Entwicklungsprozess für vernetzte Elektroniksysteme

Development process for networked electronic systems

Dipl.-Ing. (FH) **B. Hardung**, AUDI AG, Ingolstadt
Dipl.-Ing. (FH) **M. Wernicke**, Vector Informatik GmbH, Stuttgart
Dr. **A. Krüger**, (MBA), Dipl.-Ing. **G. Wagner**, AUDI AG, Ingolstadt
Dipl.-Inform. **F. Wohlgemuth**, DaimlerChrysler AG, Sindelfingen

Inhalt

Die Komplexität der Software in heutigen Kfz-Elektroniksystemen ist bedingt durch die Vernetzung und die Anzahl der Steuergeräte sehr hoch. Der hohe Vernetzungsgrad der Fahrzeugfunktionen und die Verteilung der Entwicklungstätigkeit auf viele Unternehmen erfordern neue Entwicklungsprozesse für diese software-intensiven Systeme. In diesem Beitrag werden die Ziele für derartige Entwicklungsprozesse, die sich daraus ergebenden Erwartungen an eine unterstützende Methodik sowie ein möglicher Lösungsansatz vorgestellt.

Summary

The complexity of today's automotive electronics systems is very high, which is largely due to networking and the number of electronic control units. The high interconnectivity of the vehicle functions and the distribution of the development efforts on a number companies calls for new development processes for these software-intensive systems. This paper presents the objectives of such development processes, the expectations on a design methodology, and finally a possible approach.

1 Einleitung

Die Automobilindustrie steht derzeit vor einer großen Herausforderung: Im Jahre 2010 werden allein 13% der Herstellkosten eines Kraftfahrzeugs auf Software entfallen /1/. Dieser Wert beträgt derzeit 4%, was die rasante Zunahme der Bedeutung von Software für die Automobilindustrie aufzeigt. Diese Tatsache bedeutet eine große Veränderung für die Elektronikentwicklung. Immer mehr Steuergeräte, deren Funktionen in verstärktem Maße vernetzt

sind, müssen bei gleichzeitig verkürzten Entwicklungszeiten zur Serienreife gebracht werden. Um diesen Herausforderungen zu begegnen, ist ein veränderter Entwicklungsprozess notwendig, der sich von der Ausrichtung an einzelnen Hardwarekomponenten löst und in Richtung Funktionsorientierung bewegt.

2 Elektronik-Entwicklungsprozesse

Ein Ziel ist der Aufbau einer Bibliothek aus wiederverwendbaren Funktionen, die aus selbst erstellten oder zugekauften Software-Komponenten besteht. Eine solche Bibliothek soll als Basis dienen können, um bei der Entwicklung von einzelnen Baureihen auf fertig entwickelte und getestete Funktionsmodule zurückgreifen zu können.

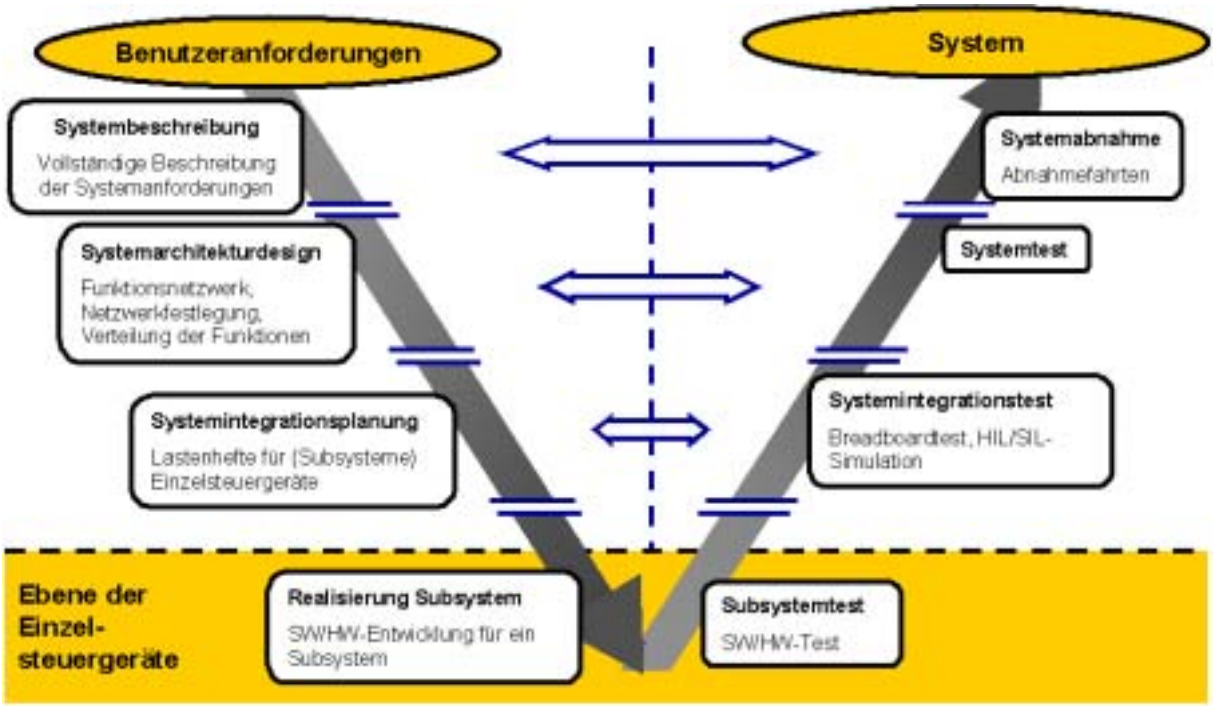

Abbildung 1: Die Entwicklung des Systems Fahrzeug aus der Elektroniksicht / The development of the system car from the view of electronics

Künftige Entwicklungsprozesse müssen in der Lage sein, ein Portfolio an elektronik-basierten Fahrzeugfunktionen auf eine nahezu beliebige Fahrzeugarchitektur abbilden zu können. Dazu sind neue Methoden und Techniken erforderlich, die sich schrittweise in die existierenden Entwicklungsabläufe einfügen lassen, sowie die vorhandene Infrastruktur einerseits im Steuergerät (Standard Software Core) und in der Steuergerätumgebung (wie Fertigung und Service) nutzen und Schritt für Schritt erweitern. In Abbildung 1 ist ein am V-Modell /2/ orientierter Elektronikentwicklungsprozess dargestellt, wie er in der Automobilindustrie angewen-

det wird. In Zukunft muss eine durchgängige Toolkette vorhanden sein, die idealerweise den kompletten Entwicklungsprozess begleitet.

3 Anforderungen an eine Entwicklungsmethodik

3.1 Einsatzzwecke

Eine Entwicklungsmethodik muss unterschiedliche Einsatz-Szenarien ermöglichen.
Zunächst können diese nach dem Entwicklungsziel unterschieden werden:

- Entwicklung eines Steuergerätes mit Integration verschiedener Softwaremodule und Standardsoftwarekomponenten
- Entwicklung eines Steuergerätenetzwerkes
- Entwicklung einer Funktion in einem Steuergerätenetzwerk

Zudem muss die Verwendung verschiedener Bussysteme wie CAN /3/, LIN /4/, MOST /5/ oder Flexray /6/ unterstützt werden.

3.2 Wiederverwendung von Funktionen

Die Wiederverwendung fertig entwickelter Funktionsmodule muss ermöglicht werden. Dies muss insbesondere auch bei einer geänderten Hardwarearchitektur möglich sein. Daraus ergibt sich, dass die Kommunikation zwischen Funktionsmodulen ortstransparent erfolgen muss, d.h. es darf bei der Funktionsmodellierung keine Unterscheidung zwischen lokalem Datenaustausch und Buskommunikation gemacht werden.
Schon bei der Entwicklung müssen zudem mögliche Varianten in der Zukunft berücksichtigt werden, da dies ein wichtiges Kriterium für die Wiederverwendbarkeit darstellt /7/. Trotz der erhöhten Variabilität darf sich weder die benötigte Prozessor-Performance noch die Codeeffizienz zum Negativen verändern. Weitere Kriterien sind z.B. die Optimierung der Buslast und des Speicherverbrauchs im Steuergerät.
Bei der Entwicklung des Fahrzeugnetzwerks einer neuen Baureihe wird nicht das gesamte Konzept komplett umgestellt. Die Entwicklungsmethodik muss es deshalb erlauben, alte Hard- und Softwarekomponenten wiederzuverwenden und an die neu entwickelten anzubinden. Dies muss sowohl netzwerkübergreifend als auch steuergeräteintern möglich sein.
Die Forderung der Wiederverwendbarkeit gilt ebenfalls für bestehende Entwicklungsverfahren. Die Entwicklungsmethodik muss abwärtskompatibel gestaltet sein. Vorhandene Entwicklungsmethoden und vor allem vorhandene Entwicklungstools müssen weiterhin verwendet werden können. Dies steigert einerseits die Akzeptanz bei den Entwicklerinnen und Entwicklern und senkt andererseits die Kosten für Schulungen mit neuen Tools.

3.3 Verteilte Entwicklung

Die Komplexität der Software erfordert neben der Wiederverwendung von Funktionsmodulen auch einen Prozess, der die verteilte Entwicklung von Funktionen ermöglicht. Es ist abzusehen, dass die Komplexität in der Zukunft soweit steigt, dass verschiedene Zulieferer sowie unterschiedliche herstellerinterne Teams an der Entwicklung der Software für ein einziges Steuergerät beteiligt sein werden, wie in Abbildung 2 verdeutlicht wird. Dies erfordert folgende Eigenschaften:

- Die Schnittstellen zwischen Softwarekomponenten müssen klar definiert sein.
- Entwicklungsartefakte vom Modell über Quelltext bis hin zum Objektcode müssen eingebunden werden können.
- Die Integration von Software unterschiedlicher Herkunft erfordert Hardware- & Software-Mechanismen für Speicherschutz, Laufzeitüberwachung und abschrankbare Interruptlatenzzeiten /8/ /9/.

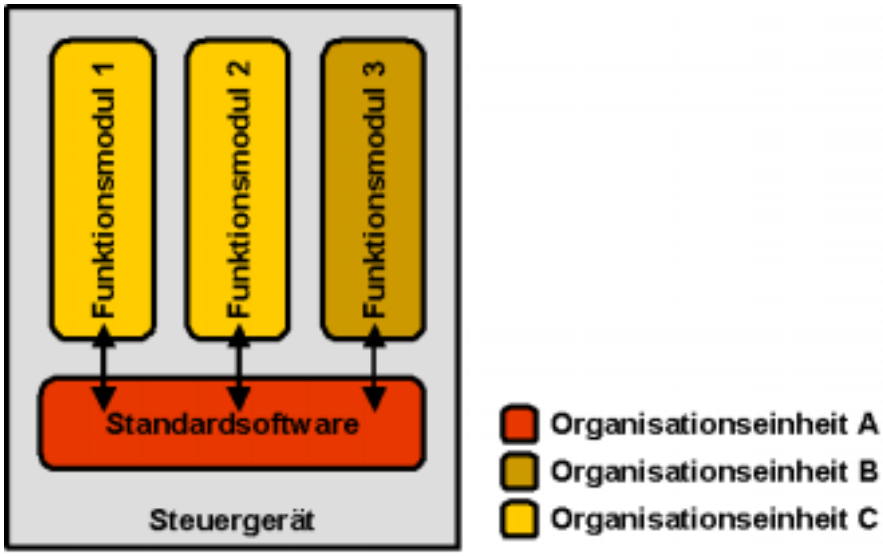

Abbildung 2: Die Spezifikation eines Steuergeräts durch den Hersteller erfolgt in verschiedenen Organisationseinheiten / The specification of a control unit takes place in different organizational units

Das System muss auf unterschiedlichen Abstraktionsebenen simuliert werden können, z.B. zur funktionalen Absicherung mit Modellierungswerkzeugen oder zur Absicherung der Kommunikationseigenschaften mit einem Netzwerksimulationstool. Dabei müssen Iterationsschleifen zurück in frühe Stadien der Entwicklung möglich sein.

Nach der Implementierung durch Software-Lieferanten müssen die Softwaremodule zudem auf ihre korrekte Implementierung hin überprüft werden können. Falls Modelle von anderen

Funktionen vorhanden sind, können sie so schon in sehr frühen Entwicklungsphasen im Modellverbund getestet werden (SIL: Software In the Loop).

3.4 Einbindung von Standardsoftwarekomponenten

Die Einbindung der Standardsoftware-Komponenten muss optimiert je nach Verteilung der Funktionen und Ausprägung der Hardware unterstützt werden. Die manuell erstellte oder automatisch generierte Software muss dabei mit Standardsoftware-Modulen wie z. B. einem OSEK/VDX-konformem /10/ Betriebssystem, einer Hardware-Abstraktionsschicht gemäß der HIS-Richtlinien /11/, /12/, /13/, Fehlerspeichermodulen oder Kommunikationstreibern für die im Fahrzeug eingesetzten Bussysteme zusammenarbeiten.
Zudem muss die Methode sowohl sich ändernde Standardsoftware-Komponenten als auch gleichartige Module unterschiedlicher Software-Hersteller (z. B. OSEK OS) zulassen. Insbesondere die Nutzung von derzeit bereits existierenden Standardsoftware-Infrastrukturen muss ermöglicht werden.

4 Analyse existierender Entwicklungsverfahren

4.1 Architekturbeschreibung als Basis einer Entwicklungsmethodik

Die Basis einer Entwicklungsmethodik für den Automobilbereich ist eine geeignete Beschreibung der funktionalen Architektur der Software im Fahrzeug. Für die Beschreibung von eingebetteten Systemen - auch von verteilten eingebetteten Systemen - sind in der Vergangenheit bereits verschiedene Notationen entwickelt worden, für die sich der subsummierende Begriff ADL (Architecture Description Language) etabliert hat. In einer ADL werden die Struktur und die funktionalen Zusammenhänge eines Systems beschrieben. Hierzu werden funktionale Einheiten, deren Schnittstellen sowie Interaktion zwischen diesen Einheiten (z.B. über Daten- und Kontrollflüsse) modelliert. Ein allgemeines Ziel ist es hierbei, durch geeignete Abstraktionsmechanismen die Unabhängigkeit und damit Wiederverwendbarkeit der funktionalen Einheiten zu erreichen und eine Systemübersicht aus verschiedenen Sichten heraus zu ermöglichen.
Als grundsätzliches Unterscheidungsmerkmal kann dabei dienen, ob diese Sprachen domänenübergreifend oder domänenspezifisch ausgelegt sind.

4.1.1 Domänenübergreifende Architekturbeschreibungen

Durch die Verwendung eines standardisierten Verfahrens scheinen sich gewisse Synergien nutzen zu lassen; vorausgesetzt, es etabliert sich domänenübergreifend (z.B. im Automotive und im Telekommunikationsbereich). Es gibt bereits einige Entwicklungen, die auf einer do-

mänenspezifischen Adaption einer generischen, erweiterbaren Modellierungssprache wie ACME /14/ oder UML /15/ basieren. Hierbei ergibt sich grundsätzlich das Dilemma, dass eine zu starke Spezialisierung der Sprache die domänenübergreifende Verständlichkeit erschwert und damit das Standardisierungsprinzip ad absurdum geführt wird, andererseits aber genau diese Modifikationen für eine Anwendung in den jeweiligen Domänen nötig scheinen.

4.1.2 Kfz-spezifische Architekturbeschreibungen

Eine andere Möglichkeit ist es, die ADL von vornherein domänenspezifisch auszurichten. Beispiele hierfür sind die TITUS-Methodik /16/ und die im Rahmen des europäischen Forschungsprojektes ITEA EAST/EEA entwickelte ADL /17/.

4.2 Kriterien für eine Architekturbeschreibungssprache

4.2.1 Allgemeine Kriterien

Grundsätzlich gibt es die folgenden allgemeinen Kriterien, die bei der Wahl der Modellierungssprache eine Rolle spielen:

- Mächtigkeit der Sprache
  Sprachmittel aus dem Bereich der Objektorientierung wie Kapselung, Polymorphismus oder Mehrfachinstanziierung haben sich bereits im allgemeinen Software-Engineering (v.a. im nicht-eingebetteten Bereich) bewährt und sind auch im Bereich der Kfz-Elektronik sinnvoll.
- Eindeutigkeit
  Bei der Anwendung der ADL sollte klar sein, durch welche Sprachmittel ein bestimmtes Problem der Domäne (z.B. die Tatsache, dass eine Fahrzeugfunktion verteilt auf zwei Steuergeräten abläuft) ausgedrückt wird. Falls die ADL hierbei mehrere Ausdrucksmöglichkeiten zulässt, stiftet dies Verwirrung.
- Vollständigkeit
  Man sollte in der Lage sein, ein System in der ADL vollständig in allen benötigten Sichten (z.B. aus Sicht der abstrakten Funktionszusammenhänge bis hin zur detaillierten Implementierung) beschreiben zu können.
- Verständlichkeit
  Eine verständliche, intuitive ADL verringert den Schulungsbedarf und erhöht deren Akzeptanz. Zum Verständnis tragen neben einer sinnvollen grafischen Notation auch ein bewusst abgregrenzter Satz an Sprachmitteln bei.

- Technische Umsetzbarkeit

  Das Ziel einer Entwicklungsmethodik ist es, das über die ADL beschriebene Modell als Grundlage für die - möglichst automatisierte - Erzeugung des Zielsystems zu verwenden. Dabei müssen die technischen Randbedingungen berücksichtigt werden, die u.U. die Verwendung gewisser Sprachmittel ausschließt.

4.2.2 Kfz-Spezifische Kriterien

Speziell im Bereich der Kfz-Komfortelektronik lassen sich die im vorigen Kapitel beschriebenen Kriterien wie folgt präzisieren:

- Mächtigkeit

  Einige Funktionen sind ähnlich oder gar identisch auf der linken und rechten Seite des Fahrzeugs (z.B. Fensterheber-Funktionen). Andere Funktionen existieren allerdings nur einmal pro Fahrzeug. Dies spricht dafür, dass sowohl eine Mehrfach- als auch eine Einfachinstanziierung einer Funktion als Sprachmittel benötigt wird. Weiterhin ist es typisch, dass bestimmte Funktionen in allen Baureihen vorkommen, allerdings in unterschiedlichen funktionalen Ausprägungen. Ein in der ADL vorgesehenes Variantenkonzept kann diese Tatsache bereits in der Architekturbeschreibung sichtbar machen.

- Vollständigkeit

  Ein nicht zu geringer Anteil der Software auf einem Steuergerät steuert dessen Verhalten in speziellen Betriebszuständen z.B. während des Hochfahrens, im Diagnosebetrieb, oder in Fehlerfällen wie z.B. Unterspannung. Diese Betriebszustände werden entweder von außen gesteuert (d.h. sie hängen direkt von der Hardware ab oder werden softwaremäßig durch Standardkomponenten erkannt) oder aber aus der Applikation heraus gesteuert. Aus Sicht der ADL bedeutet dies, dass die Schnittstellen der funktionalen Komponenten sprachlich so ausgebildet sein müssen, dass sie für die Funktionsentwicklung zum Einen eine gewisse Abstraktion vom konkreten Steuergerät ermöglichen, zum Anderen aber den Zugriff auf Steuergeräte-spezifische Firmware erlauben.

- Verständlichkeit

  Die Konzeption von Steuergerätefunktionalität findet häufig durch Blockdiagramme statt, die auch aus der Regelungstechnik vertraut sind. Die Verwendung solcher Blockdiagramme in der grafischen Notation der ADL erleichtert damit die Verständlichkeit.

- Technische Umsetzbarkeit

  Der Einsatz einer strukturierten und formalisierten Funktionsentwicklung über eine ADL darf nicht zur Folge haben, dass durch zusätzliche Module (z.B. einer komplexen Middleware-Schicht) auf dem Steuergerät unnötig viel Ressourcen verbraucht werden. So muss z.B. der Einsatz von 8-bit Mikrocontrollern mit geringer Rechenleistung und wenig Speicher möglich bleiben.

  Ein weitere Aspekt ist der eingeschränkte Freiheitsgrad bei der Realisierung der in der ADL ausgedrückten Kommunikationsmechanismen zwischen den funktionalen Komponenten. Die eingesetzten Bustechnologien wie z.B. CAN unterstützen von ihrem Charakter her eine botschafts-orientierte Broadcast-Kommunikation. Diese Charakteristik sowie die Anforderungen an das Echtzeitverhalten und Funktionssicherheit der Steuergeräte erschweren hingegen die Abwicklung eines synchronen (d.h. blockierenden) Kommunikationsprotokolls, das z.B. für einen peer-to-peer RPC (Remote Procedure Call) benötigt werden. Die "natürliche", signalorientierte Modellierung von Datenflüssen sollte deshalb in der ADL möglich sein.

4.2.3 Praktische Kriterien

Neben den technischen Kriterien für die Wahl der ADL spielen auch praktische Kriterien eine große Rolle. Dazu zählt z.B. die Möglichkeit, sich bei einer allzu komplexen ADL auf eine geeignete Untermenge zu einigen (conformance class). Ein wesentlicher Aspekt für die praktische Anwendbarkeit einer wie auch immer gearteten neuen Methodik ist die Frage, ob sie sich "stufenlos" in einem neuen Fahrzeug- oder Steuergeräteprojekt einführen lässt. Dadurch lässt sich das Risiko gegenüber einer schlagartigen Umstellung verringern.

5 Die DaVinci-Entwicklungsmethodik

Die DaVinci-Methodik wurde bei Vector mit dem Ziel entwickelt, die hohen Anforderungen an eine Entwicklungsmethodik zumindest teilweise durch eine pragmatische und realitätsnahe Vorgehensweise abzudecken. Damit soll es möglich werden, durch eine stufenlose Migration aus den aktuellen Entwicklungsprozessen die Vorteile einer methodisch gestützten Entwicklung ohne allzu große technologische Umstellungen und ohne allzu große Schulungsaufwände nutzen zu können.

5.1 Ziele

Ziel der DaVinci-Entwicklungsmethodik ist es zunächst, durch eine geeignete Beschreibung der Schnittstellen die funktionale Struktur von Steuergeräte-Anwendungssoftware erfassen

zu können. Wenn solche präzise und vollständig beschriebenen Schnittstellen vorliegen, erleichtert dies

- die Beschreibung der funktionalen Architektur eines verteilten Elektroniksystems
- die Bildung und Wiederverwendung von Software-Komponenten
- den Austausch und die Integration von Fahrzeugfunktionen
- die Integration der Anwendung mit der Basis-Software (Betriebssystem, Kommunikationsschichten) der Steuergeräte

5.2 Designelemente

In diesem Kapitel werden die wesentlichen Designelemente der DaVinci-Methodik beschrieben.

5.2.1 Signal

Ein Signal ist ein semantisch wohldefiniertes und typisiertes Datum aus der Anwendungsdomäne, das zum Austausch von Information dient. Ein Signal hat u.a. die folgenden Attribute:

- Name
- Länge (Anzahl von Bits)
- Umrechnungsformel für die physikalische Interpretation der Signalwertes
- Initialwert

Signale sind in einem globalen Kontext definiert und können von anderen Designobjekten referenziert werden. Das Signal an sich sagt noch nichts über dessen konkreten Transport (z.B. als Bussignal) aus. Dies ergibt sich erst aus der Verwendung des Signals in einem Systemkontext.

5.2.2 Software-Komponente

Eine Software-Komponente ist die Repräsentation einer Fahrzeug(teil)funktion, deren "Innenleben" durch wohldefinierte Schnittstellen gekapselt ist. Die Kommunikationsschnittstellen einer Software-Komponente bilden Signale. In der grafischen Notation werden Eingangssignale links, Ausgangssignale rechts dargestellt.

Abbildung 3: Software-Komponente mit Signalschnittstellen (Beispiel: Rücklichtsteuerung) / software component with signal interfaces (example: back light control)

Eine Software-Komponente ist alternativ realisiert über eine

- Unterstruktur

  Die Software-Komponente enthält eine hierarchische Dekomposition durch eine weitere Ebene von Software-Komponenten. Durch eine solche Unterstruktur kann spezifiziert werden, dass eine Fahrzeugfunktion potentiell verteilt auf mehreren Steuergeräten ablaufen kann. Die an der äußeren Schnittstelle spezifizierten Signale werden dabei an den Schnittstellen der Unterkomponenten wieder aufgenommen. Zur Kommunikation ausschließlich zwischen den Unterkomponenten können zusätzliche Signale dienen, die nicht an der äußeren Schnittstelle auftauchen.

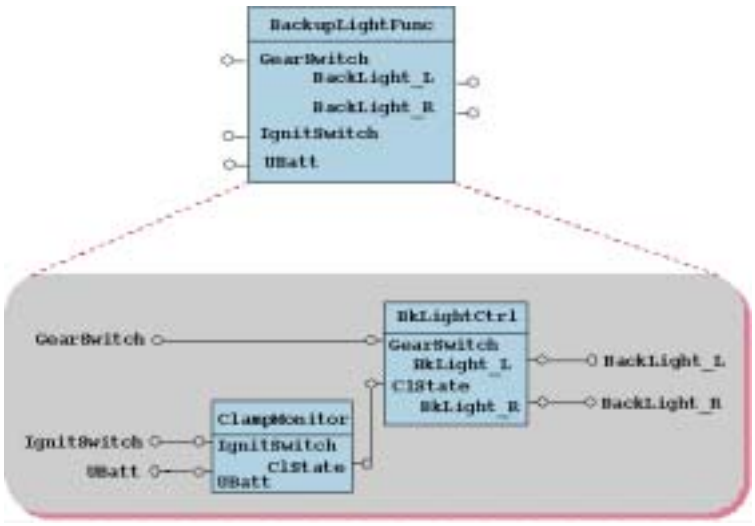

Abbildung 4: Software-Komponente mit Unterstruktur / Software component with substructure

- Verhaltensbeschreibung

  Die Software-Komponente ist nicht weiter unterteilt, sondern enthält direkt die Implementierung der Funktion. Die Implementierung kann dabei auf unterschiedliche Arten erfolgen (siehe Kapitel 0).

  Eine durch eine Verhaltensbeschreibung realisierte Software-Komponente kann neben den Signalschnittstellen noch weitere Arten von Schnittstellen enthalten. Diese sind Diagnose-Schnittstellen für den Zugriff auf interne Daten der Software-Komponente sowie die später durch die Steuergeräte-Infrastruktur bereitzustellenden Timer-Dienste.

  Weiterhin lassen sich Parameter spezifizieren, die als Vorgabe für die Ausführung der Software-Komponente auf einem Steuergerät dienen. Dazu gehören z.B. die Spezifikation einer Zykluszeit für eine zyklische Ausführung der Software-Komponente oder die Spezifikation einer bedingten Ausführung. Eine bedingte Ausführung wird spezifiziert, indem man die Software-Komponente auf generisch im Modell formulierte Zustandsmaschinen bezieht und per Definition eine Software-Komponente z.B. nur bei einem bestimmten Zustandwechsel ausführt. Die Implementierung einer solchen generischen Zustandsmaschine wird später in der Steuergeräte-Infrastruktur bereitgestellt (siehe Kapitel 0).

5.2.3 Software-System

Ein Software-System ist eine Konfiguration aus Software-Komponenten und repräsentiert die funktionale Software-Gesamtstruktur eines konkreten Fahrzeugs. Ein Software-System ist aus Sicht der Signalflüsse in sich geschlossen. Ein Eingangssignal einer verwendeten Software-Komponente wird entweder durch eine andere Software-Komponente geliefert, oder kommt "von außen" durch einen Sensor. Ein Ausgangssignal einer Software-Komponente wird von einer beliebigen Anzahl anderer Software-Komponenten verwertet und/oder als Aktuator nach außen geführt.

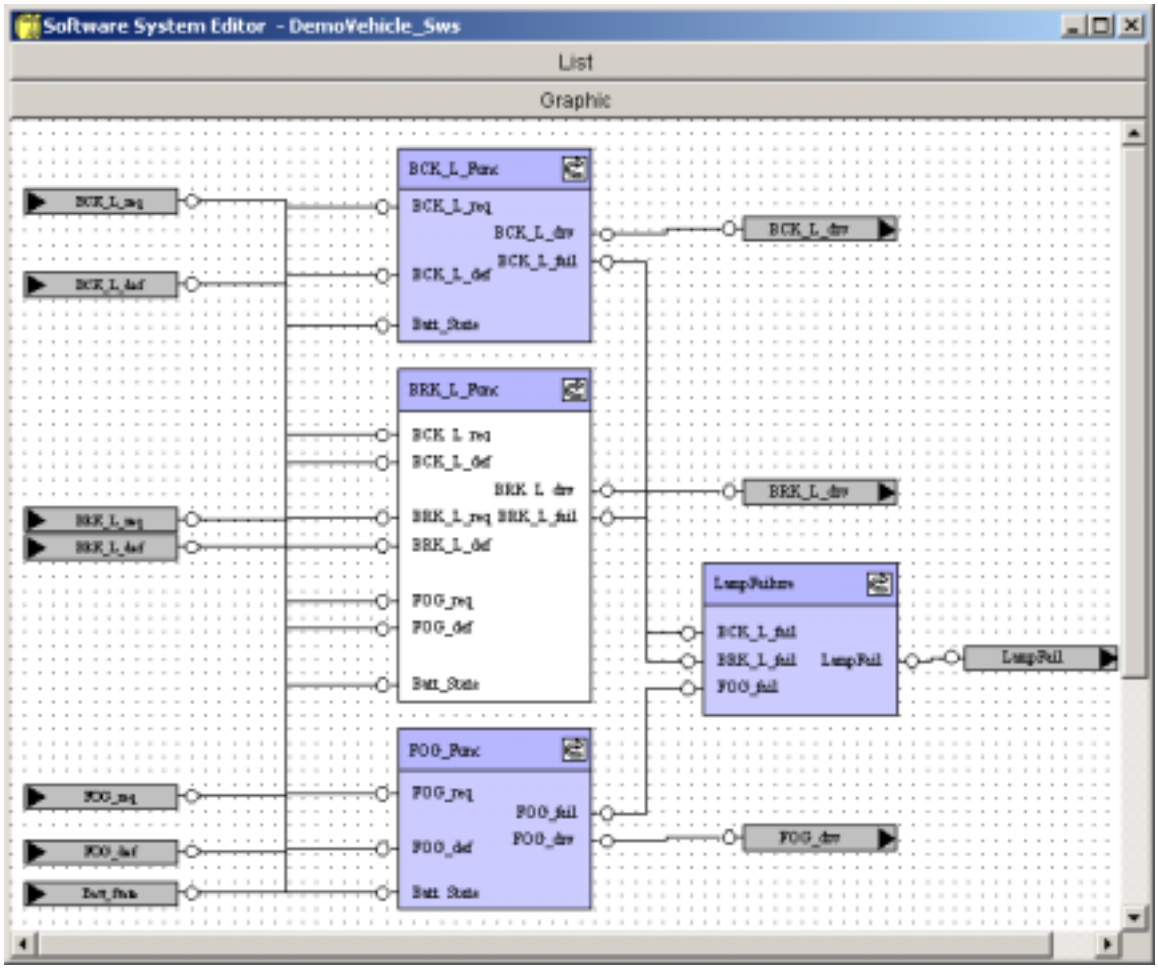

Abbildung 5: Software-System (Beispiel: Ausschnitt aus der Ansteuerung der Heckleuchten) / Software system (example: cutout from the back lights control)

5.2.4 ECU

Eine ECU beschreibt ein physikalisches Steuergerät mit seinen Schnittstellen (Sensorik und Aktuatorik), jedoch noch nicht die tatsächlich auf dem Steuergerät ablaufende Anwendungsfunktionalität. Durch die Zuordnung jeweils eines Signals zu jedem Sensor und Aktuator wird ausgedrückt, dass die ECU die Quelle bzw. Senke dieses Signals ist.

5.2.5 Hardware-System

Ein Hardware-System ist eine Konfiguration aus ECUs und den - ebenfalls modellierten - Bussen, durch die die ECUs miteinander verbunden sind. Durch ein Hardware-System wird damit die physikalischen Topologie eines konkreten Fahrzeugs oder aber eines bestimmten Testaufbaus der Steuergeräte ausgedrückt.

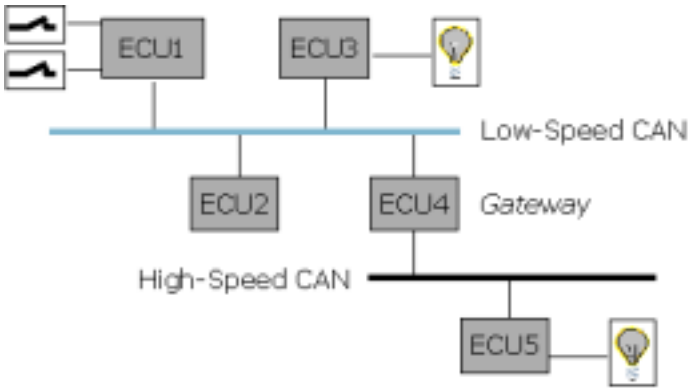

Abbildung 6: Hardware-System / Hardware system

5.2.6 Mapping-System

Ein Mapping-System ist die Integration eines Software-Systems auf einem Hardware-System. Diese Integration wird durch die folgenden Design-Schritte innerhalb des Mapping-Systems beschrieben

- Komponenten-Mapping

  Das Komponenten-Mapping beschreibt die Zuordnung der im Software-System enthaltenen Software-Komponenten auf die im Hardware-System enthaltenen ECUs. Diese Zuordnung wird dabei nur für die durch eine Verhaltensbeschreibung realisierten Software-Komponenten getroffen, also der untersten Ebene in der Hierarchie der Software-Komponenten.

  Aus dem Komponenten-Mapping und den Signalschnittstellen der Software-Komponenten ergibt sich implizit die benötigte physikalische Transportweise der zwischen den Software-Komponenten ausgetauschten Signale (Intra-ECU-Signal oder Bussignal) sowie die Sende- und Empfangsrelationen zwischen ECUs und Bussignalen.
- Kommunikations-Mapping

  Das Kommunikations-Mapping beschreibt die Zuordnung der abhängig vom Komponenten-Mapping entstandenen Bussignale zu Busbotschaften. Die Attribute dieser Botschaften wie Priorität, Sendeart und Zykluszeit bestimmen das Zeitverhalten der Buskommunikation.
- Implementierungs-Design innerhalb der ECUs

  Das Implementierungs-Design beschreibt
  - die Zuordnung der Software-Komponenten innerhalb einer ECU auf Tasks, d.h. Ausführungskontexte im Sinne eines Echtzeit-Betriebssystems

- die Ausführungsbedingungen der Tasks wie z.B. eine zyklische oder ereignisgesteuerte Aktivierung

Weiterhin wird im Rahmen des Implementierungs-Design die auf dem Steuergerät eingesetzte Basissoftware wie z.B. OSEK-Betriebssystem und Kommunikations-Stack definiert und gemäß dem auf dem Steuergerät eingesetzten Controller konfiguriert.

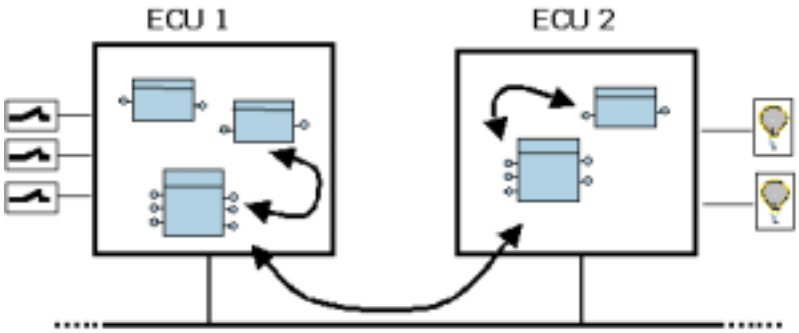

Abbildung 7: Mapping-System / Mapping system

Diese Designschritte können durch den Einsatz geeigneter Algorithmen bis zu einem gewissen Grade automatisiert werden, so dass man sehr schnell zu einem ausführbaren System kommt, das dann durch manuellen Eingriff weiter optimiert werden kann. Durch eine Reihe von Mapping-Systemen können damit Experimente mit unterschiedlichen Auslegungen einer bestimmten, gleich bleibenden Funktionalität auf einer bestimmten Hardware-Topologie durchgeführt werden.

Die Definition der Kommunikation zwischen den Steuergeräten (Kommunikationsmatrix) ist in den aktuellen Entwicklungsprozessen die Schnittstelle zwischen dem Fahrzeughersteller und den Steuergeräte-Zulieferern. Um die Migration aus den aktuellen Prozessen zu ermöglichen, kann eine solche Kommunikationsmatrix importiert werden. Die in der Kommunikationsmatrix beschriebenen Signale können dann direkt als Schnittstelle an der Software-Komponenten genutzt werden. Das oben beschriebene Kommunikations-Mapping ist hierbei ebenfalls vorgegeben. Andererseits ist es auch möglich, die in einem Mapping-System beschriebene Kommunikation als Kommunikationsmatrix zu exportieren. Damit kann wie bisher die Kommunikationsmatrix als "Master" dienen und es ist z.B. möglich, die DaVinci-Methodik mit lokaler Wirkung nur an einigen wenigen Steuergeräten oder Funktionen in einem Fahrzeugprojekt einzusetzen.

5.3 Implementierung

Durch die im vorigen Kapitel genannten Designobjekte wird in einem Modell die Struktur und die Schnittstellen von Software auf vernetzten Steuergeräten beschrieben. Gemäß der im Modell beschriebenen Schnittstellen wird nun den einzelnen Designobjekten eine Implementierung hinterlegt.

Implementierung von Software-Komponenten

Die Verhaltensbeschreibung einer Software-Komponente enthält die tatsächliche Implementierung der Funktionalität. Diese Implementierung kann alternativ auf eine der folgenden Arten erfolgen

- Direkte C-Implementierung
  Gegen eine aus dem Modell abgeleitete Programmierschnittstelle (API) wird das Verhalten der Software-Komponente programmiert. Der lesende bzw. schreibende Zugriff auf die Signale findet dabei über eine Makroschnittstelle statt.
- Modellbasierte Implementierung
  Für eine modellbasierte Implementierung wird die Software-Komponenten mit ihren Schnittstellen durch eine äquivalentes Modell gemäß der Notation des gewählten Verhaltensbeschreibungswerkzeuges abgebildet. Im Beispiel des Werkzeuges Matlab® Simulink® wird eine Software-Komponente durch ein Simulink-Subsystem abgebildet und die Eingangs- und Ausgangssignale als Ports an diesem Subsystem. Die eigentliche Implementierung erfolgt dann innerhalb dieses Modells z.B. durch einen endlichen Zustandsautomaten.
  Die Codegeneratoren dieser Verhaltensbeschreibungswerkzeuge sind dabei so angepasst, dass der von ihnen erzeugte C-Code auf die bereits erwähnte Programmierschnittstelle passt.

Die Implementierungsart der Software-Komponente ist nach außen hin völlig gekapselt. Dadurch lassen sich Software-Komponenten auf dem gleichen Steuergerät integrieren, die durch unterschiedliche Entwicklungstechniken entstanden sind.

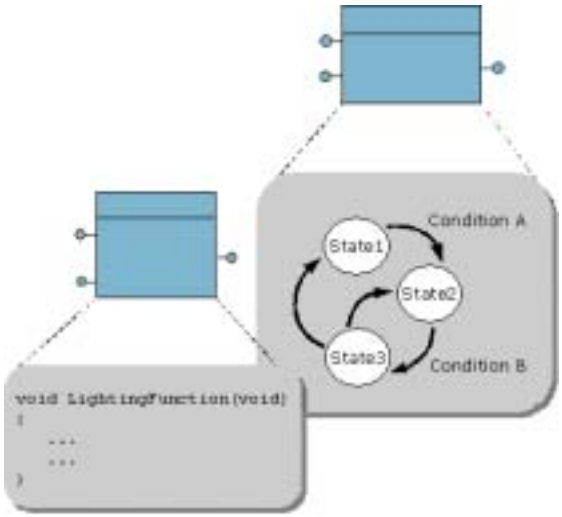

Abbildung 8: Implementierungsalternativen einer Software-Komponente / Alternative implementations of a software component

5.3.2 Implementierung von I/O-Firmware auf Steuergeräten

Im Modell ist spezifiziert, welche Signale von den Sensoren einer ECU geliefert werden bzw. welche Signale von den Aktuatoren abgenommen werden. Gegen eine aus dem Modell abgeleitete Programmierschnittstelle (API) wird nun der Zugriff auf die Sensorik/Aktuatorik über eine I/O-Firmware implementiert. Diese I/O-Firmware dient damit als Treiber für die Signale.

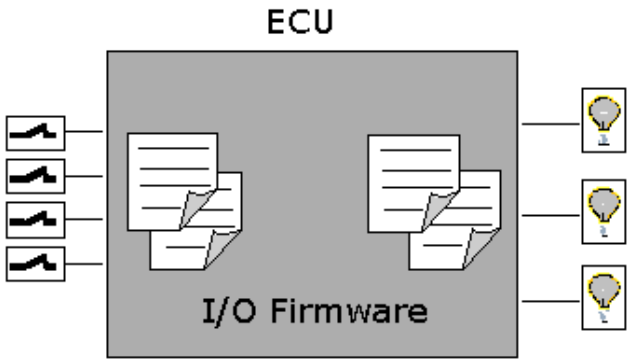

Abbildung 9: I/O Firmware auf einem Steuergerät / I/O firmware on a control unit

Weiterhin wird die Steuergeräte-spezifische Implementierung der bereits in Kapitel 0 erwähnten generischen Zustandsmaschinen beigelegt. Damit ist das Steuergerät in der Lage, beliebige Software-Komponenten auszuführen, die sich ebenfalls auf diese generischen Zustandsmaschinen beziehen.

5.4 Generierung der Zielsystems

Ein Mapping-System enthält alle benötigten Modellinformationen, um den Target-Code für die einzelnen Steuergeräte automatisch zu generieren. Hierbei laufen die folgenden Schritte ab:

- Die im Mapping-System beschriebene Buskommunikation wird verwendet, um Standard-Kommunikationskomponenten zu konfigurieren. Hierbei kommen serienerprobte Module und Generierungswerkzeuge zum Einsatz. Abhängig von der Buskommunikation werden die in Kapitel 0 erwähnten Signalmakros als Zugriffe auf die Kommunikationskomponenten oder aber als Zugriffe auf globale Variablen für die Steuergeräte-interne Kommunikation ausgeneriert.
- Das ebenfalls im Mapping-System beschriebene Implementierungsdesign der einzelnen Steuergeräte wird genutzt, um das Betriebssystem des Steuergeräts spezifisch zu konfigurieren. Die Task-Implementierungen werden ebenfalls generiert und enthalten die Aufrufe zur Ausführung der einzelnen Software-Komponenten sowie der Standard-Kommunikationsmodule.
- Das Betriebssystem, der Kommunikationsstack und die im Modell hinterlegten Implementierungen der Software-Komponenten und der Steuergeräte-Firmware werde schließlich gemeinsam übersetzt und gelinkt.

Durch diese automatisch ablaufende Codeintegration sind keine manuellen Aufwände mehr nötig, um die Applikation mit dem Betriebssystem und den Kommunikationskomponenten zu integrieren.

5.5 Prozessbeispiel: Entwicklung einer Software-Komponente

Die wohldefinierten Schnittstellen der Software-Komponenten bilden die Basis für verteilte Entwicklungsprozesse, bei denen neben dem Fahrzeughersteller viele Steuergerätezulieferer aber auch reine Softwarezulieferer mitwirken. In Abbildung ist ein beispielhafter Prozess für die iterative Entwicklung einer Software-Komponente dargestellt. Eine bestimmte Funktionalität wird dabei durch einen Zulieferer entwickelt, der diese Funktionalität als sein schützenswertes Know-How erachtet. Um Fehler bei der Integration mit den anderen Funktionen frühzeitig zu erkennen, möchte der Hersteller z.B. Integrationstests auf dem PC und auf einem Muster-Steuergerät durchführen.

Im vorgestellten Prozess wird von den folgenden Annahmen ausgegangen:

- Die Schnittstellen der Software-Komponenten sind öffentlich. Das eigentliche schützenswerte Know-How ist die Implementierung der Software-Komponenten.
- Object-Code bietet ausreichenden Know-How-Schutz

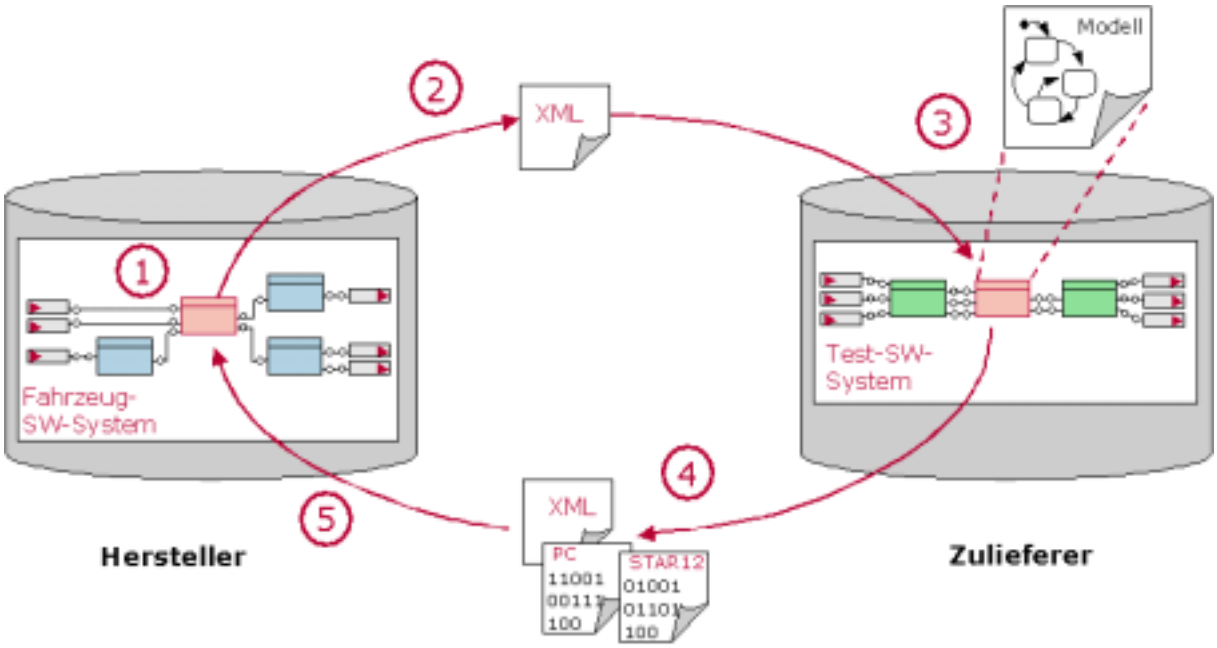

Abbildung 10: Know-How-geschützter Datenaustausch / Know how protected data exchange

Die folgenden Schritte werden dabei durchlaufen

- Schritt 1: Der Hersteller definiert die funktionale Architektur des Fahrzeugs. Als Ergebnis entstehen "leere" Software-Komponenten mit wohldefinierten Schnittstellen und zusätzlichen Spezifikationsdokumenten.
- Schritt 2: Der Hersteller exportiert die zu entwickelnde Software-Komponente in ein XML-Austauschformat.
- Schritt 3: Der Zulieferer importiert die Software-Komponente und bettet sie in ein Testsystem zur Entwicklung der Funktionalität ein. Anschließend entwickelt der Zulieferer die Funktionalität der Software-Komponente.
- Schritt 4: Der Zulieferer exportiert die Software-Komponente für den Hersteller. Der Zulieferer kann dabei entscheiden, ob er die vollständige Software-Komponente - inklusive Implementierung (d.h. Modell oder C-Code) exportiert, oder nur Object-Code für bestimmte, zwischen Hersteller und Zulieferer vereinbarter Targets.
- Schritt 5: Der Hersteller importiert die Software-Komponente und kann den ebenfalls importierten Object-Code für z.B. eine Funktionsintegration nutzen, hat aber keinen Zugriff auf die Implementierung der Software-Komponente.

Dieser Datenaustausch findet unter der Kontrolle eines Versionsmanagements statt. Die Historie der Software-Komponente sowie die Änderungen zwischen den einzelnen Versionen der Software-Komponente werden dabei dokumentiert und sind reproduzierbar. Dadurch lässt sich der obige Prozess bei Bedarf iterativ mehrmals durchlaufen und ermöglicht funktionale Tests und Integrationstests bereits in frühen Entwicklungsphasen.

5.6 Charakterisierung der DaVinci-Methodik

Als Abgrenzung zu anderen Entwicklungsmethoden kann die DaVinci-Methodik wie folgt charakterisiert werden:

- Die Methodik ist geeignet zur Beschreibung von geschlossenen (d.h. zum Entwicklungszeitpunkt vollständig bekannten), verteilten Systemen speziell im Bereich der Komfortelektronik im Fahrzeuginnenraum.
- Software-Komponenten dienen sowohl zur Strukturierung als auch zur Implementierung von Funktionen.
- Das detaillierte Software-Design findet innerhalb einer Software-Komponente statt.
- Zwischen den Software-Komponenten wird lediglich der Datenfluss, aber kein Kontrollfluss modelliert.
- Die Software-Komponenten werden direkt implementiert, d.h. es gibt keine Beschreibung des Verhaltens mit abstrakt formulierten Daten bzw. Algorithmen.
- Die Wiederverwendbarkeit von Software wird durch Mehrfachverwendung von Komponenten über verschiedene Konfigurationen hinweg erreicht, nicht aber durch Mehrfachinstanziierung oder Abstraktion.
- Die grafische Notation ist im Wesentlichen ein Blockdiagramm (block-and-line).
- Alle Signalschnittstellen der Software-Komponenten sind bis hin zur obersten Strukturierungsebene sichtbar, d.h. es gibt keine "versteckten" Schnittstellen in der Unterstruktur. Damit kann eine Software-Komponente als Spezifikationsmedium mit vordefinierten Schnittstellen dienen.

Bezogen auf das V-Modell in Kapitel 0 lässt sich DaVinci bei den Prozessschritten auf der "linken Seite des V" einsetzen. Gut abgedeckt sind die Anforderungen bei Prozessschritt "Systemarchitekturdesign". Im Prozessschritt "Realisierung Subsystem" lässt sich DaVinci als Integrationsplattform für die Softwareentwicklung einsetzen. In den Prozessschritten "Systembeschreibung" und "Systemintegrationsplanung" kann das in DaVinci beschriebene Strukturmodell des Systems als Schnittstelle zu weiteren Teilprozessen wie dem Requirements Engineering dienen.

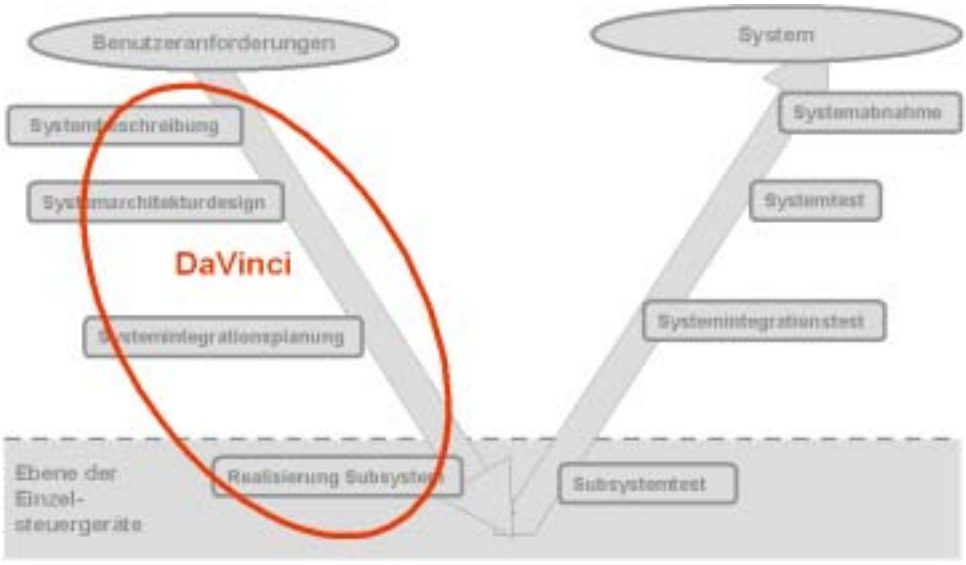

Abbildung 11: Einsatzmöglichkeiten von DaVinci / Application potentials of DaVinci

6. Zusammenfassung

Der Einsatz einer Entwicklungsmethodik mit einer strukturierten, formalen Beschreibung der Funktionsarchitektur bringt wesentliche Vorteile für die Entwicklung von zunehmend komplexer werdenden Elektroniksystemen im Kraftfahrzeug. Insbesondere wird die Wiederverwendung von Funktionen und die Durchführung von verteilten Entwicklungsprojekten erleichtert.

Die DaVinci-Methodik bietet die Möglichkeit, diese Vorteile bei der Entwicklung von vernetzten Elektroniksystemen nutzen zu können. Software-Komponenten mit wohldefinierten Signalschnittstellen dienen zur Beschreibung des verteilten Gesamtsystems und zur Software-Integration auf Steuergeräten. Ein mittels DaVinci entstandenes Architekturmodell kann damit als Basis für die Funktionsentwicklung und das Testen dienen.

Durch Integration und Anbindung der aktuell eingesetzten Technologien und Verfahren wird eine schrittweise Migration aus den aktuellen Prozessen ermöglicht. DaVinci ist damit ein ergänzender Schritt in Richtung eines durchgängigen, modellgestützten Entwicklungsprozesses.

Quellen und Hinweise

/1/ Mercer Management Consulting und Hypovereinsbank: Studie, Automobiltechnologie 2010. München, August 2001.

/2/ Bundesministerium des Inneren: Entwicklungsstandard für IT-Systeme des Bundes. Vorgehensmodell. Kurzbeschreibung. Bonn, 1997.

/3/ DIN – Deutsches Institut für Normung e. V.: ISO/DIS 11898-1-4: Road vehicles - Controller area network (CAN).

/4/ LIN Cosortium: LIN Specification Package Revision 1.3. http://www. lin-subbus.org/.

/5/ MOST Cooperation: MOST Specification Rev., 2.2. http://www.mostnet. de/downloads/Specifications/.

/6/ The Flexray Consortium, http://www.flexray.com/.

/7/ P. Clemens, L. Northop: Software Product Lines - Practices and Patterns. Addison-Wesley, Boston, 2002.

/8/ OSEK/VDX: OSEK/VDX – Time-Triggered Operation System Specification Document, Version 1.0, Juli 2003. http://www.osek-vdx.org/.

/9/ Hersteller-Initiative Software: Extensions of OSEK OS for Protected Applications. Juni 2003.

/10/ OSEK/VDX, http://www.osek-vdx.org/.

/11/ Hersteller-Initiative Software, http://www.automotive-his.de/.

/12/ Hersteller-Initiative Software: API I/O Driver, Version 2.1. Juni 2003.

/13/ Hersteller-Initiative Software : API I/O Library, Version 2.0. Februar 2003.

/14/ Acme: Architecture Description Interchange Language, Carnegie Mellon University, http://www-2.cs.cmu.edu/~acme/.

/15/ UML: Unified Modeling Language, http://www.omg.org/uml/.

/16/ U. Freund et. al.: Interface Based Design of Distributed Embedded Automotive Software - The TITUS Approach, VDI-Berichte (1547), 2000.

/17/ EAST- EEA: Embedded Electronic Architecture. http://www.east-eea.net/.