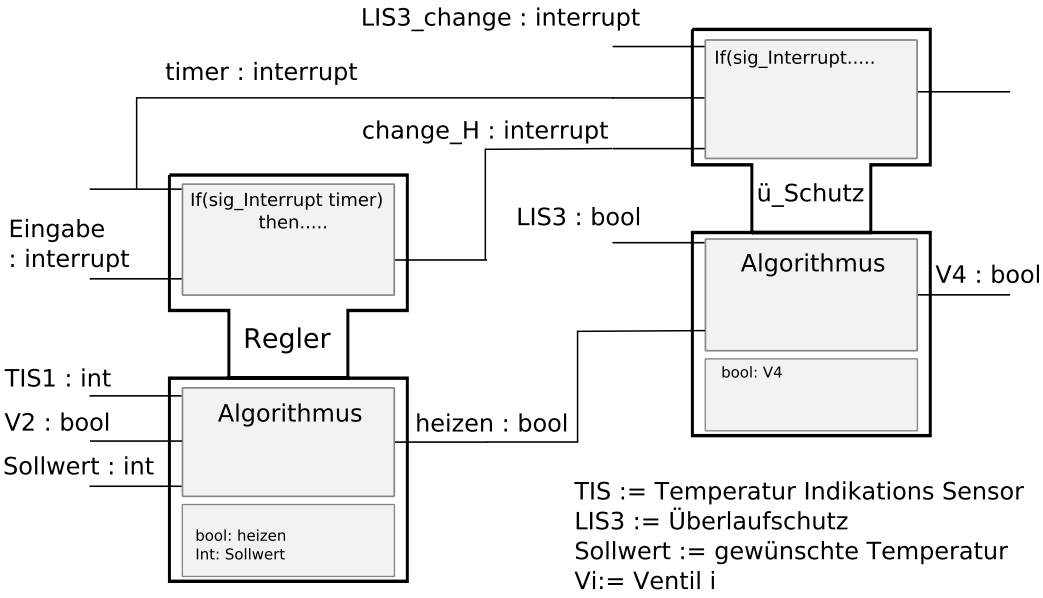



Abbildung 3: Realisierung einer Ventilsteuerung mit IEC 61499 Funktionsblöcken.

Die Abbildung 3 zeigt die Realisierung einer Anwendung mit Hilfe einzelner Komponenten. Zusammenfassend stellen Funktionsblöcke ein standardisiertes Entwicklungsparadigma für Automatisierungsanwendungen dar. Die Weiterentwicklung des IEC 61131 Standards zum IEC 61499 Standard ermöglicht, wie das folgende AUTOSAR Modell, eine übersichtliche Modellierung von verteilten Anwendungen.

# 4 AUTOSAR

*AUTomotive Open System Architecture*, oder kurz AUTOSAR, ist ein von den wichtigsten Automobilherstellern und Zulieferern definiertes Komponentenmodell zur komponentenorientierten Erstellung von verteilten Anwendungen im KFZ. Die Kernidee der Initiative ist es, die positiven Aspekte hauseigener Standards der Mitglieder in einem neuen komponentenbasierten Ansatz zusammenzufassen, um später standardkonforme Softwarekomponenten hersteller- und zuliefererübergreifend austauschen zu können. Im allgemeinen werden so Entwicklungskosten reduziert und Produktzuverlässigkeit erhöht. Daneben werden Flexibilität, Skalierbarkeit und frühzeitige Fehlererkennung (schon auf Modellebene) verbessert.

Erste Bestandteile von AUTOSAR wurden 1997, von der 1993 gegründeten OSEK und 1994 mit VDX zu OSEK/VDX fusionierten Initiative geschaffen [13]. Hierbei handelt es sich um die Spezifikation von OSEK-OS welche von AUTOSAR übernommen wurde. Der OSEK-OS-Standard definiert Anforderungen an OSEK-konforme Betriebssysteme. Ein solches unterscheidet sich von Desktopsystemen zum einen in den Anforderungen an Speicherplatz (hunderte von Megabyte versus wenige Kilobyte) und zum anderen durch seine anwendungsspezifische Erstellung. Einen weiteren OSEK-Standard stellt die *OSEK Implementation Language* OIL dar, welche die Möglichkeit bietet, für eine Applikation erforderliche Betriebssystemdienste normiert zu beschreiben.

Den nächsten Schritt in Richtung AUTOSAR ging die H.I.S. (Herstellerinitiative Software) durch Verschärfung der OSEK-OS- und OSEK-OIL-Spezifikation. Des Weiteren hat H.I.S auch die Anforderungen an Komponenten und Prozesse vereinheitlicht. Auf diese Art versucht die H.I.S. die Wartbarkeit, Prüfbarkeit und Zuverlässigkeit der OSEK-Anwendungen weiter zu verbessern [6]. Schließlich wurde 2003 das AUTOSAR-Konsortium gegründet und mittlerweile liegt der Standard für AUTOSAR in der Version 2.1 vor und erlaubt erste Prototypenrealisierungen.

## 4.1 Übersicht

Der Grundgedanke von AUTOSAR ist die Realisierung eines standardisierten virtuellen Funktionsbusses zur Erleichterung der Integration neuer Softwarekomponenten in eine bestehende Architektur. Hierdurch soll die Erstellung verteilter Software effizienter gestaltet, und deren Architektur handhabbar gemacht werden. Zur Realisierung des VFB muss von der zugrundeliegenden Hardware und ihrer Vernetzung abstrahiert werden. Dies geschieht in AUTOSAR über mehrere Schichten [3]. Abbildung 4 zeigt die Grundstruktur der AUTOSAR-Architektur

Die unterste Ebene der Architektur bildet die Hardware. Auf der Hardware läuft die Basissoftware welche hauptsächlich aus einem OSEK-OS und Treibern für die einzelnen Hardwarekomponenten besteht. Die Basissoftware bietet den Anwendungskomponenten Dienste und standardisierte Kommunikationsschnittstellen an und realisiert des weiteren die hierzu nötige Hardwareabstraktion. Das Zusammenspiel aller, mit dieser Basissoftware ausgestatteten Microcontroller, stellt die AUTOSAR-Laufzeitumgebung dar, welche den VFB realisiert. Der VFB bietet den über ihm liegenden AUTOSAR-Softwarekomponenten AUTOSAR-Schnittstellen an über welche diese interagieren und Basissoftwaredienste nutzen können.

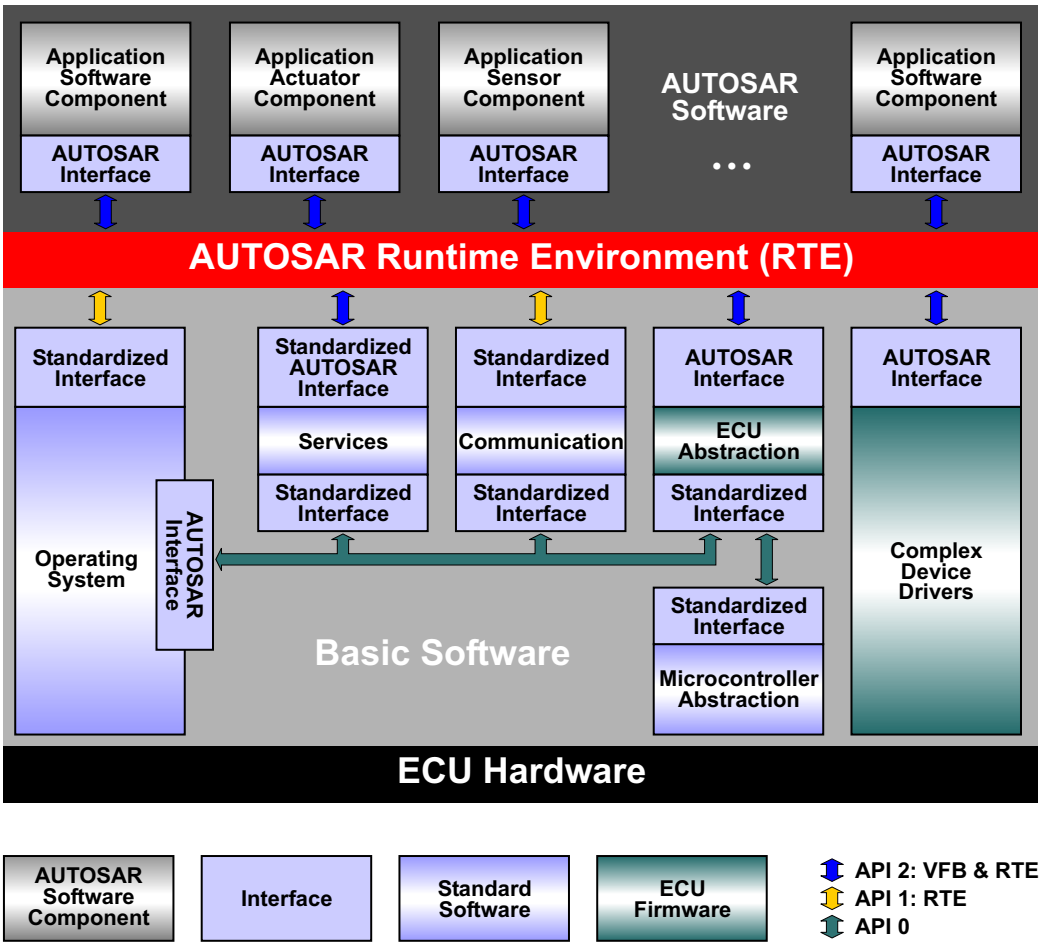



Abbildung 4: AUTOSAR-Architektur

Dank der Abstraktion des VFB von der Netzwerktopologie können Anwendungen durch Entnahme von Komponenten aus einer standardisierten AUTOSAR-Komponentenbibliothek, und deren Zusammenbindung über ihre Schnittstellen realisiert werden. Dieser Vorgang ist Teil des Entwurfs welcher in Abschnitt „4.5 Entwicklungsprozess“ genauer erläutert wird. Die Eigenheiten der einzelnen Schnittstellentypen werden in Abschnitt „4.3 Virtual Function Bus“ erläutert.

## 4.2 AUTOSAR Anwendungssoftware

Um eine Anwendung zu realisieren, werden Anwendungskomponenten im allgemeinen aus einer Komponentenbibliothek entnommen. Um zu wissen welche Funktionalitäten eine Komponente anbietet, ist eine detaillierte semantische und

syntaktische Schnittstellenbeschreibung nötig. Zu jeder Komponente aus einer Bibliothek sollte eine solche verfügbar sein.

Eine solche Bibliothek enthält Komponenten verschiedenster Größenordnung. Kleine Komponenten die etwa nur triviale boolesche Funktionen oder Filter realisieren werden hierbei meist in einer Architektur mehrfach wiederverwendet. Große Komponenten, wie etwa eine Realisierung des ESP werden nur einmal in einer Architektur verwendet. Ein Aufteilen solcher Komponenten in kleinere standardisierte Architekturbausteine liegt nicht im Interesse der Hersteller. Das Zusammenspiel einzelner, diese Komponente realisierender Module, soll zum Schutz von Firmen-Know-How verborgen werden. Daneben wird so garantiert, dass eine solche Komponente integral auf einer Kontrolleinheit läuft was zu einer besseren Reaktivität führt.

Im AUTOSAR-Modell haben Komponenten klar definierte Schnittstellen zur Interaktion, diese werden in R-Ports und P-Ports unterteilt. Jeder Port ist genau einer Komponente zugeordnet und definiert welche Dienste oder Daten eine Komponente braucht oder anbietet. P-Ports stellen Exportschnittstellen dar; die Komponente zu welcher er gehört muss daher in der Lage sein intern die Operationen auszuführen die laut Schnittstellenbeschreibung über diesen Port angestoßen werden. Über sie wird somit im Allgemeinen ein, eventuell parametrisierbarer, Unterprogrammaufruf vollzogen. R-Ports stellen Importschnittstellen dar und benötigen ein AUTOSAR-Interface. Über sie erhält eine Komponente Daten oder stützt sich auf andere Komponenten ab.

In jeder Architektur ist daher jedem R-Port einer Komponente K1 ein P-Port einer Komponente K2 zugeordnet, falls K1 sich auf K2 in irgendeiner Form abstützt. Welche Art von Informationen genau (z.B. Datentypen) über solch ein P-Port/R-Port Paar ausgetauscht werden kann dem *Software Component Template* entnommen werden.

Die Schnittstellenkommunikation stellt die einzige Zugriffsmöglichkeit auf Anwendungskomponenten dar. Es wird sowohl Client/Server als auch Sender/Receiver Kommunikation unterstützt. Abbildung 5 verdeutlicht wie bei komponentenbasierter Anwendungserstellung von der eigentlichen Vernetzungstopologie abstrahiert wird. Es spezifiziert lediglich welche Schnittstellen welcher Komponenten miteinander (und auf welche Art) kommunizieren, nicht jedoch wo sich diese später physikalisch wiederfindet.

Das Zusammenspiel von Anwendungskomponenten entspricht im Wesentlichen dem von Funktionsblöcken nach IEC 61131. Funktionalitäten werden komponentenintern realisiert und Schnittstellen sind von AUTOSAR standardisiert. Anforderungen an die Anwendung werden durch Komposition einzelner Funktionalitä-

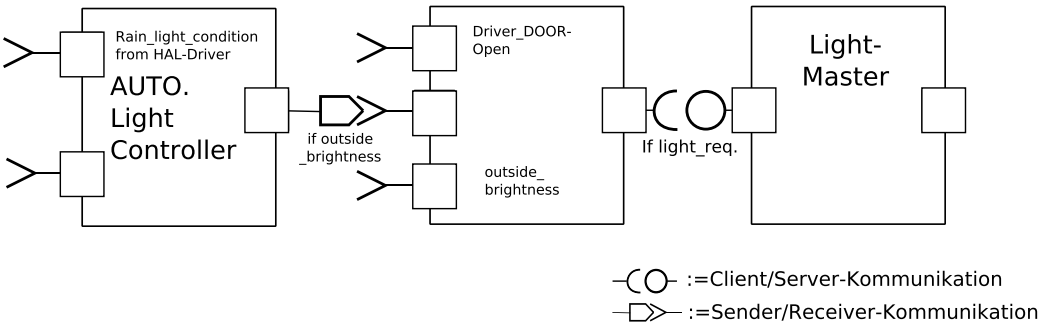



Abbildung 5: Interaktion von Anwendungskomponenten

ten erfüllt. Die Verteilung der einzelnen Anwendungskomponenten auf mehrere Kontrolleinheiten stellt hier bei der Modellierung kein Problem dar, da das *Routing* Aufgabe der Laufzeitumgebung ist. Dies ist auch der Grund dafür, dass die Modellierung auf den IEC 61131, und nicht auf den IEC 61499 Standard zurückgreift.

Hierdurch fehlt jedoch eine klare Trennung von Kontroll- und Datenfluss was eine Visualisierung komponentenreicher Anwendungen unübersichtlich machen kann und später ein aufwendiges manuelles Festlegen eines Prozessschedulings für die einzelnen Kontrolleinheiten erfordert (siehe Abschnitt 4.5.2). Zur Verbesserung von Fehlererkennung und Überprüfbarkeit der Erfüllung nicht-funktionaler Anforderungen auf Modellebene wäre eine Erweiterung des Anwendungserstellungsparadigmas, entsprechend IEC 61499 wünschenswert. Da die aus dem Modell erstellte Anwendungsspezifikation in XML hinterlegt wird und Werkzeuge zum Abspeichern von IEC 61499 Funktionsblöcken in XML erhältlich sind sollte dies bei vertretbarem Aufwand möglich sein [7]. Somit würden Kontroll- und Datenfluss die Interaktion von Komponenten übersichtlicher unterteilen.

Derzeit wird jedoch von allen nicht durch Funktionsblöcke nach IEC 61131 beschreibbaren Kommunikationsdetails, wie etwa explizitem Kontrollfluss, der physikalischen Lage der Kommunikationspartner oder dem benutzten physikalischen Kommunikationsmedium im AUTOSAR-Modell abstrahiert, und die eigentlich komplexe Vernetzungstopologie als VFB dargestellt.

## 4.3 Virtual Function Bus

Im AUTOSAR-Modell interagieren Komponenten ausschließlich über den VFB welcher standardisierte Schnittstellen für Softwarekomponenten zur Verfügung stellt [4]. Um eine neue Komponente in die Architektur einzuhängen, muss die-

se über eine AUTOSAR-konforme Schnittstelle verfügen welche im virtuellen Modell mit einer AUTOSAR-Schnittstelle des VFB verbunden wird. Daher wird der VFB in der Literatur auch als „Abstraktion der Verbindung aller AUTOSAR-Softwarekomponenten des Fahrzeugs“ definiert.

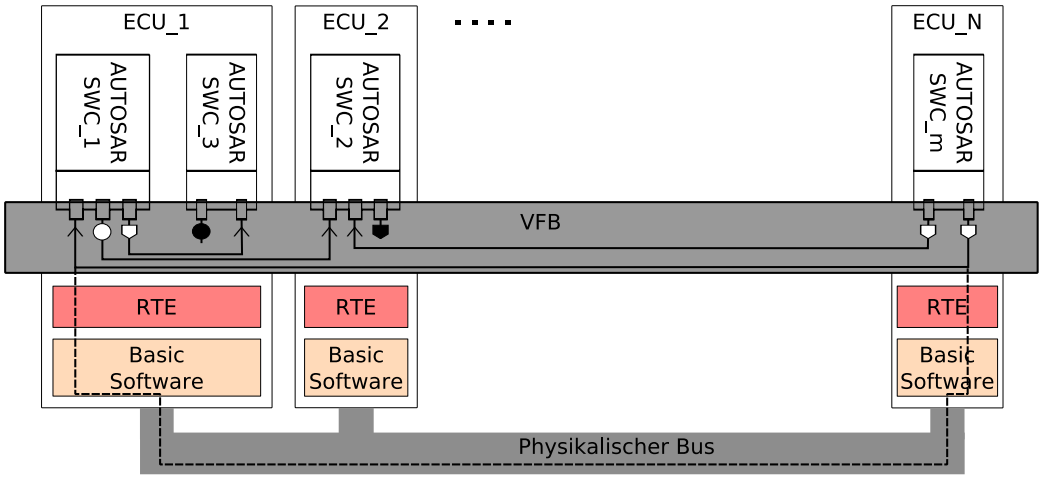



Abbildung 6: Virtual Function Bus: Die Abbildung zeigt wie von der Vernetzung der Kontrolleinheiten durch den VFB abstrahiert wird.

Dank der Abstraktion des VFB kann die Spezifikation der Kommunikation zwischen Anwendungskomponenten untereinander, oder zwischen Anwendungskomponenten und Basisdiensten völlig unabhängig von der zugrunde liegenden Hardware erfolgen [4]. Anwendungskomponenten müssen nicht wissen wie, von welcher Kontrolleinheit zu welcher, die Kommunikation verläuft. Diese Funktionalität wird durch präzise Festlegung von Kommunikationsmustern realisiert. Die Kommunikationsmuster sind durch Protokolle festgelegt welche wie AUTOSAR-Dienste von der Basissoftware implementiert werden. AUTOSAR-Dienste stellen eine Erweiterung der Funktionalität für Nutzer des VFB dar. So könnte etwa ein Basis-Dienst *set_Timer_t1(value, prescaler, topVal)* von einer Anwendungskomponente aufgerufen werden, welcher dann eine ganze Reihe von hierzu nötigen Assemblerbefehlen an die entsprechenden Kontrolleinheiten weiterleiten würde. Die Abbildung 6 zeigt wie sich in jeder Kontrolleinheit die Anwendungskomponenten auf die eigens für sie von der Basissoftware realisierte RTE zur Dienstnutzung und zu Routingzwecken abstützen. Auf diese Weise wird die *Deployment-Anomalie* [4] vermieden.

[4]Die Deployment-Anomalie liegt vor wenn Neuverteilung von Anwendungskomponenten auf Kontrolleinheiten ein Abändern der Softwarearchitektur erzwingt.

Da Komponenten nur über Schnittstellen zugänglich, und somit vernetzbar, sind gilt es nun deren Rolle bei der Realisierung des VFB genauer zu erläutern. In AUTOSAR gibt es drei Typen von Schnittstellen , *AUTOSAR Schnittstellen*, *standardisierte Schnittstellen* und *standardisierte AUTOSAR Schnittstellen*.

*AUTOSAR Schnittstellen* dienen dem Austausch von Informationen zwischen Anwendungskomponenten und definieren deren Ports. Ausnahmen stellen der standardisierte Zugriff auf Kontrolleinheiten und auch die Kommunikation mit komplexen nicht standardisierbaren Treibern dar. Dabei ist die Portbeschreibung unabhängig von den verwendeten Programmiersprachen, Kontrolleinheiten und Netzwerktypus. Für AUTOSAR-Schnittstellen macht es daher keinen Unterschied ob sie Anwendungskomponentenkommunikation kontrolleinheitintern oder übers Boardnetz realisieren.

*Standardisierte Schnittstellen* sind hingegen Anwendungsprogrammierschnittstellen (Header). Sie sind von AUTOSAR her genormt und sind meist abhängig von der Programmiersprache (meist C), in welcher die an der Schnittstelle angebotenen Funktionalitäten, implementiert werden. Sie finden hauptsächlich Verwendung in der Kommunikation zwischen Komponenten die immer auf gleichen Kontrolleinheit laufen und sind somit meist in den unteren hardwarenahen Schichten einer Architektur zu finden. Ausnahmen sind hier der direkte Zugriff auf das Betriebssystem (Betriebsmodus festlegen) oder Kommunikation mit der Basissoftware.

Schließlich bleiben noch die *standardisierten AUTOSAR-Schnittstellen* zu erwähnen; diese sind vollständig in Syntax und Semantik von AUTOSAR genormt und bieten den AUTOSAR-Softwarekomponenten standardisierte Basissoftwaredienste an. Diese Schnittstellen sind somit sowohl unabhängig von der auf der RTE aufsetzenden Architektur, als auch von der zu Grunde liegenden Hardware und bieten Anwendungskomponenten Basisdienste an.

## 4.4 Basic Software

Die Laufzeitumgebung welche mit Hilfe der benötigten Basissoftwarekomponenten realisiert wird, implementiert zur Laufzeit den VFB. Die RTE abstrahiert hierbei im Allgemeinen über mehrere Schichten von der zugrunde liegenden Hardware. Dies gilt sowohl für die Verwendung der Kontrolleinheiten als auch der Kommunikationshardware, also den physikalischen Bussen. Dieses Schichtenkonzept wird im folgenden genauer, von der RTE hin zur zugrunde liegenden Hardware beschrieben.

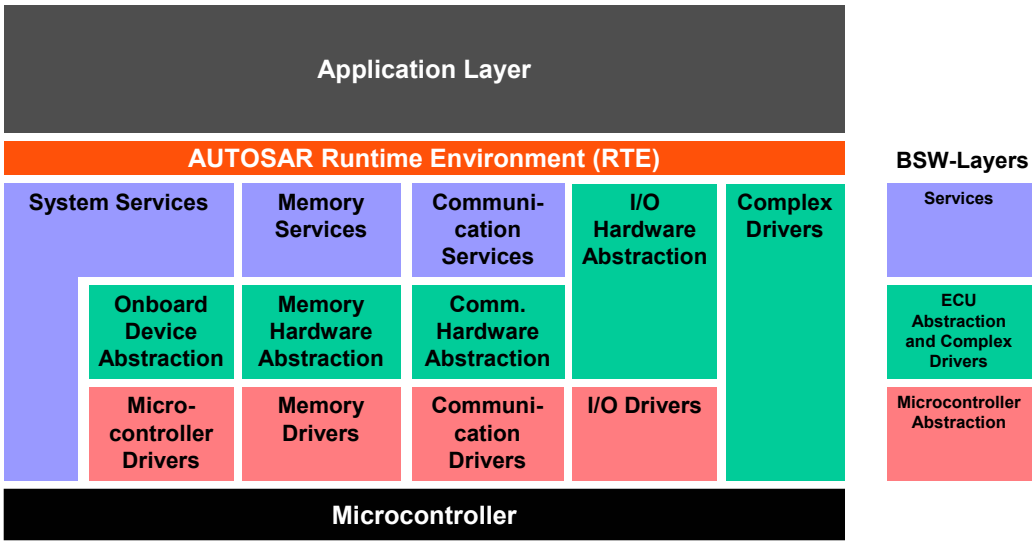



Abbildung 7: AUTOSAR Schichtenkonzept

Abbildung 7 skizziert die drei Schichten der AUTOSAR-Basissoftware. Die unterste Schicht stellt den *Microcontroller Abstraction Layer*, die mittlere den *ECU Abstraction Layer* und die oberste den *Service Layer* dar.

### 4.4.1 Service Layer

Als erste Schicht, gleich unter der Laufzeitumgebung befindet sich der *Service Layer* (Dienst-Schicht). Dieser ist unterteilt in Systemdienste, Speicherdienste und Kommunikationsdienste.

Kommunikationsdienste stellen eine einheitlich Schnittstelle zum Bordnetz zur Kommunikation zwischen Anwendungen bereit. Sie realisieren standardisierte Dienste zur Netzwerkorganisation und verkapseln anwendungsspezifische Protokolle und Botschaften. Darüber hinaus gilt es ein standardisiertes Interface zur Netzwerkdiagnose anzubieten.

Speicherdienste verwalten nichtflüchtigen Speicher. Sie stellen mithilfe eines „NVRAM Managers“ einen RAM-Spiegel für schnellen Speicherzugriff für Applikationen zur Verfügung. Die Hauptfunktionen dieser Module sind einheitliche Bereitstellung volatiler Daten für Anwendungen, Speicher- und Ladefunktionalitäten sowie die Abstraktion von der Art der Speicherelemente.

Als letzte Komponente dieser Schicht bleiben noch die Systemdienste zu nennen, welche direkt auf die Hardware aufsetzen, und somit eigentlich zwei Abstrak-

tionsstufen überspringen. Hierdurch wird es für Module aller Schichten möglich diese zu benutzen, was allerdings mit dem Preis einer teilweisen hard- und softwareabhängigen Implementierung erkauft wird. Die Komponente stellt Basisdienste sowohl für Anwendungen, als auch für andere Basisdienste zur Verfügung.

### 4.4.2 *ECU Abstraction Layer*

Die darunter liegende Schicht ist der *ECU Abstraction Layer*. Dieser unterteilt sich „vertikal“ in die *Onboard Device Abstraction* (ODA), die Speicherhardwareabstraktion (SHA), Kommunikationshardwareabstraktion (KHA), der Ein/Ausgabe-Hardwareabstraktion (EAHA) und die *Complex Device Drivers* (CDD). Hierbei abstrahiert die ODA von speziellen Kontrolleinheiten, die SHA von der physikalischen und virtuellen Speicherverteilung und die KHA von speziellen Kommunikationsprimitiven. Diese Schicht setzt also die Abstraktion des *Microcontroller Abstraction Layer*s fort und bietet dem darüber gelegenen *Service Layer* standardisierte Zugriffsmechanismen auf Kontrolleinheiten, Kommunikation und Speicher an.

Des weiteren enthält der *ECU Abstraction Layer* die IO-Hardwareabstraktion. Diese ist neben den CDD die einzige Modulansammlung dieser Schicht, auf welche die Laufzeitumgebung direkt aufsetzt. Ihre Funktion ist die Signaldarstellung und natürlich die Abstraktion von der ECU Hardware.

Schließlich enthält der *ECU Abstraction Layer* noch die CDD. Diese verstoßen eigentlich gegen das Prinzip der schichtenartigen Abstraktion, da sie sowohl direkt auf die Hardware aufsetzen als auch ihre Dienste direkt an die RTE weitergeben. Ihre Funktion ist die Realisierung hardwarespezifischer funktionaler und oft zeitkritischer Anforderungen zur Steuerung von Sensoren und Aktoren. Einspritzkontrolle und Ventilsteuerung sind u.a. typische Anwendungsgebiete dieser Module. CDDs bestehen teils aus herstellereigenen, echtzeitfähigen Basissoftwarekomponenten und bietet einen direkten standardisierten Zugriff auf spezielle Hardwarekomponenten mit meist stark eingeschränkten Ressourcen an.

### 4.4.3 *Microcontroller Abstraction Layer*

Die unterste Abstraktionsschicht ist der *Microcontroller Abstraction Layer* dessen Hauptfunktion eine erste Abstraktion von den verwendeten Microchips und Speicherbausteinen darstellt. Er ist abhängig von den zugrunde liegenden Microchips und Speicherbausteinen und soll an den *ECU Abstraction Layer*, standardisierte Microcontroller-Dienste weiterleiten. Man kann diese Schicht auch verein-

facht als Treiberschicht interpretieren. Unterteilt ist sie in Treiber für Ein- und Ausgabe, Kommunikation der elektronischen Kontrolleinheiten untereinander, Speicherorganisation und Treiber für die eigentlichen Microcontroller. Die physikalissche Kommunikation erfolgt hierbei oft über mehrere verschiedene Bussysteme welche in Thema 5 des Seminarbandes in den Kapiteln 5.2.2 und 5.2.3 näher erläutert sind.

## 4.5 Entwicklungsprozess

Abbildung 8 zeigt welche Abhängigkeiten bei der Entwicklung von Architekturen mit Hilfe des AUTOSAR-Modells zu berücksichtigen sind. Die Semantik der Darstellung ähnelt der von Petrinetzen; liegen alle eingehenden Arbeitsprodukte (Rechtecke) eines Werkzeugs (Kreise) vor, so können alle ausgehenden erzeugt werden. Die Methodik definiert keine totale zeitliche Ordnung der Arbeitsprodukte sondern deren kausalen Zusammenhang. Diese Abhängigkeiten zwingen dem Entwickler ein bestimmtes Vorgehen, die AUTOSAR-Methodik, auf. Während das AUTOSAR-Modell festlegt wie eine Komponente zu beschreiben ist, definiert die Methodik wie diese Beschreibungen zu benutzen sind. Arbeitsprodukte liegen meist in Form von XML-Dateien vor.

### 4.5.1 Software Entwurf

Um Komponenten in eine vorhandene AUTOSAR-Architektur eingliedern zu können oder um aus ihnen Anwendungen zusammenzubauen, müssen diese in einer bestimmten Form vorliegen. Hierzu müssen Kriterien erfüllt sein die garantieren dass die Komponente mit dem VFB kompatibel ist. Eine Komponente muß daher über eine genormte Schnittstelle verfügen und sollte bei Lieferung aus Objekt/Quellcode und *AUTOSAR Software Component Description* (AUTOSAR Softwarekomponenten Beschreibung) (ASB) bestehen [2].

Da man auch diese Beschreibungen wieder in einem standardisierten Format haben möchte, etwa um sie über eine Standardeingabemaske einlesen/ausgeben zu können, wird die *AUTOSAR Software Component Description* durch Ausfüllen des *Software Component Template*s, einer standardisierten Vorlage für *AUTOSAR Software Component Description*s festgehalten. Die Beschreibung sollte keine Anforderungen an spezifische Kontrolleinheiten, Microcontroller oder Speicherverteilungen beinhalten da dies Aufgabe der *System Constraint Description* ist.

### 4.5.2 AUTOSAR Methodik

Zur Erstellung einer verteilten Anwendung nach AUTOSAR Methodik sind eine Reihe von werkzeugunterstützten Transformationen zwischen Arbeitsprodukten zu realisieren. Zunächst gilt es den *System Configuration Input* durch Ausfüllen von Eingabemasken zu definieren. Dieser besteht aus *SW Component Description*, *ECU Resource Description* und *System Constraint Description*. Mit Hilfe des *AUTOSAR System Generator*s kann dann die *System Configuration Description* und die Systemkommunikationsmatrix erzeugt werden. Dabei definiert die *SW Component Description* die benötigten Ports, und deren Semantik, für alle vorkommenden Komponenten. Die *ECU Resource Descriptions* beschreibt die Ressourcen aller verfügbaren Steuereinheiten die *System Constraint Description* gibt an wie diese miteinander verbunden sind.

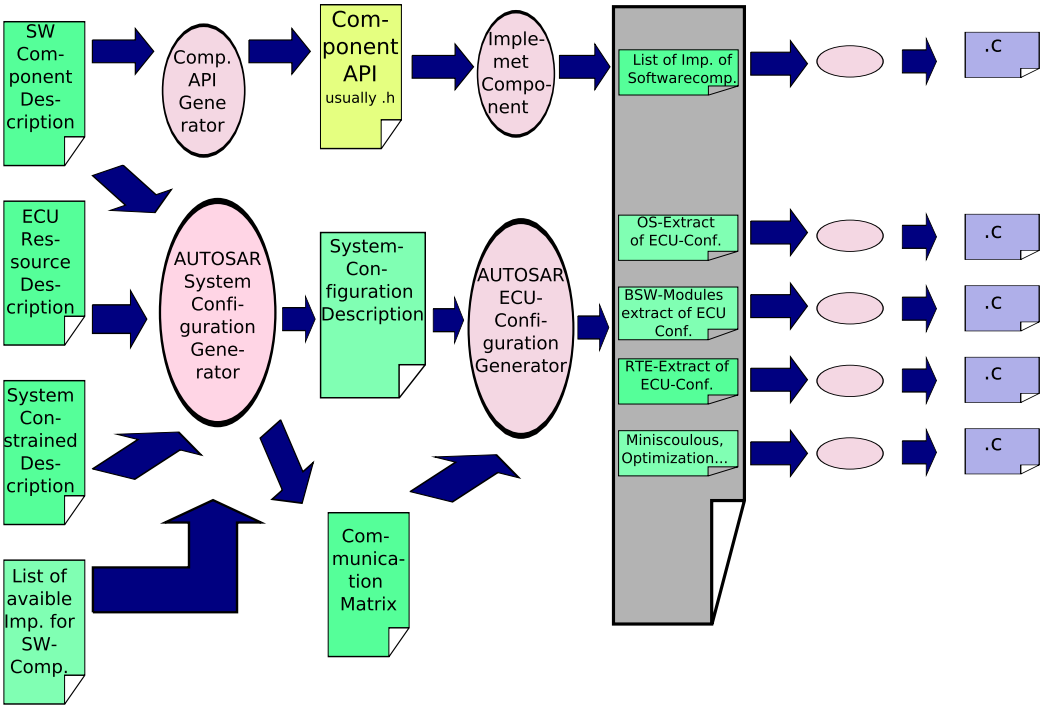



Abbildung 8: AUTOSAR Methodik

Der *AUTOSAR System Generator* entscheidet zunächst welche Softwarekomponente später auf welcher Kontrolleinheit laufen soll. Dies wird auch als *Mapping* bezeichnet (Siehe Abbildung 6) und bedarf qualifizierter personeller Unterstützung. Im Allgemeinen stützt sich die Systemkonfiguration auch noch auf eine Liste aller verfügbaren Implementierungen aller auftretenden Softwarekomponenten

ab, da die konkrete Implementierung einer Komponente Einfluss auf das *Mapping* haben kann.

Nach dem *Mapping*, steht die Aufgabe jeder Kontrolleinheit des Systems fest, und ist in den einzelnen Auszügen der *System Configuration Description* beschrieben. Daneben enthält die *System Configuration Description* die *Mapping*-Entscheidungen und die für jede Komponente gewählte Implementierung[5]. Neben der *System Configuration Description*, wird während der Systemkonfiguration auch die *Systemkommunikationmatrix* erzeugt. Sie beschreibt die gesamte Vernetzung der Softwarekomponenten untereinander.

Zur Konfiguration der einzelnen Kontrolleinheiten extrahiert der *AUTOSAR ECU configuration generator (AECG)* jeweils, für jede Kontrolleinheit den passenden Auszug aus der *System Configuration Description*. Dies geschieht völlig automatisiert da hier lediglich alle, eine bestimmte Kontrolleinheit betreffenden Informationen, aus der *System Configuration Description* in ein neues Dokument überschrieben werden müssen. Das neu entstandene Dokument stellt den ECU-Auszug der *System Configuration Description* der Kontrolleinheit dar.

Als nächstes wird mit Hilfe des AECG der Hauptteil *ECU Configuration Description* erstellt. Dies stellt den anspruchsvollsten Schritt der Methodik dar. Es gilt, die benötigte Basissoftware in Abhängigkeit der auf die Kontrolleinheit gemappten Softwarekomponenten als auch das Scheduling für die einzelnen, später auf einer Kontrolleinheit laufende Prozesse zu bestimmen. Des weiteren muss unter Zuhilfenahme der Kommunikationsmatrix ein Scheduling für die Kommunikation der Softwarekomponenten erstellt werden. Dieser Vorgang ist heute noch nicht automatisierbar und bedarf in der Regel erfahrenem Personals. Ein Grund hierfür ist die schon in Kapitel 4.2 kritisierte Anwendungsmodellierung ähnlich dem IEC 61131 Standard, welcher keine Modellierung eines expliziten kontrollflussbasierten Schedulings erlaubt, was an dieser Stelle teilweise nachgeholt werden muss.

Parallel, werden aus der *SW Component Description* mit Hilfe des *Component API Generator* passende API's für die einzelnen Komponenten erzeugt. Hierbei handelt es sich im allgemeinen um Header-Dateien welche die Syntax der Schnittstellen der Komponenten festlegen. Auf Basis dieser API´s kann dann eine Liste der in Frage kommenden Softwarekomponentenimplementierungen, als Teilmenge der anfänglichen Liste von Implementierungen, erstellt werden. Diser Schritt ist in Abbildung 8 als *Implement Component* eingetragen und völlig automatisierbar.

[5]Oft werden Komponenten mit gleicher Funktionalität verschieden implementiert um später eine geeignete Implementierung wählen zu können.

Als Produkt der beiden Arbeitsschritte *Implement Component* und AECG liegt die fertige *ECU Configuration Description* vor. Diese enthält einen Laufzeitauszug, Betriebssystemauszug und Basissoftwaremodulauszug der Kontrolleinheit und eine Liste von gewählten Implementierungsmöglichkeiten für die auf der Kontrolleinheit laufenden Softwarekomponenten. Für jeden dieser Auszüge existieren Werkzeuge um diese in C-Code zu übersetzen. Die einzelnen C-Dateien (je mit Header-Dateien) werden anschließend in Maschinen-/Objectcode übersetzt, welcher dann ggf. zu einer ausführbaren Datei zusammengelinkt und auf die dem Mapping entsprechende Kontrolleinheit überspielt wird.

In der Praxis müssen diese Schritte meist mehrfach, mit verschiedenen Mapping- und Schedulingentscheidungen, durchgeführt werden bis eine zufriedenstellende Gesamptkonfiguration der einzelnen Kontrolleinheiten vorliegt. Hier ist wieder eine gewisse Analogie zu Thema 5 Kapitel 3 *Universal Communication Model* zu erkennen, da auch hier der VCC Benutzer solange Parameter einstellt bis die gewünschte Konfiguration vorliegt.

### 4.5.3 Komponentenerstellung

Die Komponente stellt die wichtigste strukturelle Einheit des AUTOSAR-Modells dar. Erstellt wird eine Anwendungskomponente in 4 Schritten. Als erstes wird eine *Component Internal Behavior Description* (CIBD) erstellt. Dies geschieht durch das Ausfüllen von Standard-Eingabemasken und dem damit verbundenen Bearbeiten des *Software Component Template*s. Die CIBD legt dabei fest, auf welche Ereignisse die Komponente später reagieren soll und welche lauffähigen Entitäten durch diese angestoßen werden sollen. Die genaue Funktion der Entitäten ist dabei jedoch noch nicht bestimmt. Des Weiteren findet sich in der CIBD ein Verweis auf die *Component Type Description* welche die Anforderungen an Basisdienste enthält. Diese Anforderungen müssen bei der API-Erstellung und späteren Implementierung berücksichtigt werden.

Als zweites erzeugt der *Component API Generator*, auf Basis der CIBD, die API der Komponente woraufhin ein Softwareentwickler auf Basis der CIBD und API die Implementierung realisiert, was den dritten Schritt darstellt. Die CIBD legt dabei fest auf was die Komponente wie zu reagieren hat und die API standardisiert die Syntax dieser Funktionalität. Als Arbeitsprodukt dieses Schrittes liegt im Allgemeinen eine Softwarekomponente in Form von C-Code, eine um Implementierungsdetails erweiterte CIBD und eine *Component Implementation Description* (CID) vor. Die CID enthält hauptsächlich vom Compiler geforderte Compileroptionen. Als letzter Schritt gilt es, die C-Dateien und Header-Dateien, den CID-Parametern entsprechend, zu kompilieren. Dies erzeugt eine fertige Softwa-

rekomponente in Form von *Maschinen-/Object-Code*. Dieser Code kann nun auf Effizienz bezüglich Speicher- und Prozessorressourcen getestet, und gegebenenfalls inklusive CID und CIBD in einer Bibliothek hinterlegt werden.

# 5 Fazit

Komponentenmodelle sind sehr verbreitet in der heutigen Softwaretechnik. Je ähnlicher die zu erstellenden Architekturen sich sind, je einfacher kann mithilfe eines solchen Modells die eine aus der anderen gewonnen werden. Je komplexer die Architekturen sind, um so hilfreiche sind die Modelle zur Veranschaulichung relevanter Entwurfsentscheidungen. In dieser Arbeit wurden Funktionsblöcke und das AUTOSAR-Komponentenmodell vorgestellt. Anhand der Evolution von IEC 61131 Funktionsblöcken zu IEC 61499 Funktionsblöcken wurde illustriert wie und warum sich ein klassisches Komponentenmodell durch eine klare Trennung von Kontroll- und Datenfluss zu einem modernen und zur Modellierung verteilter Architekturen geeigneten Komponentenmodell entwickelte.

Des Weiteren wurde das AUTOSAR-Komponentenmodell vorgestellt, welches es dem Anwendungsentwickler erlaubt Anwendungen aus bestehenden Komponenten zusammen zu bauen. Hierbei kann der Entwickler die spätere Verteilung der Komponenten über mehrere Kontrolleinheiten ignorieren, da das *Routing* der schnittstellenbasierten Kommunikation zwischen Anwendungskomponenten, von Basissoftwarekomponenten realisiert wird. Im AUTOSAR-Modell wird von der gesamten Netzwerktopologie der Kontrolleinheiten abstrahiert und diese als VFB dargestellt. Empfehlenswert sei dennoch ein Übergang zu einem an IEC 61499 angelehnten Erstellungparadigma für Anwendungskomponenten zur Erleichterung des Festlegens eines Schedulings während der Kontrolleinheitskonfiguration.

Klare Trennung von Daten- und Kontrollfluss und Abstraktion von Komponentenverteilung sollen also Entwicklern von Anwendungen für verteilte Systeme die Arbeit erleichtern. Darüber hinaus führt die Wiederverwendung von Architekturbausteinen zu einem insgesamt reduzierten Arbeitsaufwand und die standardisierte AUTOSAR-Methodik sollte künftigen Entwicklern Umschulungen beim Arbeitgeberwechsel ersparen. Auf diese Weise tragen die vorgestellten Modelle zu Effizienzsteigerungen und Kosteneinsparungen bei der Entwicklung verteilter Anwendungen, vor allem in der Automobilindustrie, wo viele komplexe und doch ähnliche Anwendungsarchitekturen vorzufinden sind, bei.

# Literatur

[1] ADAM DUNKELS, BJÖRN GRÖNVALL, THIEMO VOIGT: *Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors*. Swedish Institute of Computer Science, Swedish Institute of Computer Sciencefadam,bg,thiemog@sics.se, 2005.

[2] AUTOSAR GBR: *AUTOSAR Methodology*. AUTOSAR GBR, http://www.autosar.org/download/AUTOSAR_Methodology.pdf, 2006.

[3] AUTOSAR GBR: *AUTOSAR Technical Owerview*. AUTOSAR GBR, http://www.autosar.org/download/AUTOSAR_TechnicalOverview.pdf, 2006.

[4] AUTOSAR GBR: *The Virtual Funktion Bus*. AUTOSAR GBR, http://www.autosar.org/find02_ns6.php, 2006.

[5] A., W. RUH U.: *Middleware, Enterprise Application Integration.* Wiley, www.springer-ny.com, 2001.

[6] DAIMLERCHRYSLER AG: *OSEK OS Extensions for ProtectedApplications*. DaimlerChrysler AG, http://www.automotive-his.de/download/HIS_ProtectedOSEK10.pdf, 2003.

[7] INC, ICS TRIPLEX ISAGRAPH: *IEC 61499 Funktion Block Model*. ICS Triplex ISaGRAPH Inc, www.isagraph.com, 2006.

[8] INTERNATIONAL ELECTROTECHNICAL COMMISSION: *IEC 61131-3 Ed. 1.0 en:1993: Programmable controllers — Part 3: Programming languages*. International Electrotechnical Commission, Geneva, Switzerland, 1993.

[9] INTERNATIONAL ELECTROTECHNICAL COMMISSION: *IEC 61499-1 Ed. 1.0 en:2005: Function blocks — Part 1: Architecture*. International Electrotechnical Commission, Geneva, Switzerland, 2005.

[10] PHILIP LEVIS, SAM MADDEN: *The Emergence of Networking Abstractions and Techniques in TinyOS*. University of California, EECS Department, University of California, Berkeley 2150 Shattuck Ave., 2006.

[11] PLCOPEN: *Function blocks for motion control, Version 1.0*. PLCopen - Technical Committee 2, http://plcopen.org/TC2/Motion/MC_Extensions.htm, 2001.

[12] SZYPERSKI, CLEMENS: *Component Software*. Addison-Wesley, ISBN 0-201-17888-5, 1998.

[13] WIKIPEDIA: *OSEK*. Wikipedia, http://de.wikipedia.org/wiki/OSEK, 2006.