SW-basierte Integration von neuen Fahrzeugfunktionen in zentralisierten Controllern

Klaus Becker[1], Christian Buckl[1], Alexander Camek[1], Rainer Falk[2], Ludger Fiege[2], Jürgen Geßner[2], Stephan Sommer[1]
1) fortiss gemeinnützige GmbH, Guerickestraße 25, 80805 München
2) Siemens AG, Corporate Research and Technologies,
Otto-Hahn-Ring 6, 81730 München

Abstract: In aktuellen Fahrzeugen wird ein wesentlicher Teil der Fahrzeugfunktionen durch Software realisiert. Die Integration aktiv eingreifender Assistenzsysteme wird diesen Trend noch verstärken und die Komplexität des Bordnetzes wird weiter zunehmen. In diesem Artikel stellen wir einen Ansatz vor, Bordnetz und Software im Fahrzeug über eine datenzentrische Middleware zu entkoppeln. Sie koordiniert die Kommunikation zwischen Funktionen, Sensorik und Aktorik zur Laufzeit. Basierende auf einer Zentralrechnerarchitektur wird eine redundante Datenbasis zur Verfügung gestellt, die eine fail-operational Ausführung auch von sicherheitskritischen Funktionen erlaubt. Wir stellen am Beispiel Plug-and-Play (PnP) eine neue Sekundärfunktion vor, die durch diesen Ansatz ermöglicht wird. Safety- und Security-Aspekte der vorgestellten Architektur werden ebenfalls betrachtet.

1. Einführung und Motivation

Heutige Automobile sind ein komplexes Zusammenspiel von verteilen, gekoppelten Einzelsystem. Derzeitige Premiumfahrzeuge enthalten eine hohe Anzahl von bis zu ca. 100 Steuergeräten (Electronic Control Units, ECUs), die mit verschiedenen Bussystemen miteinander verbunden sind. Selbst wenn Funktionen in *Domänenkontrollern* konsolidiert werden, so ist der entstehende Verkabelungsaufwand jedoch enorm. Assistenz- und Komfortsysteme, die in den Antriebsstrang eingreifen, machen die Integration im Hinblick auf die funktionale Sicherheit sehr aufwändig.

Ein weiterer Treiber der Domänen übergreifenden Integration sind hochintegrierte Radnaben- oder radnahe Elektromotoren. Die Integration von Lenkung, Vortrieb, Verzögerung wie auch von ABS und ASR in ein einziges, autark arbeitendes Modul, lässt Differenzial, Torque-Vectoring sowie die Beschleunigung selber zu einer verteilten Steuerung werden, in der die Funktionen der Module zentral koordiniert werden müssen. Als Antwort sind in Avionik, Industrieautomatisierung und Robotik zentralisierte Architekturen entworfen worden, die auch bereits in automobilen Forschungsprojekten angewendet wurden (vgl. [SP02] und [HA11]). Dies wäre eine technische Konsolidierung wie sie zum Beispiel mit dem Wechsel von diskreten Datenleitungen hin zu Bussystemen bereits erfolgte (vgl. Abbildung 1).

In diesem Artikel stellen wir eine datenzentrische Middleware vor, die die Entwicklung neuer Funktionen und die Ankopplung der Sensorik/Aktorik über ein Backbone-

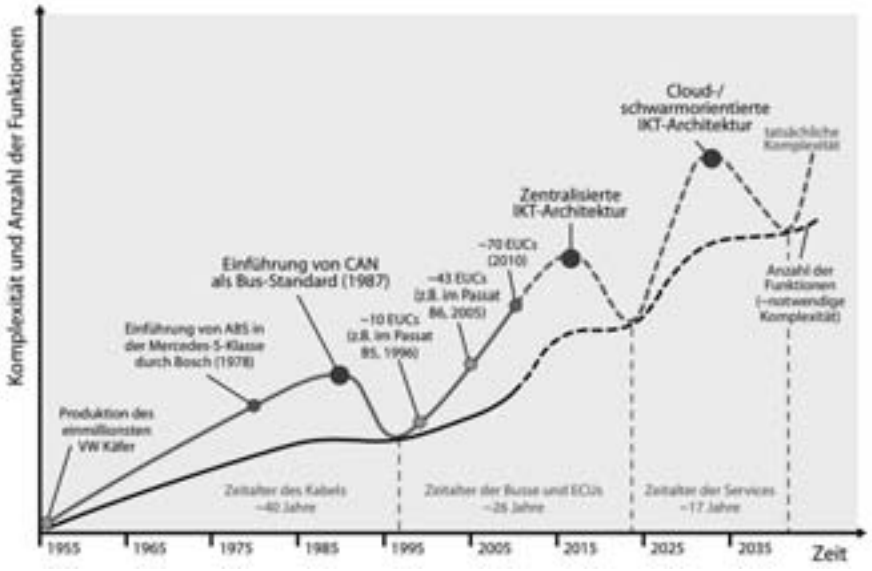

Abbildung 1: Wachsende Komplexität durch inkrementelle Weiterentwicklung [Fo12]

Netzwerk entkoppelt (vgl. [Bu12], [Br11]). Basierend auf vorverarbeitender, intelligenter Peripherie, werden Daten in aufbereiteter Form in einer Datenbank abgelegt. Die Daten werden allen Steuergeräte/SW-Funktionen zur Verfügung gestellt, die dafür konfiguriert wurden. Als wesentlicher Treiber wird die Integration neuer Funktionen auch nach Auslieferung des Fahrzeugs betrachtet, um damit Plug-and-Play Szenarien zu unterstützen. Die Middleware muss dabei die Ressourcen-Zuweisung und Safety-bzw. Security-Anforderungen umsetzen. Die Validierung der Konzepte wird im Rahmen des BMWi Förderprojektes „Robust and Reliable Automotive Computing Environment for Future eCars“ (RACE) erfolgen [RA12]. Anhand von zwei Versuchsfahrzeugen werden die technischen Grundlagen umgesetzt und evaluiert werden.

Dieser Artikel ist wie folgt strukturiert: Kapitel 2 stellt zunächst die zugrunde liegende zentralisierte Systemarchitektur vor und vergleicht diese mit existierenden Ansätzen. In Kapitel 3 wird der datenzentrische Ansatz vorgestellt. Wichtige Systemaspekte, wie Plug-and-Play, Safety und Security, werden in Kapitel 4 beschrieben. Kapitel 5 schließt mit einer Zusammenfassung.

2. Zentralisierte Systemarchitektur

Ein Kern der Systemarchitektur, die sich hier an Avionik und Industrieautomatisierung orientiert, ist eine deutlich horizontale Schichtung der Architektur. Die heute vorherrschende Bündelung von Funktion, ECU, Sensorik in Teilsystemen wird ersetzt von einer Aufteilung in Feldebene zur Anbindung von Sensorik/Aktorik, lokalen Regelung von

Stellgrößen (Motoren) und koordinierenden bzw. planenden Aufgaben in der Steuerungs- bzw. Leitebene.

Die betrachtete Systemarchitektur umfasst zentrale Rechner für koordinierende Aufgaben, ein Ethernet-basiertes Backbone und intelligente Sensoren/Aktoren, die direkt oder via Sektor-Gateways (z.B. für den Frontbereich) angeschlossen sind. Grundsätzliches Merkmal der zentralen Rechner ist in allen Fällen, dass sie keine funktionsspezifischen Ein-/Ausgaben (ADC, spezielle Busse, etc.) besitzen, sondern lediglich an das Backbone-Netzwerk angeschlossen sind.

Die Zentralrechner übernehmen das vorausschauende Planen und setzen zusätzlich modulübergreifende, überlagerte Regelungen und koordinierende Funktionen um [Sp02]. Sie bilden beispielsweise den Fahrvektor aus den Vorgaben des Fahrers und der Assistenzfunktionen und setzen diese in Leistungsanforderungen (Zielwerte) an die Motoren (Aktoren) um. Die Radmodule bestehen aus E-Motoren, Inverter und lokaler Regelung, die schnelle, geschlossene Regelungsaufgaben übernimmt und die Zielwerte von den Zentralrechnern erhält. In Zukunft können die Radmodule auch Grundfunktionen der Blockier- und Schlupfregelung übernehmen, jedoch enthalten diese Module prinzipiell keine koordinierenden Aufgaben zwischen anderen Aktoren bzw. Aufgaben, die nichts mit ihrer lokalen Kontrolle physikalischer Größen zu tun haben.

Die hohe Integration eines Elektroradmoduls lässt sich hervorragend mit einer X-by-Wire Architektur kombinieren. Eine beispielhafte Darstellung für ein X-by-Wire-Fahrzeug ist in Abbildung 2 dargestellt.

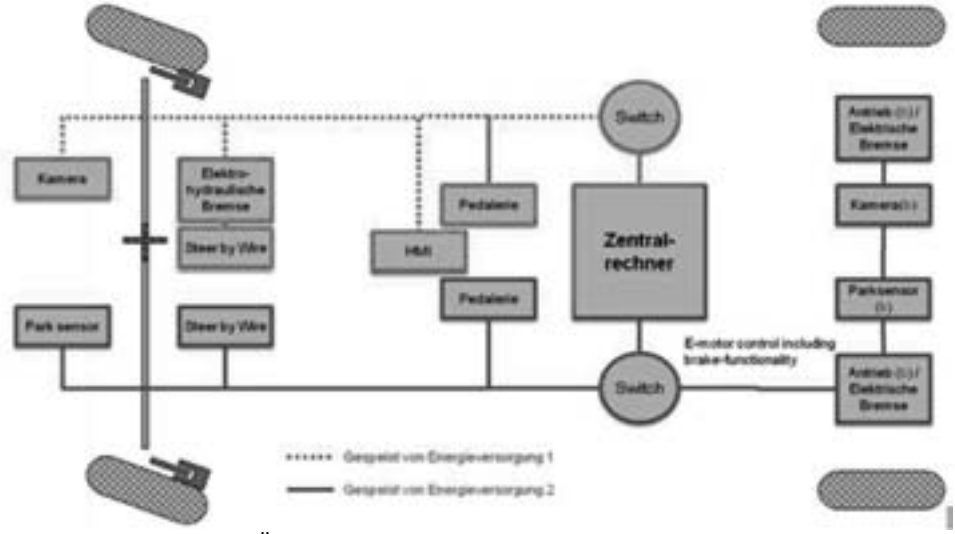

Abbildung 2: Übersicht Systemarchitektur mit logisch zentralem Rechner

Da es für ein fahrendes Automobil keinen sicheren Zustand gibt, müssen sicherheitskritische Funktionen, wie z.B. Lenkung und Bremse, ASIL-D Anforderungen erfüllen. Dies betrifft sowohl die Kommunikation als auch die Zentralrechner und die Energieversorgung. Gemäß der ISO 26262 Anforderungen muss die Wahrscheinlichkeit eines fehlerfreien Betriebs bei Einzelfehlern bei >99% bzw. für Mehrfachfehler bei > 90% liegen und das bei einer Fehlerrate von weniger als 10^{-8} /h.

Es gibt mehrere Architekturkonzepte, die ASIL-D Anforderungen mit Fail-Operational-Verhalten erreichen: Wir wählen einen Duo-Duplex-Aufbau. Jeweils ein Paar zentraler Rechner ist in einem Duplex Control Computer (DCC) verbunden, um die Ein/Ausgangsdaten gegenseitig zu überwachen und Inkonsistenzen festzustellen. Ein zweites Paar (zweiter DCC) läuft als hot-standby, um im Fehlerfall die Funktionalität aufrecht zu erhalten. Der Duo-Duplex-Ansatz lässt sich im Vergleich zu Triplex-Systemen leichter skalieren und die Aktoren benötigen kein systemisches Wissen und keine an den Grad der Redundanz gebundenen „Voter“ zur Fehlererkennung.

Das Netzwerk muss ebenfalls redundant aufgebaut werden. Hier kommt eine Ringtopologie basierend auf Ethernet zum Einsatz; für Details sei auf [Ar12] verwiesen.

Aus den Verantwortlichkeiten der Sensoren/Aktoren und der Zentralrechner (DCC) ergibt sich, dass Zustands- und Sensordaten über das Netzwerk den Applikationen in den DCCs zur Verfügung gestellt werden müssen. Umgekehrt müssen die errechneten Zielwerte den Aktoren übermittelt werden. Als Zykluszeit für den regelmäßigen Austausch dieser Daten ergibt sich ein Richtwert aus den Erkenntnissen der Projekte Sparc und HAVEit. Dort waren zentralisierte, koordinierende Funktionen mit einer Zykluszeit im Bereich von 10-20ms für betrachtete Fahrsituationen eines PKW ausreichend.

2.1. Relation zu AUTOSAR

Die hier vorgestellten Konzepte sind kein Ersatz für AUTOSAR, sondern konzentrieren sich auf die Kombination von Erweiterbarkeit (zur Laufzeit) und Safety Mechanismen, die die Entwicklung von Applikations- und Sicherungsfunktionalität trennen. Die Konzepte sind zunächst orthogonal zu AUTOSAR und könnten daher zukünftig als eine mögliche Erweiterung angesehen werden.

Die vorgestellte Architektur schreibt systemische Redundanzen vor, die so im AUTOSAR Standard nicht gefordert werden. Dies erlaubt zum einen Sicherungsmechanismen zu entwerfen, die HW-Fehler unabhängig von den Applikationen erkennen und behandeln. Dies beinhaltet auch den mixed-criticality Betrieb, in dem Anwendungen unterschiedlicher Safety-Levels auf einer DCC laufen und kommunizieren können. Des Weiteren wird die datenzentrische Middleware oberhalb der AUTOSAR RTE-Funktionalität Datenfusion und -bereitstellung anbieten, die nicht statisch konfiguriert sondern zur Laufzeit geändert werden kann (s. Abschnitt 4.1). Im Rahmen der laufenden Arbeiten vermeiden wir zunächst den Aufwand, AUTOSAR in einer mixed-criticality Umgebung einzusetzen bzw. mit einer dynamischen Middleware zu verbinden.

3. Datenzentrische Middleware

Die datenzentrische Middleware muss also die Elemente der Systemarchitektur (Sensoren, Aktoren, DCCs) verbinden und dabei die folgenden Ziele umsetzen: Echtzeitfähigkeit (Zykluszeit 10ms), Erweiterbarkeit, Mixed-criticality (Kombination QM bis ASIL-D, fail-operational), sowie Security. Im Gegensatz zu den aktuell verfolgten Ansätzen, bei denen von verschiedenen Steuergeräten Nachrichten verarbeitet werden, die auf dem Bus anliegen, wird bei dem hier vorgestellten Ansatz mit Daten gearbeitet, die ein Abbild des aktuellen Fahrzeugzustands wiederspiegeln.

3.1. Übersicht der datenzentrischen Architektur

Zur Entkoppelung der Anwendungen von der Infrastruktur (Netzwerk, ECU) werden in der datenzentrischen Architektur vorab verschiedene Datentypen (so genannte Topics) definiert. Beispiele für Topics sind die Fahrzeuggeschwindigkeit, Radgeschwindigkeit und Außentemperatur. Softwarekomponenten kommunizieren miteinander ausschließlich über an Topics gebundene Daten, indem sie diese an die Middleware weitergeben bzw. diese von der Middleware erhalten. Neben den Topics kann es notwendig sein, die Anforderungen an die Daten noch näher zu beschreiben. Dazu werden Meta-Daten verwendet. Diese beschreiben sowohl Anforderungen an die Daten als auch Garantien. Beispiele für Metadaten sind Ort der Erzeugung, Genauigkeit, Konfidenz und Sicherheitslevel.

Die Ausführung der Anwendungen erfolgt zyklenbasiert (Zykluslänge z.B. 10ms). Dabei werden die Daten zum Beginn des Zyklus von der Middleware in den Arbeitsbereich der in diesem Zyklus startenden SW-Komponenten kopiert und bei Beendigung der Funktion aus logischer Sicht (siehe Ausführungszeiten [He01]) wieder in die Middleware zurückkopiert. Ein Zugriff der Funktion auf weitere Daten während der Abarbeitungszeit ist nicht möglich. Neben volatilen Daten, also Daten, die in jedem Zyklus neu erzeugt werden, sollten auch persistente Daten unterstützt werden. Persistente Daten behalten dabei solange ihren Wert, bis ein neuer Wert vorliegt.

Um die Verschaltung der Software-Komponenten zu ermitteln geben diese an, welche Daten sie benötigen und welche sie bereitstellen. Eigenschaften wie Latenz und Jitter werden dabei durch Quality of Service (QoS) Parameter beschrieben und fließen in die Berechnung mit ein. Auf Basis dieser Informationen wird schließlich die Verschaltung der Software-Komponenten berechnet. Als Basis der Umsetzung in RACE wird die Middleware Chromosome [CH12] verwendet und im Zuge des Projektes erweitert.

3.2. Chromosome (XME)

Die Chromosome-Middleware (siehe Abbildung 3) besteht aus mehreren, modular gestalteten Komponenten. Eine Auswahl dieser Komponenten wird in diesem Abschnitt betrachtet.

Die Hardware-Basis stellen die in Kapitel 2 vorgestellten, gedoppelten Rechner eines DCCs dar. Zur Unterstützung von Anwendungen verschiedener Kritikalitäten wird in RACE ein Betriebssystem mit Virtualisierungsunterstützung verwendet, die Portierbarkeit auf neue Plattformen wird durch einen Hardware Abstraction Layer (HAL) erleichtert. Dieser kapselt den Zugriff auf die Hardware bzw. das Betriebssystem und setzt die Arbitrierung beim gleichzeitigen Zugriff auf Ressourcen um.

Die eigentlichen Middleware-Funktionen werden dann im oberen Teil (in der Abbildung hellblau gekennzeichnet) umgesetzt. Die Kommunikation ist im Interface-Manager gekapselt, der, je nach Anforderung der Anwendung, mit Plug-Ins für verschiedene Kommunikationsstandards versehen werden kann. In Standardanwendungen sind dies häufig TCP und UDP, in RACE wird Ethernet (Layer 2) für die sicherheitskritische Kommunikation verwendet. Empfangene Daten werden in der Middleware (Datenbank, siehe nächster Abschnitt) zwischengespeichert, vom I/O Management validiert und anschließend über den Broker an die einzelnen, lokalen Empfänger verteilt. Dies erlaubt eine vollständige Entkoppelung von Sender (Daten-Produzent) und Empfänger (Daten-Konsument). Die Verschaltung der Komponenten wird dabei vom Directory berechnet und als Konfiguration an die Broker verteilt. Die jeweilige Ausführung der Komponenten (sowohl der Middleware als auch der Anwendung) wird vom Execution Manager angestoßen und überwacht.

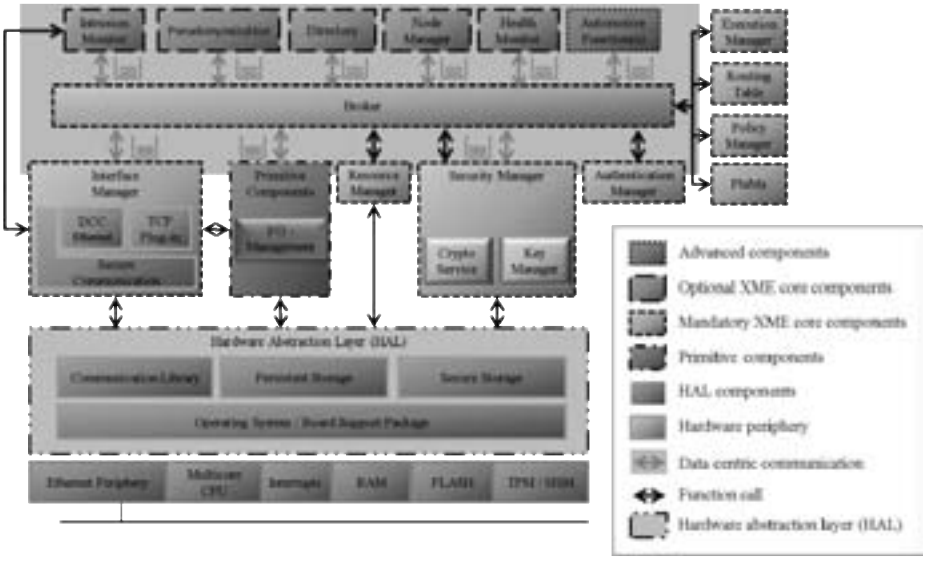

Abbildung 3: Chromosome Stack-Übersicht

Um die Echtzeitfähigkeit des Systems zu garantieren, wird die Ausführung der Funktionen offline geplant und online überwacht. Basierend auf Informationen unter anderem zur Worst-Case Execution Time (WCET) können dabei die Komponenten optimal eingeplant werden. Ebenfalls können dadurch Degradationsstrategien festgelegt und hinterlegt werden. Die offline berechnete Allokation der Ressourcen und die Ausführungspläne für die verschiedenen Betriebszustände (incl. Degradationsstufen) sind im Plattform-Management hinterlegt. Zur Laufzeit wird durch das Plattform-Management der aktuelle Ausführungsplan im Execution Manager hinterlegt. Sobald ein Fehler erkannt wird, kann durch das Plattform-Management eine geeignete Alternativkonfiguration im Execution Manager hinterlegt werden. Für nicht sicherheitskritische Anwendungen ist die Ressourcenplanung für neue, vorher nicht bekannte Anwendungen zur Laufzeit möglich. Die Separation von den sicherheits-kritischen Funktionen wird dabei durch Virtualisierung des zugrunde-liegenden Systems und der dadurch erreichten Separation sichergestellt. Dies ermöglicht eine wesentlich höhere Flexibilität und somit das nachträgliche einbringen neuer Funktionen in das System (z.B. Plug-and-Play).

3.3. Logisch zentrale Datenbasis

Um den Anforderungen sicherheitskritischer Anwendungen gerecht zu werden, müssen Daten vor der Verarbeitung in den einzelnen Anwendungen auf Plausibilität geprüft werden. Als zentraler Punkt in XME dient hierzu eine logische, zentrale Datenbank als Datenbasis. Alle von XME bzw. den Anwendungen auf einem Knoten (DCC) verarbeiteten Daten werden zunächst in der logischen Datenbank zwischengespeichert. Zu Beginn eines Zyklus werden die Daten der Anwendung zur Verfügung gestellt. Die Ergebnisse der Anwendung werden nach der Berechnung zum Ende eines Zyklus wieder in die Datenbank geschrieben.

Bei sicherheitskritischen Anwendungen können die Berechnungen einer Anwendung dupliziert auf mehreren Knoten (DCCs) im Netzwerk ausgeführt werden. Dabei werden sowohl die Eingabedaten als auch die Ergebnisse der Berechnungen basierend auf den Ergebnissen in der Datenbank durch das I/O Management verglichen und bewertet. Neben den Aufgaben im Regelbetrieb ist die Datenbank ebenfalls von zentraler Bedeutung während der Entwicklung und der Systemtests. Dabei werden über die Datenbank synthetische Daten eingespeist bzw. Ergebnisse ausgelesen. Dadurch können künstlich Fehler provoziert bzw. ins System eingespeist und die Reaktion des Systems darauf getestet und aufgezeichnet werden.

3.4. Kontrollfluss

Wie bereits in Abschnitt 3.1 eingeführt, erfolgt die Ausführung von Funktionen in der RACE-Architektur Zyklus-basiert. Je nach Berechnungsaufwand können Anwendungen mindestens einen oder auch mehrere Zyklen zur Berechnung benötigen. Dabei werden der Anwendung die Daten zu Beginn des ersten Zyklus' von der Datenbank zur Verfügung gestellt und am Ende der Berechnung wieder zurück an die Datenbank

übertragen. Nach einer optionalen Konsistenzprüfung stehen die Daten dann im nächsten Zyklus für weitere Komponenten zur Verfügung.

Eine beispielhafte Anwendung ist in Abbildung 4 dargestellt. Weder die Sensoren noch die Aktoren sind dediziert für die Funktion Notbremsassistent installiert, noch direkt mit einem dedizierten Steuergerät verbunden, das exakt diese Funktion abbildet. Alle beteiligten Softwarekomponenten werden auf einem Zentralrechner (DCC) ausgeführt.

Zu Beginn werden die Messwerte der Sensoren zuerst über eine Komponente zur Datenfusion vorverarbeitet und in Zusammenhang gesetzt. Die Ergebnisse, wie z. B. Objektlisten, werden dann an die Software-Komponente des Notbremsassistenten übertragen. Im Arbitrierer werden die Ergebnisse des Notbremsassistenten mit den Eingaben des Fahrers abgeglichen und die Entscheidung über die Fahrstrategie berechnet. Über die Softwarekomponente des Antriebsstrangs werden im Anschluss die einzelnen Stellgrößen für die Aktoren ermittelt und an diese übertragen. In dem gezeigten Anwendungsfall sind die Aktoren zugleich Sensoren. Sie kennen den eigenen Zustand und ermöglichen so der Softwarekomponente für den Antriebsstrang einen Soll / Ist Abgleich.

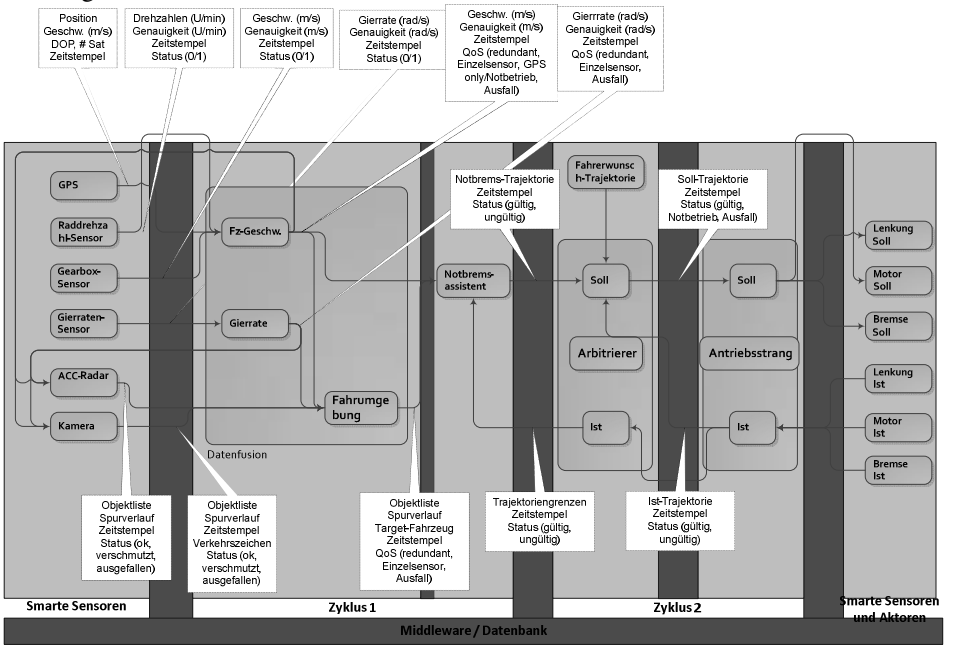

Abbildung 4: Funktionsblöcke der Anwendung Notbremsassistent

Durch die Entkoppelung der einzelnen Funktionen und die Verfügbarkeit der Informationen im System wird im Gegensatz zur Realisierung in einem dedizierten Steuergerät die Möglichkeit geschaffen, die Informationen für weitere Funktionen zu nutzen. Dies ermöglicht es den Herstellern z. B. bei bereits ausgelieferten Fahrzeugen Sicherheitsfunktionen nachzurüsten, da nur eine weitere Software-Komponente

installiert werden muss. Für nicht sicherheitskritische Funktionen ist sogar eine Installation von neuen Anwendungen durch den Benutzer während der Fahrt möglich.

4. Im Fokus stehende Systemaspekte

In diesem Kapitel beschreiben wir wichtigen Sekundärfunktionen, die mit Hilfe der vorgestellten Middleware umgesetzt werden.

4.1. Nachträgliches Hinzufügen neuer Funktionen (Plug-and-Play (PnP))

Das vorab beschriebene Zentralrechnerkonzept und die Datenzentrik sind eine wesentliche Grundlage, um ein nachträgliches (After-Sale) Hinzufügen von neuen Fahrzeugfunktionen und neuer Fahrzeughardware (z.B. Sensorik) per PnP zu ermöglichen. Bezüglich bereits existierender und neu hinzuzufügender sicherheitskritischer Anwendungen muss das PnP-Konzept besonderen Qualitätsmerkmalen genügen, um die Safety-Eigenschaften nicht negativ zu beeinflussen.

Warum PnP? Eine wesentliche Beschaffenheit von Heimcomputern ist die individuelle nachträgliche Erweiterbarkeit um neue Funktionalitäten in Form von neu installierten Programmen oder hinzufügbarer interner oder externer Hardware. Hierdurch wird es ermöglicht, eine alternde Basisplattform durch Erweiterungen aktuell zu halten. Dem Kunden wird es zudem durch persönliche Modifikationen erlaubt, sein System individuell zu gestalten. Auch die Entwicklung der letzten Jahre im Mobiltelefonbereich hat gezeigt, dass Kunden individuelle Erweiterungsmöglichkeiten fordern, welche hier in Form des Verkaufs von Anwendungen über online Applikationsmärkte umgesetzt sind.

Im Automobilbereich ist eine solche Erweiterbarkeit der Fahrzeugfunktionalität ebenfalls wünschenswert. Derzeit ist dieses jedoch nach dem Verkauf kaum möglich. Beim Kauf eines Neuwagens muss sich der Käufer schon während der Bestellung über alle Wünsche bzgl. der erweiterten Ausstattungsoptionen im Klaren sein. Eine nachträgliche Erweiterung ist derzeit i.d.R. aus technischen Gründen kaum möglich. Besonders gilt dies für Fahrzeugfunktionen, die zum Kaufzeitpunkt noch gar nicht verfügbar waren.

Auch aus OEM-Sicht hätte eine Erweiterbarkeit der Fahrzeugfunktionalität enorme Vorteile. So lässt sich der Übergang von einem Fahrzeugmodell zu dessen Nachfolger fließender gestaltet, da eine Erweiterung der Fahrzeugfunktionalität durch das PnP vereinfacht wird. Hierdurch werden auch kürzere Modellintervalle möglich. Auch die Wiederverwendbarkeit bzw. die Aktualisierbarkeit der bereits vorhandenen Software wird erhöht. Gleiches gilt für den Übergang zwischen verschiedenen Fahrzeugmodellen, die auf einem ähnlichen Zentralrechner basieren.

Durch das in diesem Artikel beschriebene PnP-Konzept wird ein Weg aufgezeigt, die nachträgliche Erweiterbarkeit der Fahrzeugfunktionalität zu ermöglichen, sowohl um neue Software- als auch um neue Hardwarebausteine (z. B. Sensoren und Aktoren).

Um das PnP ohne weitere Benutzerinteraktion autonom durchzuführen, ist eine Schlüsselvoraussetzung die Möglichkeit der dynamischen Selbst-Konfiguration des Fahrzeugs. Das Vorgehen hierbei muss Safety-Anforderungen erfüllen (siehe auch [Ar12] und [Ar09]) und zudem durch geeignete Security-Mechanismen abgesichert sein (siehe Kapitel 4.3). Um die Erweiterbarkeit zu gewährleisten sind zudem standardisierte Schnittstellen notwendig.

Während des PnP-Vorgangs muss abgesichert werden, dass eine neue Funktion die vorhandenen Funktionen nicht durch Seiteneffekte beeinflusst. Dieses betrifft unter anderem das Scheduling, die Kommunikation, sowie den Zugriff auf Aktoren. Ein Mittel ist hierbei die Aufteilung unterschiedlicher Arten von Funktionen in voneinander entkoppelte Partitionen. Die Partitionierung kann anhand verschiedener Kriterien erfolgen, z. B. anhand der Sicherheitsstufen. Geeignete Datenfusions- und Arbitrierungskonzepte sind ebenfalls zur Absicherung der Seiteneffekte notwendig.

In unserem Konzept findet das PnP in der Regel in einem speziellen Wartungsmodus statt, teilweise aber auch im normalen Fahrzeugbetrieb (in einem reduzierten Umfang, welcher im Folgenden näher beschrieben wird).

PnP im normalen Fahrbetrieb:
Aus Safety-Gründen ist das PnP im Fahrbetrieb nur sehr eingeschränkt möglich. Beim Hinzufügen von neuen Softwarefunktionen in den Zentralrechner werden lediglich freie Ressourcen zur Verfügung gestellt, die zuvor für das PnP vorreserviert wurden (inkrementelles Addon). Ressourcen von bereits vorhandenen Funktionen bleiben hiervon unangetastet.

Falls für eine solche additive Konfigurationserweiterung nicht genügend freie Ressourcen zur Verfügung stehen, da beispielsweise der Verschnitt freier Ressourcen über die Kontrollrechner hinweg ungünstig ist obwohl die Gesamtressourcen ausreichen würden, wird ein Übergang in den Wartungsmodus des Systems notwendig (siehe auch „PnP im Wartungsmodus“). Hierbei kann eine Rekonfiguration stattfinden, bei der auch bereits bestehende Teile der Konfiguration verändert werden können. Währenddessen ist das Fahrzeug nicht fahrbereit.

PnP im Wartungsmodus:
Im Wartungsmodus ist es möglich, bestehende Teile der Systemkonfiguration anzupassen. So kann hier unter anderem das Deployment von Funktionen auf die Kontrollrechner geändert werden, wie auch die Vergabe von Speicher- und Rechenressourcen sowie die Netzwerkkonfiguration. Dieses erlaubt die Optimierung der Systemkonfiguration. Die Validierung der berechneten Konfiguration anhand verschiedener Kriterien ist ebenfalls Bestandteil des PnP-Vorgangs.

Dieses ermöglicht nun partiell die Integration von Funktionen, die während des Normalbetriebs nicht in die Konfiguration eingebunden werden konnten. Solche Funktionen werden bei der Initialisierung im Wartungsmodus automatisch erkannt. Ist dennoch das Hinzufügen einer neuen Funktion nicht möglich, so wird diese abgelehnt oder eine vorhandene Funktion wird deinstalliert.

Im Wartungsmodus kann das System zudem um neue Hardwaremodule ergänzt werden (z.B. Sensoren und Aktoren), die automatisch erkannt und eingebunden werden. Bei der Initialisierung wird zunächst überprüft, ob Hardwaremodule hinzugefügt oder entfernt wurden. Nachdem alle Hardwaremodule erkannt, auf Konformität geprüft und konfiguriert wurden, werden die Softwarefunktionen initialisiert. Manche Funktionen und Hardwaremodule sind obligatorisch (beispielsweise zur Umsetzung von Grundfunktionen wie Lenken, Bremsen und Beschleunigen), ohne die das Fahrzeug nicht als fahrbereit klassifiziert wird. Andere Funktionen sind hingegen optional (beispielsweise eine Einparkhilfe).

Ebenfalls ist beim PnP zu berücksichtigen, dass sich das Fahrzeug in verschiedenen Betriebsmodi befinden kann. Eine Fahrzeugfunktion kann hierbei so konfiguriert werden, dass sie nur in bestimmten Fahrzeugmodi aktiv sein soll. Bezüglich der Selbst-Konfiguration des Fahrzeuges während des PnP bedeutet dies, dass eine Konfigurationsinstanz für jeden Fahrzeugmodus berechnet werden muss.

4.2. Redundanz und Safety

Die SW-basierte Integration von Funktionen oberhalb der datenorientierten Middleware erfordert weitere Maßnahmen, wenn es sich um sicherheitskritische Funktionen handelt. Die in Kapitel 2 beschriebene Redundanz der zentralen DCCs kann zur Fehlererkennung und Maskierung genutzt werden, aber die SW-Funktionen müssen im Normalfall auf diese Redundanz angepasst werden. Ein Ziel der Middleware ist es, die Applikations-entwicklung von den Sicherheitsmechanismen der DCC-Hardware-Überwachung, -Umschaltung und Degradation zu trennen, damit die SW-Entwicklung sich auf die Funktionalität und nicht auf die Eigenheiten der DCC konzentrieren kann (horizontale Schichtung).

Die „generische“, funktionsunabhängige Safety bietet im Wesentlichen Fehlerursachenbereich eindeutig zu erkennen und das Eindringen von Fehlern darin zu verhindern [Ar09]. Ein DCC bildet als Rechnerpaar einen solchen Bereich. Beide enthaltene Rechner empfangen und senden im fehlerfreien Fall die gleichen Daten für ASIL-eingestufte Funktionen. Das sog. IO-Management prüft eingehende Daten auf Konsistenz, bevor sie den Applikationen über die MW weitergeleitet werden. Gleiches gilt für ausgehende Daten. Fehler werden nach außen gemeldet bzw. sind eindeutig von außen erkennbar, da auch die Ereignisse doppelt gesendet werden. Dies sichert die Datenkonsistenz (es werden keine fehlerhaften Daten verarbeitet).

Im Falle von Inkonsistenzen wird dieser Fehler dem *Plattform-Management* gemeldet. Dies ist eine auf allen DCCs aktive SW-Komponente, die Aufbau und Konfiguration des Systems sowie mögliche Degradationsstufen kennt. Fehlerindikatoren aus den Applikationen sowie o.g. Integritätsfehler werden hier behandelt. Reaktions-möglichkeiten umfassen: DCC-Umschaltung auf Slaves und die Abschaltung einzelner Applikationen. Das Plattform-Management sichert dadurch Aktionskonsistenz (die Zentralrechner führen die gleichen Applikationen aus).

Der zyklische Ablauf der Konsistenzsicherung (10ms Zyklen) erlaubt Fehlererkennung und Umschaltung auf einen bereits aktiven (hot-standby) Slave-DCC innerhalb einer Toleranzzeit von 50ms. Auf diesem Wege stellt die Middleware eine fehlersichere Ablaufumgebung für Applikationen zur Verfügung, die auch ASIL-D fail-operational Anforderungen genügt. Diese Sicherung der DCCs enthebt natürlich die Applikationsentwicklung nicht von dem Zwang, nach den Anforderungen der jeweiligen Kritikalitätseinstufung (ASIL-Level) zu entwickeln.

4.3. Security

Security spielt in Kraftfahrzeugen bereits heute eine wesentliche Rolle, um absichtliche Angriffe abzuwehren. Wesentliche Security-Funktionalitäten betreffen die Wegfahrsperre und Fahrzeugschlüssel, korrekter Kilometerstand, zugangsgeschützter Wartungs- und Diagnosezugang einschließlich Feature-Freischaltung und Software-Update („Flashen“) ebenso wie ein Schutz der Fahrzeug-externen Kommunikation mit Backend-Diensten, mit nomadischen Geräten und mit anderen Fahrzeugen (Car-2-X-Kommunikation). Elektrofahrzeuge werden außerdem kryptographische Security-Features für einen Manipulationsschutz des Ladeprotokolls sowie für die Unterstützung von Plug-and-Charge realisieren [Fa11]. Diese Security-Features sind teilweise in Hardware realisiert, um die Performance bzw. die Robustheit der Implementierung zu erhöhen.
Auch in der neuen Systemarchitektur müssen diese bekannten Security-Funktionen robust realisierbar sein. Der in diesem Artikel beschrieben Ansatz für die Fahrzeug-Systemarchitektur erfordert jedoch darüber hinausgehende, spezifische Security-Funktionalitäten. Die wesentlichen Treiber für Security sind dabei das Zusammenführen unterschiedlicher Arten von Funktionen auf einem Zentralrechner sowie die Erweiterbarkeit durch PnP um neue Software-Funktionen wie auch um neue Hardware-Komponenten (Sensoren, Aktoren).

1. *Zusammenführen von unterschiedlich kritischen Funktionen:* Es werden unterschiedlich kritische Funktionen in einer gemeinsamen, verteilten Ausführungsumgebung realisiert. Die Applikationen sind bezüglich ihrer Safety-Anforderung, ihrer Echtzeit-Anforderungen sowie ihrer eigenen Security-Anforderungen unterschiedlich kritisch. Diese Funktionen müssen voneinander logisch separiert werden, um eine gegenseitige Beeinflussung auch bei Ausführung in einer gemeinsamen Ausführungsumgebung zu verhindern.
2. *Erweiterbarkeit (PnP):* Es muss ermittelt werden, ob eine gewünschte PnP-Operation zulässig ist (Freigabe entsprechend Fahrzeug-Security-Policy, Lizensierung/Know-How-Schutz, für PnP-Operation berechtigter Nutzer), sodass nach einer PnP-Operation eine zulässige Konfiguration vorliegt. Bei einer PnP-Operation müssen die Mechanismen zur Separierung abhängig von den ermittelten Attributen konfiguriert werden. Mit dem Fahrzeugbus direkt dürfen nur vertrauenswürdige Komponenten direkt verbunden werden. Andere Komponenten dürfen nur eingeschränkt Zugriff auf den Fahrzeugbus erhalten.

Weitere Anforderungen an das Design der Security-Lösung kommen aus dem Realisierungs- und Anwendungsumfeld. Aufgrund der erläuterten Safety-Mechanismen

muss die Security-Lösung eine redundante Berechnung von kryptographischen Operationen, eine redundante Übertragung und ein Failover zwischen redundanten Rechnern unterstützen, ohne dass dadurch die Security selbst gefährdet wird. Außerdem ist im Fahrbetrieb ein „fail-operational“ Verhalten erforderlich, damit bei detektierten Security-Vorfällen ein einfaches Abweisen nicht immer eine angemessene Reaktion darstellt.

4.3.1. Security-Lösungsbausteine

Dieser Abschnitt gibt einen Überblick über erforderliche, für diese Architektur spezifische Security-Lösungsbausteine:

- Separierung von Funktionsdomänen: Zwischen unterschiedlichen Funktionsdomäne (z.B. Safety, Body, Komfort, Vehicle Security, Infotainment) wird durch eine Partitionierung die Rückwirkungsfreiheit erreicht. Auch bei einer Fehlfunktion oder einer erfolgreichen Manipulation eines offeneren Funktionsbereichs ist dadurch sichergestellt, dass die kritische Funktionalität nicht beeinträchtigt wird.
- Separierung von Applikationen: Einzelne Applikationen werden voneinander isoliert, sodass eine Beeinflussung durch eine andere Applikation nur über vorgesehene, zugelassene Datenpfade möglich ist, d.h. über berechtigte Zugriffe auf die Signaldatenbank. Die Applikationen sind außerdem von der Ausführungsumgebung RTE isoliert, sodass eine Applikation nicht die interne Funktion der RTE manipulieren kann. Zugriffe auf Sensoren/Aktoren sind nur indirekt über die Signaldatenbank möglich.
- Zugriffskontrolle auf Signaldatenbank: Eine Applikation kann auf Einträge der Signaldatenbank nur zugreifen, soweit sie dazu berechtigt ist.
- Eine Komponente kann nur auf Einträge der Signaldatenbank zugreifen, soweit sie dazu berechtigt ist. Nur vertrauenswürdige Komponenten dürfen direkt mit dem Fahrzeugbus verbunden sein. Andere Komponenten erhalten Zugriff über eine Anschaltkomponente „Vehicle Security Network Guard“. Diese stellt sicher, dass nur zulässige Zugriffe auf den Fahrzeugbus und damit die Signaldatenbank möglich sind.
- Authentisierung einer Software-Applikation. Eine Software-Applikation wird authentisiert, um deren für die PnP-Einbindung verwendeten Attribute korrekt zu ermitteln. Ebenso erfolgt eine Device Authentisierung von Hardware-Komponente, um deren für die PnP-Einbindung verwendeten Attribute korrekt zu ermitteln.
- PnP Security Policy: Bei einem PnP-Vorgang erfolgt eine Prüfung, ob eine PnP Operation zulässig ist. Dabei müssen unterschiedliche Kriterien und Policies unterstützt werden: Die betroffene Fahrzeugfunktionalität (Funktionsdomäne), Lizensierung, der durchführende Nutzer.
- PnP Security Konfiguration: Die vorgesehenen Security-Mechanismen zur Separierung und Zugriffskontrolle müssen bei einer PnP-Operation konfiguriert werden.

Eine robuste Implementierung dieser Security-Features in der RTE kann durch weitere Security-Maßnahmen unterstützt werden. Wesentliche Aspekte hierbei betreffen die

Schlüsselspeicherung, z.B. in einem Hardware Secure Element, eine robuste Implementierung von kryptographischen Algorithmen (Seitenkanalresistenz, Fehlerresistenz), ein Security-Hypervisor, ein sicheres Booten, Kommunikationssicherheit auf dem Fahrzeugbus und die Detektion von Manipulationen. Im Fehlerfall sind durch entsprechende Log-Einträge im Fehlerspeicher die Ursache und die Verantwortlichkeit ermittelbar.

4.3.2. Realisierungskonzept Systemarchitektur

Die RACE-spezifischen Security-Funktionen werden durch Security-Module des Runtime Environment realisiert. Für die Erweiterbarkeit (PnP-Manager) benötigte Security-Module sind:

- Device Authentication Module: Authentisierung einer Hardware-Komponente
- Application Authentication Module: Authentisierung einer Software-Applikation
- PnP Security-Policy Manager: Prüfen einer PnP-Operation auf Zulässigkeit
- Access Policy Provider: Berechtigungen bei einem PnP-Vorgang automatisch erzeugen.
- Access Policy Configurator: Berechtigungen auf Enforcement-Komponente konfigurieren.

Für die Durchsetzung der Separierung zur Laufzeit werden folgende Module verwendet:

- Signaldatenbank Access Control: Zugriffskontrolle auf Einträge der Signaldatenbank
- Vehicle Network Access Guard (Real-Time Firewall): Zugriffskontrolle auf den Fahrzeugbus und damit die Signaldatenbank durch eine Hardware-Komponente
- Hypervisor: Angriffs-resistente Separierung von Funktionsbereichen (Partitionierung)

Diese RACE-spezifischen Security-Funktionen verwenden als Basis allgemein bekannte Security-Funktionen zur Schlüsselspeicherung, kryptographische Algorithmen, sicheren Kommunikation, Policy-Prüfungen, Fahrerauthentisierung etc. Weiterhin stellt die RTE einzelnen Applikationen diese Security-Basisfunktionen bereit, um Applikations-spezifische Security-Funktionalität effizient realisieren zu können.

Innerhalb des geschlossenen RTE-Security-Kerns ist ein IT-Security-Schutz nicht zwingend erforderlich. Dabei wird jedoch ein ausreichender physikalischer Schutz der betroffenen Komponenten vorausgesetzt. Die oben beschriebenen Security-Mechanismen schützen vor möglicherweise manipulierten Komponenten, die sich außerhalb des RTE-Security-Kerns befinden, d.h. die Applikationen und die Hardware-Komponenten. Weitere Security-Funktionen können jedoch einen Beitrag leisten, den RACE-Security-Kern selbst in Ergänzung zu physikalischen Schutzmaßnahmen angriffsresistent zu realisieren.

4.3.3. Realisierungskonzept Middleware

Die zu Beginn des Kapitels 4.3 formulierten prinzipiellen Ziele der RACE-Security Architektur werden durch den Security Manager und dessen Teilmodule implementiert. Im Folgenden werden einige Module des Security Managers vorgestellt, wie sie in der Abbildung 3 dargestellt werden. Die zur Verfügung gestellten Dienste werden näher betrachtet, die erzeugten Daten und innerer Abläufe erklärt.

Zur Authentifizierung bietet der Security Manager ein *Authentification Module* an. Dieses Modul realisiert Authentifizierungsprozesse bei der Installation und der Aktualisierung von Komponenten. Diese Authentifizierung basiert zum Teil auf dem Austausch und der Überprüfung von Zertifikaten, die durch autorisierte Stellen für die Anwendung ausgestellt wurden. Daneben wird eine Authentifizierung beim Austausch von Daten zwischen verteilten Middlewareinstanzen oder Komponenten durchgeführt. Das Modul benötigt für die Durchführung dieses Dienstes einen kryptographischen Dienst.

Das *Crypto-Service Modul* bietet dem System kryptographische Dienste an. Die für die Berechnungen nötigen Schlüssel werden nicht selbst verwaltet, sondern sind zuvor von einem dafür spezifischen Dienst, dem Key-Management Modul angefordert. Beide Module bilden mit dem Secure Storage eine gesonderte vertrauenswürdige Zone, die es erlaubt direkt auf das verwaltete Schlüsselmaterial zuzugreifen und entsprechend dem erlaubten Zweck zu verwenden. Daneben verwaltet das Modul die Konfigurationseinträge für die im System vorhandenen Verschlüsselungsalgorithmen, wie z.B. symmetrische oder asymmetrische Verfahren. Dies ermöglicht ein späteres Einspielen, Aktualisieren oder Löschen von Algorithmen. Werden neue Algorithmen mittels PnP in das System eingepflegt, alte oder schwache Algorithmen entfernt oder auf den neuesten Stand gebracht, führt dies zu einer Anpassung der im Modul verwalteten und im Secure Storage Modul hinterlegten Konfiguration.

Die Verwaltung von kryptographischen Schlüsseln übernimmt das *Key Management Module*. Hier können Schlüssel in das System eingepflegt, entfernt und nach Bedarf angefordert werden. Da direkter Zugriff auf Schlüssel ein Sicherheitsrisiko darstellt, werden die Schlüssel nicht direkt verfügbar gemacht, sondern nur indirekt auf Anforderung bereitgestellt.

Das Speichern von Konfigurationsdateien, Schlüsselmaterial, Zertifikaten und weiteren Middleware immanenten Daten wird durch das *Secure Storage Module* übernommen. Dieser ist selbst verschlüsselt und bietet die Daten nur vorgelagerten Modulen nach Autorisierung über die Schnittstellen des Security Manager Moduls an. Dabei kann das Modul von in Hardware existierender Lösungen (z.B. TPM / HSM) unterstützt werden. Aber das ganze kann auch als reine Softwarelösung angesehen werden, wenn es keine Möglichkeit gibt, spezifische Hardwareunterstützung zu nutzen.

Neben den bereits vorgestellten Modulen existieren noch weitere Module, die ihre Dienste der Middleware und den Anwendungen zur Verfügung stellen. Grundsätzlich umfasst das die Dienste *Secure Communication Module* zur Bereitstellung sicherer

Kommunikation mit anderen Middlewareinstanzen und Anwendungen, sowie das *Policy Module* zum Prüfen von auszuführenden Aktionen auf deren Zulässigkeit.

Neben den sicheren Verbindungen zwischen den verteilten Middlewareinstanzen baut das *Secure Communication Module* auch die sicheren Verbindungen zwischen einzelnen verteilten Anwendungen auf. Dabei dient dieses Modul als zentrale Anlaufstelle für übergreifende Kommunikation und führt die hierfür nötigen Prozeduren und Modul-Interaktionen transparent durch. Dies schließt die Überprüfung der Verbindungs-Policies, die Authentifizierung und das konkrete Absichern der Verbindung ein.

Das *Policy Module* dient als zentrale Verwaltungs- und Entscheidungsstelle für die Anwendung von Policies. Module oder auch Anwendungen können hier Entscheidungen über geplante Aktionen oder Parameter für die Aktionen anfragen. Das Enforcement der Policy-Entscheidungen übernimmt hierbei die jeweils anfragende Instanz.

Die Überwachung der korrekten Funktionsweise wird durch den *Intrusion Monitor* realisiert. Dabei wird das Verhalten der Software zentral protokolliert und auf Auffälligkeiten untersucht. Bei einem identifizierten Angriff können lokale Reaktionen eingeleitet oder entsprechende Informationen an eine weitere den Angriff betreffende Middlewareinstanz weitergeleitet werden. Neben der Überwachung von Anwendungen beobachtet der Intrusion Monitor die Kommunikation, die über den Interface Manager verschickt wird. Hierbei arbeitet der Intrusion Monitor mit klassischen Ansätzen um schädlichen Datenverkehr zu erkennen und geeignete Gegenmaßnahmen einzuleiten.

Das *Pseudonymization Module* verändert fahrzeugbezogene Daten derart, dass diese nicht mehr dem Fahrzeug und seinem Halter zugeordnet werden können. Sollen Daten mit der Außenwelt ausgetauscht werden, müssen alle Fahrzeugbeziehungen zuerst vom Modul durch ein Pseudonym ersetzt werden. Dies soll verhindern, dass Fahrzeuginformationen aussagekräftig gesammelt und zum Nutzen dritter ausgewertet werden können. Auch verwaltet dieses Modul die Zugriffsrechte auf einzelne fahrzeugbezogene und halterbezogene Daten.

4.4. Weitere Qualitätseigenschaften

Die datenzentrische Middleware unterstützt die *Übertragbarkeit* von Datenproduzenten und -konsumenten. Sie sind voneinander entkoppelt, wodurch eine nahezu beliebige Platzierung von Funktionen auf den DCCs möglich wird. Die Entkoppelung der Funktionen sowohl untereinander als auch von der Hardware ermöglicht die effiziente, plattformunabhängige Implementierung und somit eine einfache Übertragung auf ein neues Zielsystem. Die Anpassung der Verschaltung erfolgt automatisch durch die Middleware.

Die zentralisierte Datenbank unterstützt die *Testbarkeit* des Systems. Die Ein/Ausgaben der Anwendungen sind übe die Datenbank verfügbar und die Schnittstellen können sowohl für Unit- als auch für Integrationstests genutzt werden. Traces und auch Fehlerinjektionen sind über diese Komponente der Middleware leicht möglich. Des Weiteren sind die Mechanismen der Middleware selbst leichter testbar, da die

Ergebnisse von Konsistenzprüfung, Zeitmessungen, etc. als Output der Middleware über die Datenbasis abfragbar werden.

In Zukunft werden wohl nur die kleinsten ECUs ohne *Multicore-Support* auskommen können. Erfahrung aus der Programmierung großer IT-Systeme hat gezeigt, dass eine datenfluss-zentrierte Programmierung leichter über mehrere Prozessorkerne skaliert als eine Task-synchrone Programmierung. Der vorgestellte Ansatz unterstützt einen solchen Datenfluss und erlaubt zusammen mit der Separierung von Partitionen in der RTE die Kombination mit deterministischen harten Echtzeitaufgaben. Die extrem kleinen Zykluszeiten der Aktoren werden in der vorgestellten Systemarchitektur von den koordinierenden Aufgaben der Zentralrechner getrennt.

Die *Kosten* sind ein wesentlicher Treiber für den automobilen Architekturentwurf. Der vorgestellte Ansatz ist kein Weg, Siliziumfläche oder Stücklisten zu minimieren. Vielmehr zielt er auf die Erweiterbarkeit und Entwicklungsgeschwindigkeit ab, in dem Basismechanismen der Rekonfiguration und Safety in einer Referenzarchitektur so von den Anwendungen entkoppelt werden, dass eine getrennte Anwendungsentwicklung ermöglicht wird.

5. Zusammenfassung

Die vorgestellte datenzentrische Middleware unterstützt die SW-basierte Integration von Funktionen. Über die bereitgestellten Daten lassen sich Funktionen kombinieren, die von unterschiedlichen Zulieferern entwickelt werden. Automatische Rekonfigurationsverfahren erlauben das nachträgliche Einbringen von Funktionen, dies unterstützt auch verschiedenen Entwicklungszyklen für einzelne Funktionen. Die entworfene Systemarchitektur erlaubt eine Kombination mit Safety-Mechanismen, die einen fail-operational Betrieb unterstützen.

Die Umsetzung hat im Projekt RACE begonnen [Ra12]. Die Erfahrungen können zukünftig mit vorhandenen AUTOSAR-Basisdiensten kombiniert werden und ggf. als Input für eine Erweiterung des Standards dienen; dies ist aber nicht Teil des laufenden Projekts. Das Projekt untersucht ebenfalls ein Migrationsszenario, das nicht die vollen (Safety) Garantien nutzt und damit eine schrittweise Einführung in bestehende Systeme darstellt.

Literatur

[Ar09] Armbruster, M.; Zimmer, E.; Lehmann, M.; Reichel, R. et al.: "Modularisation of Safety & Control for X-By-Wire Multiapplication-Platforms," SAE Int. J. Passeng. Cars - Electron. Electr. Syst. 1(1):8-17, 2009, doi:10.4271/2008-01-0113.

[Ar12] Armbruster, M. et al.: Ethernet meets Safety, 4. Elektronik Automotive Congress, 2012

[Br11] Broy, M.: „Mit welcher Software fährt das Auto der Zukunft?“, ATZ extra, 125 Jahre Automobil, pp. 92-97, April 2011

[Bu12] Buckl, C.; Camek, A.; Kainz, G.; Simon, C.; Mercep, L.; Staehle, H.; Knoll, A.: , "The software car: Building ICT architectures for future electric vehicles," IEEE International Electric Vehicle Conference (IEVC), 2012, pp.1-8, 4-8 March 2012

[CH12] CHROMOSOME Middleware; http://chromosome.fortiss.org

[Fa11] Falk, R.; Fries, S.: Securing the Electric Vehicle Charging Infrastructure – Current status and potential next steps, 27. vdi/VW Gemeinschaftstagung “Automotive Security”, Oct 2011, Berlin, VDI-Berichte 2131, VDI-Verlag Düsseldorf.

[Fo12] Mehr Software (im) Wagen, Informations- und Kommunikationstechnik (IKT) als Motor der Elektromobilität der Zukunft; http://www.fortiss.org/ikt2030

[HA11] Projekt HAVE-IT, FP7, Highly automated vehicles for intelligent transport; http://www.haveit-eu.org

[He01] Henzinger, T. A.; Horowitz, B.; Kirsch, C. M.: Giotto: A time-triggered language for em-bedded programming, Proceedings of the First International Workshop on Embedded Software (EMSOFT), 2001, 166 - 184.

[ISO11] ISO/DIS 26262: - Road vehicles – Functional Safety — Part 1-10

[RA12] Projekt RACE, Robust and Reliable Automotive Computing Environment for Future eCars; http://www.projekt-race.de

[Sp02] Spiegelberg, G.; „Ein Beitrag zur Erhöhung der Verkehrssicherheit und Funktionalität von Fahrzeugen unter Einbindung des Antriebstrangmoduls MOTionX-ACT®“, ISBN 9783898733670

[SP02] SPARC Secure Propulsion using Advanced Redundant Control, FP6, IST-2002-507859.