DynaSoft: Dynamisch selbstorganisierende Softwaresysteme für Automobile

Marc Zeller, Gereon Weiss, Falk Langer, Mike Heidrich, Dirk Eilers
Fraunhofer Einrichtung für Systeme der Kommunikationstechnik ESK
Hansastrasse 32
80686 München
{marc.zeller, gereon.weiss, falk.langer, mike.heidrich, dirk.eilers}@esk.fraunhofer.de

Abstract: Software ist über die Jahre zu dem wichtigsten Innovationstreiber in der Automobilindustrie geworden. Der Anteil der Software im Automobil steigt jährlich aufgrund der Vorteile einer softwarebasierten Realisierung vieler Fahrzeugfunktionen rasant an. Die wachsende Anzahl, sowie die wachsende Vernetzung und Variantenvielfalt der Funktionen führt zu einer immer schneller wachsenden Komplexität automobiler Softwaresysteme. Um diese auch in Zukunft beherrschen zu können, sind grundsätzlich neue Entwicklungsansätze notwendig. Die Methoden der Selbstorganisation bilden hierfür eine vielversprechende Lösung. Aus diesem Grund beschäftigt sich das Projekt DynaSoft mit der Umsetzung der Selbstorganisation in automobilen Softwaresystemen. Im Folgenden wird eine domänenspezifische Adaption der Selbstorganisation im Automobil vorgestellt und Begriffe der Selbstorganisation werden im automobilen Kontext definiert. Es werden Herausforderungen aufgezeigt, die sich bei der Umsetzung von Selbstorganisation im Automobil stellen. Schließlich werden die im Projekt DynaSoft erarbeiteten Lösungsansätze und identifizierte Problemstellungen bei der Einführung selbstorganisierender Software in Fahrzeuge beschrieben.

1 Einführung

Seit 1976 das erste Mal Software in Fahrzeugen eingesetzt wurde, hat sich die Komplexität der Softwaresysteme im Automobil enorm erhöht. Heute werden die Funktionsmerkmale moderner Fahrzeuge zu einem Großteil durch Elektronik und Software realisiert. Das Spektrum der Software reicht dabei von sicherheitskritischen Funktionen zur Steuerung des Motors oder der Bremsen mit harten Echtzeitanforderungen, über Funktionen im Komfortbereich (wie z.B. die Steuerung der Klimaanlage oder elektrischer Fensterhebern), bis hin zu Entertainment- und Telematikfunktionen zur Unterhaltung des Fahrers. Dies kann an verschiedenen Merkmalen verdeutlicht werden: Moderne Fahrzeuge der Premiumklasse beinhalten heute schon bis zu 2.500 „atomare“ Softwarefunktionen, die dem Kunden ca. 270 Dienstmerkmale (Features) zur Verfügung stellen [PBKS07]. Diese Software wird verteilt auf bis zu 90 Steuergeräten (Electronic Control Units, ECUs) ausgeführt [Bro05].

Für die zukünftige Entwicklung im Automobilbereich sind zwei wichtige Trends erkennbar: Eine steigende Funktionsvielfalt und ein dabei steigender Anteil der Fahrzeugsoftware an der Automobilelektronik. Zukünftige Generationen von Fahrzeugen werden eine gan-

ze Reihe neuer Dienste für den Endkunden beinhalten. Beispiele hierfür sind Funktionen im Rahmen der aktiven Fahrsicherheit (der Schwerpunkt liegt hier auf Fahrerassistenzsystemen), Funktionen die durch neue Antriebskonzepte entstehen (z.B. neue Motorsteuerungen für Hybridantriebe) oder durch neue Ausstattungs- und Komfortmerkmale (wie zum Beispiel neue Infotainment-Funktionen). Eine Vielzahl dieser Funktionen wird auch zukünftig durch Software realisiert werden, was zwangsläufig zu einer Steigerung des Anteils und der Bedeutung von Software im Automobil führt. Allerdings führen diese neuen *Features* [CE00] auch zu einer erheblichen Steigerung der Komplexität in der Systemarchitektur zukünftiger Automobile. Dies ist beispielsweise dadurch gekennzeichnet, dass neue Funktionen wie Fahrerassistenzsysteme auf mehrere Fahrzeugdomänen (z.B. Antriebsstrang und Infotainment) zugreifen. Dies führt zu einer wachsenden Vernetzung der Domänen untereinander und damit zu einem erhöhten Kommunikationsbedarf. Ein weiterer wichtiger Aspekt ist zudem die steigende Anzahl an Funktionsvarianten, die beispielsweise durch kundenspezifische Ausstattungsvarianten oder länderspezifische Reglementierungen entstehen. Gleichzeitig stellt die Automobilindustrie seit jeher sehr hohe Anforderungen an die Qualität der Software im Fahrzeug. Diese Anforderungen müssen auch in Zukunft trotz wachsender Komplexität eingehalten werden, auch wenn das Management dieser Systeme von außen zunehmend schwieriger - wenn nicht sogar unbeherrschbar - wird.

In den letzten Jahren hat sich der Ansatz der Selbstorganisation in der Forschung als vielversprechendes Konzept erwiesen, um Systeme zu managen, die zu komplex sind, um sie noch mit traditionellen Methoden zu beherrschen. Hierfür werden Regelkreise eingesetzt, die das System kontrollieren und um sogenannte Selbst-X-Eigenschaften (z.B. Selbstheilung, Selbstkonfiguration, Selbstschutz, usw.) [Wal04] erweitern.

2001 stellte IBM das Autonomic Comupting (AC) Paradigma vor [PH05]. Ziel dieses Ansatzes ist das Management von autonomen Elementen durch einen Regelkreis, der jedes Element überwacht, die aktuellen Umweltbedingungen analysiert, entsprechende Veränderungen des Elements plant und diese anschließend umsetzt. Der Fokus liegt dabei auf dem Management großer Server-Systeme ohne das Eingreifen eines Administrators. Das Organic Computing (OC) [Sch05] Forschungsprogramm der Deutschen Forschungsgemeinschaft (DFG) versucht aufbauend auf der Methodik des Autonomic Computing selbstorganisierende System zu erforschen, die sich lebensähnlich, also organisch verhalten. Der Fokus der Forschung liegt auf dem Einsatz der Selbstorganisation in verteilten eingebetteten Systemen.

Die Methoden der Selbstorganisation sind auch für die Beherrschbarkeit der immer stärker anwachsenden Softwaresysteme im Automobilbereich eine mögliche Lösung (Abbildung 1).

Die Arbeit ist wie folgt gegliedert: In Kapitel 2 wird Selbstorganisation im domänenspezifischen Kontext von Automobilen vorgestellt. In Kapitel 3 wird aufgezeigt, welche Herausforderungen sich bei der Umsetzung von Selbstorganisation im Automobil zu überwinden sind. Anschließend stellen wir in Kapitel 4 im Projekt DynaSoft erforschte Lösungsansätze für dynamisch, selbstorganisierende Softwaresysteme im Automobil vor. Schließlich wird in Kapitel 5 diese Arbeit zusammengefasst und ein Ausblick auf geplante Forschungsaktivitäten gegeben.

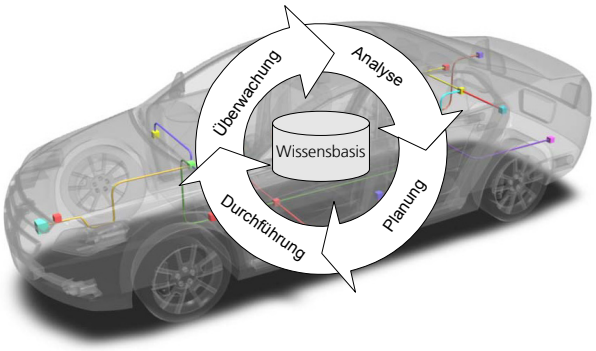

Abbildung 1: Selbstorganisation im Automobil

2 Selbstorganisation im Automotive-Bereich

Unter dem Sammelbegriff Selbstorganisation wurden im AC/OC und angrenzenden Forschungsbereichen verschiedene Begriffe geprägt wie beispielsweise die sogenannten Selbst-X-Eigenschaften eines Systems. Aufgrund ihrer allgemeinen Bedeutung für selbstorganisierende Systeme werden die für einen Einsatz im Fahrzeug relevanten Begriffe im Folgenden beschrieben. Dabei wird insbesondere auf ihre Rolle im domänenspezifischen Kontext Automobil eingegangen.

2.1 Adaptivität

Adaptivität ist eine Grundvoraussetzung für Selbstorganisation. Ein System ist adaptiv, wenn es sich selbstständig an veränderte Umweltbedingungen anpassen kann [Zad63]. Damit ein System adaptiv reagieren kann, muss es den eigenen Kontext zu einem bestimmten Grad erfassen und beurteilen können. Dazu benötigt es ein Modell, welches das System und seine gültigen Zustände beschreibt. Der Abgleich des aktuell vorgefundenen Kontexts mit einem Referenzzustand ermöglicht es, den derzeitigen Systemzustand zu beurteilen. Dieses sogenannte Selbstbewusstwein (Self-Awareness) bildet die Grundlage für die Adaptivität eines Systems bzw. eines Teilsystems.

Auch heute gibt es bereits Beispiele für adaptives Verhalten im Automobil. So passt beispielsweise die Motorsteuerung die Einspritzung des Benzins an das aktuelle Fahrverhalten an. Dieses adaptive Verhalten ist allerdings begrenzt auf klassische Regelungs- bzw. Steuerungsfunktionen und lässt Veränderungen nur in sehr kleinen fest vergebenen Varianten zu. In einem adaptiven Softwaresystem für Fahrzeuge, soll die Struktur des Systems an Veränderungen der Umweltbedingungen anpassbar sein. Hierfür muss die Zuordnung zwischen Software-Komponenten und Hardware-Ressourcen dynamisch zur Laufzeit änderbar sein. Obwohl Adaptivität auch in Hardware realisiert werden kann, be-

schränkt sich das Projekt Dynasoft auf adaptive Software. Der Einsatz rekonfigurierbarer Hardware wird innerhalb des Projekts nicht berücksichtigt, da dessen zukunftsnaher Einsatz in Fahrzeugen aktuell nicht abzusehen ist.

2.2 Emergenz

Emergenz ist definiert, als die Eigenschaft eines Gesamtsystems, welche nicht durch einfache Summation von Teileigenschaften errechnet werden kann [MSvdMW04]. Sie ist Resultat eines adaptiven, selbstorganisierenden Prozesses. Im Gegensatz zum klassischen Top-Down Entwurf technischer Systeme, ist Emergenz ein sogenanntes Bottom-Up Phänomen. Durch Selbstorganisation in Teilsystemen entsteht ein Verhalten des Gesamtsystems, das nicht explizit definiert ist [WH04]. Da es sich bei Fahrzeugen um sicherheitskritische Systeme handelt, ist es äußerst wichtig, dass die vom Entwickler vorgegeben Anforderungen und Beschränkungen des Systems eingehalten werden. Emergenz in automobilen Systemen darf nicht zu unkontrollierten Systemzuständen führen. Für den Einsatz im Fahrzeug wird eine kontrollierte Emergenz [MS04] benötigt, die definierte Anforderungen und Beschränkungen einhält und nicht im Widerspruch zum Top-Down Entwurf des technischen Systems steht.

2.3 Selbst-X-Eigenschaften

Unter dem Begriff Selbst-X-Eigenschaften werden eine Reihe von Eigenschaften selbstorganisierender Systeme zusammengefasst, die solche Systeme selbst aufweisen. Hierunter fallen:

Selbstbeschreibung: Eine Selbstbeschreibung der Komponenten und Funktionen ist notwendig, um diese adaptiv verwalten und nutzen zu können. Die Beschreibungssprache und Beschreibungsmenge sollten hierbei soweit reduziert sein, dass sie ihre Verwendungszwecke erfüllen, da andernfalls der Mehrwert einer Selbstbeschreibung den erforderlichen Mehraufwand nicht rechtfertigt. Eine Auflösung, Verwendung und Verarbeitung der Selbstbeschreibung muss durch eine maschinelle Kontrollinstanz möglich sein.

Selbstmanagement: Das System soll seine eigene Funktionalität ohne den Eingriff von außen verwalten können. Die Komplexität dieses Systemmanagements kann durch die Erweiterung der Mächtigkeit einzelner Elemente verringert werden. So muss beispielsweise der Zustand von Elementen, die selbstbewusst (Self-Aware) sind, nicht ständig erfasst werden. Zudem erleichtern selbstbeschreibende Elemente die Verwaltung der Ressourcen, da deren Beschreibung von ihnen selbst geliefert wird. Je komplexer das Management einzelne Elemente ist, desto geringer wird die Komplexität des Gesamtmanagements des Systems. Dieses muss dann jedoch durch definierte Interaktion der einzelnen autonomen Elemente realisiert werden. Für den Einsatz im Fahrzeug muss ein Trade-Off zwischen einem Selbstmanagement des Gesamtsystems und dem Management dedizierter Elemente des Systems gefunden werden.

Selbstkonfiguration: Die Konfiguration eines komplexen Systems ist derzeit nur aufwendig von Experten durchzuführen. Mittels sich selbstkonfigurierender Elemente ist es möglich, verteilt einen gültigen Systemzustand zu erreichen. Ein manueller, fehleranfälliger Konfigurationsprozess entfällt. Außerdem ermöglicht die Selbstkonfiguration eine dynamische Integration von neuen Komponenten sowie Funktionen und erleichtert bei einer Selbstheilung das Erreichen eines gültigen Systemzustands. Der automobile Infotainmentstandard Media Oriented Systems Transport (MOST) [MOS] unterstützt beispielsweise bereits heute eine automatische Konfiguration durch eine zentrale Instanz, den sogenannten NetworkMaster. Jedoch wird Dynamik nur eingeschränkt zur Positionsunabhängigkeit einzelner ECUs respektive Funktionsblöcke eingesetzt.

Selbstheilung: Eine autonome Diagnose des eigenen Systemzustands ermöglicht es, ungültige Zustände zu erkennen. Mittels Selbstheilung soll nach dem Auftreten eines ungültigen Systemszustandes ein gültiger Systemzustand wiederhergestellt werden. Um eine vollständige Heilung des Systems zu erreichen, wird allerdings ein gewisses Maß an Redundanz vorausgesetzt. Die Möglichkeiten der Selbstheilung wachsen mit der Größe des Gesamtsystems, was einen Einsatz vor allem für Infotainment- und Telematikanwendungen interessant erscheinen lässt. Verzögerungen aufgrund der Durchführung einer Selbstheilung müssen immer zusätzlich Berücksichtigung finden.

Selbstschutz: Der Selbstschutz eines Elements ist in einer dynamischen Umwelt notwendig. In einem in abgeschlossene Domänen aufgeteiltem Fahrzeug bedeutet dieser Selbstschutz jedoch einen Mehraufwand, der derzeit nicht gerechtfertigt erscheint. Jedoch sind der Schutz vor kritischen Systemzuständen und das Voraussehen von Problemzuständen eine Möglichkeit, um kritischen Zuständen präventiv entgegenzuwirken und so die gestellten Sicherheitsanforderungen im Automobil zu erfüllen. Des Weiteren wird die Bedeutung von Selbstschutzmaßnahmen mit der Öffnung der Kommunikation des Automobils nach außen (z.B. Car-2-X Kommunikation [CAR]) zunehmen.

Selbstoptimierung: Das proaktive Suchen eines Elements nach einer möglichen Optimierung des eigenen Systems ist für das Erreichen eines optimalen Systemzustands förderlich. Doch sind einige Ressourcen hierfür konstant notwendig, um diese Optimierung zu erzielen. Im Automobil muss genau abgeschätzt werden, in wie fern eine Selbstoptimierung den kontinuierlichen Ressourcenaufwand rechtfertigt. Ein mögliches Trade-Off stellt hierbei die kontextbasierte Optimierung hinsichtlich verschiedener Szenarien dar, in denen sich ein Automobil befinden kann.

Um das Potential der hier genannten möglichen Eigenschaften durch Selbstorganisation in automobilen eingebetteten Systemen ausschöpfen zu können, sind diverse Herausforderungen in der Domäne zu bewältigen. Diese werden im folgenden Abschnitt näher beschrieben.

3 Herausforderungen bei der Umsetzung von Selbstorganisation im Automobil

Bei der Realisierung eines selbstorganisierenden Softwaresystems für Fahrzeuge gibt es eine Reihe von Herausforderungen, die zu bewältigen sind.

Heute sind Software-Funktionen im Fahrzeug fest einem bestimmten Steuergerät zugeordnet. Da die Anzahl der Steuergeräte für die Integration neuer Features nicht beliebig erweiterbar ist, sind neue Konzepte für eine flexiblere Zuordnung zwischen Funktionen und Steuergeräten erforderlich. Die Automotive Open System Architecture (AUTOSAR) [Aut] Initiative verfolgt seit 2002 das Ziel, eine standardisierte Softwarearchitektur für Automobile zu etablieren. Durch einen komponentenbasierten Ansatz werden dabei die Wiederverwendbarkeit und die Skalierbarkeit von Software im Automobil erhöht. Eine virtuelle Integration von Software-Komponenten mittels des sogenannten Virtual Function Bus (VFB) ermöglicht die Allokation von Software-Funktionen auf ECUs zur Entwurfszeit und trägt damit zu einer höheren Flexibilität bei der Entwicklung von eingebetteten Systeme im Fahrzeug bei. Um Adaptivität und Selbstorganisation in Fahrzeuge zu integrieren, ist eine Weiterentwicklung der heutigen komponentenbasierten Softwarearchitektur im Fahrzeug hin zu einer service-basierten Softwarearchitektur notwendig. Eine feingranulare Dekomposition von Features in Dienste (Services) und in „atomare“ Software-Funktionen verbessert die Wiederverwendbarkeit und erhöht die Flexibilität. Dienste können flexibel aus Funktionen zusammengesetzt werden, was zur Eliminierung von redundanten Implementierungen führt. Außerdem bietet sich durch eine feinere Dekomposition ein höherer Freiheitsgrad für die Anpassung der Systemkonfiguration zur Laufzeit.

Heutige Laufzeitumgebungen für Software in der Automobilindustrie, wie OSEK/VDX [OSE] oder AUTOSAR, werden rein statisch zur Entwurfszeit konfiguriert und erlauben keinerlei dynamische Änderungen der Konfiguration zur Laufzeit (z.B. das Erzeugen neuer Tasks). Um Adaptivität und Selbstorganisation zu ermöglichen, ist eine dynamische Laufzeitumgebung für Fahrzeuge notwendig, die eine dynamische Zuordnung von Software auf Steuergeräte im Automobil ermöglicht. Hierfür ist eine Abkehr vom heute statischen Systementwurf notwendig. Dazu müssen die Ressourcen der einzelnen Steuergeräte, wie Prozessorleistung, Speicher, etc. dynamisch verwaltet werden.

In heutigen automobilen eingebetteten Systemen existiert eine enge Kopplung von Sensoren und Aktoren an Steuergeräte über deren I/O-Schnittstellen. Durch die zunehmende Vernetzung verschiedener Funktionen - auch über Domänengrenzen hinweg (z.B. durch Fahrassistenzsysteme) - müssen Sensoren und Aktoren immer häufiger fahrzeugweit für alle Funktionen zugänglich sein. Durch eine Trennung der Sensoren (Eingabe) und Aktoren (Ausgabe) von der Verarbeitung (Steuergerät), können Funktionen von überall aus auf Sensoren sowie Aktoren zugreifen. Dadurch steht beim Ausfall eines Steuergerätes weiterhin die Funktion der Sensoren/Aktoren zur Verfügung.

Eine weitere Herausforderung stellt die Heterogenität heutiger Bordnetzarchitekturen im Automobil dar. Die Vielzahl an unterschiedlichen Hardware-Plattformen und die verschiedenen Bussysteme erschweren eine dynamische Zuordnung von Software-Funktionen auf Hardware-Ressourcen zur Laufzeit. Funktionen müssen zur Migration auf ein anderes

Steuergerät entweder neu kompiliert werden, was die Dauer der Anpassung des Systems enorm erhöht, oder der Programmcode muss bereits in vorkompilierter Form für die entsprechende Hardware-Plattform vorliegen. Bei der Vielzahl an unterschiedlichen Plattformen erhöht dies den Speicheraufwand enorm und ist nicht kosteneffizient. Diese Heterogenität muss überwunden werden, um den Aufwand der dynamischen Zuordnung von Software zur Laufzeit zu reduzieren.

Softwaresysteme im Fahrzeug setzen sich aus einer Vielzahl von Funktionen mit unterschiedlichen Anforderungen an Echtzeitfähigkeit und Sicherheit zusammen. Heute werden diese Anforderungen durch die Aufteilung der Software in verschiedene Domänen (Infotainment, Antriebsstrang, Komfort, Chassis) berücksichtigt. In selbstorganisierenden Softwaresystemen müssen diese Anforderungen auch ohne eine physikalische Trennung der Funktionen eingehalten werden. Dies führt zu einer impliziten Einschränkung der Konfigurationsmöglichkeiten des Systems. Diese Beschränkungen müssen aus dem Entwurf ausgeleitet und zur Laufzeit berücksichtigt werden. Insbesondere darf das adaptive Verhalten nicht sicherheitskritische Funktionen beeinflussen. Aus diesem Grund sind sowohl die Auswirkung der Adaptivität als auch ihre Einschränkungen beim Einsatz in sicherheitskritischen Systemen zu berücksichtigen.

Um ein selbstorganisierendes (technisches) System zu realisieren, ist eine Kontrollinstanz notwendig, die Informationen über das System sammelt, diese analysiert und auf dieser Grundlage Entscheidungen trifft, wie das System verändert werden soll, um die gegebenen Systemziele zu erreichen [MWJ+07]. Eine derartige Kontrollinstanz muss sicherstellen, dass das System sich zu jeder Zeit in einem ordnungsgemäßen Zustand befindet. Damit die Kontrollinstanz die relevanten Informationen aus dem System gewinnen kann, müssen Informationen über die vorhandenen Hardware-Ressourcen und Software-Funktionen des Systems zur Laufzeit verfügbar sein. Die Beschreibung jeder Komponente des Systems (sowohl Hardware als auch Software) muss aus dem Entwurf des Systems ausgeleitet werden. Je mehr Informationen eine solche Beschreibung enthält, desto höher ist der Grad an Adaptivität des Systems. Zusätzlich nimmt aber auch der Aufwand für die Auswertung der Selbstbeschreibung zur Laufzeit zu. Wichtig ist ein Trade-Off zu finden, welche Informationen die Systembeschreibung zur Laufzeit umfassen muss.

Um diese Herausforderungen bei der Umsetzung der Selbstorganisation im Automobil zu bewältigen, ist ein Entwurfsprozess notwendig, der die Modellierung und Absicherung der Adaptivität unter Berücksichtigung der domänenspezifischen Anforderungen ermöglicht. Außerdem wird eine Laufzeitumgebung benötigt, welche die im Entwurf spezifizierten Anforderungen und Einschränkungen des Systems überwacht und eine Selbstorganisation zur Laufzeit ermöglicht. Ein möglicher Lösungsansatz hierfür wird im Projekt DynaSoft behandelt, welcher im nächsten Abschnitt vorgestellt wird.

4 DynaSoft: Lösungsansatz für dynamische, selbstorganisierende Softwaresysteme im Automobil

Ein selbstorganisierendes Softwaresystem im Automobil wird im Projekt DynaSoft erforscht. Hierbei werden Ansätze untersucht, die genannten Herausforderungen bei der Umsetzung von Selbstorganisation im Automobil zu lösen und damit ein dynamisches, selbstorganisierendes Softwaresystem für Fahrzeuge aufzubauen.

4.1 Ausgesuchte Anwendung von Selbstorganisation im Fahrzeug

Durch die Erweitung der Softwaresysteme eines Fahrzeugs um Selbstorganisation können einige wesentliche Vorteile gegenüber dem heutigen Stand der Technik erzielt werden.

Ein Automobil unterliegt einer wechselnden Umgebung. Auf einer Autobahn werden Dienstmerkmale wie beispielsweise die Geschwindigkeitsregelanlage, der Sicherheitsabstandsregler, der Spurhalteassistent oder die adaptive Fahrgeschwindigkeitsregelung sinnvoll eingesetzt. Im Stadtverkehr werden andere oder angepasste Fahrerassistenzfunktionen verwendet, wie z.B. der Parkassistent. Ein Nachtsichtassistent, adaptives Kurvenlicht oder ein Fernlichtassistent werden nur im Dunkeln bzw. bei eingeschränkter Sicht genutzt. Um eine optimale Ressourcenausnutzung zu erzielen, sollten die einzelnen Dienste szenariobezogen genutzt werden. Durch sich ausschließende Dienstmerkmale reduzieren sich hierbei die notwendigen Hardwareressourcen im Fahrzeug. Kontextbedingt nicht notwendige, aber grundsätzlich gleichzeitig mögliche Funktionen sparen Laufzeitressourcen (wie z.B. Energie, Rechenzeit, usw.).

Aus Kosten- und Effizienzgründen wird heute im Automobilbereich weitgehend auf Redundanz verzichtet. Der Ausfall von softwarebasierten Funktionen muss in der Regel durch eine Fachwerkstatt behoben werden. In einigen Fällen kann der Ausfall von elektronischen Komponenten zum vollständigen Defekt des Fahrzeugs führen. Diese Fehler in der Elektrik/Elektronik sind für den Kunden sehr negative Erfahrungen und können unter Umständen sogar das Leben des Fahrers gefährden. Durch den Einsatz von Techniken zur Selbstheilung des Softwaresystems im Fahrzeug werden die Ausfallsicherheit und die Verfügbarkeit des Systems ohne Kosten für zusätzliche Hardwareressourcen erhöht. Beispielsweise kann der Defekt eines Steuergeräts durch eine dynamische Anpassung der Systemkonfiguration kompensiert werden. So ist ein vorübergehender „Notbetrieb“ des Fahrzeuges möglich. Lebensbedrohliche Situationen für den Fahrer können so vermieden und die Zufriedenheit des Kunden erhöht werden.

Ein weiteres Anwendungsgebiet für Selbstorganisation im Automobil ist die automatische Systemintegration. Der Austausch von Fahrzeugkomponenten in der Werkstatt bzw. das Nachrüsten neuer Geräte (Aftermarket-Produkte) oder die Aktualisierung der Fahrzeugsoftware kann heutzutage zu Problemen führen, da unter Umständen die Softwarestände der einzelnen Komponenten nicht zu denen des Fahrzeuges passen. Die Selbstkonfiguration des Softwaresystems ermöglicht eine verbesserte Integration neuer Funktionen und Hardware-Komponenten und vermeidet Fehlfunktionen durch inkompatible Soft-

warestände. Durch die automatische Zuordnung der Software-Funktionen auf Steuergeräte reduziert ein selbstorganisierendes Softwaresystem die Komplexität für den Systemintegrator und den Aufwand bei der Produktion eines Fahrzeugs. Die manuelle Zuordnung von Funktionen auf Steuergeräte und das zeitaufwendige Aufspielen der Software sowie das Kodieren der Steuergeräte während der Produktion können hierdurch entfallen.

4.2 Entwurf selbstorganisierender Automobilsoftware

Der heutige Entwurfsprozess von Software im Automobilbereich ist stark geprägt von der klassischen mechanischen Entwicklung, wie sie seit Jahrzehnten in der Automobilindustrie vorherrscht.

Um die Komplexität eines verteilten eingebetteten Systems wie die Fahrzeugelektronik zu beherrschen, bedarf es eines speziellen Entwurfsprozesses, der die Abstraktion und Konkretisierung einzelner Systemkomponenten und des Systemverbunds erlaubt. Hierbei sind die Beschreibung und Beschreibungssprache ein entscheidender Faktor wie gut, d.h. wie nahe an der Realität, sich ein System auf verschiedenen Abstraktionsebenen beschreiben lässt. Neben der statischen ist, besonders in einem verteilten System, die dynamische Beschreibung von hoher Bedeutung.

Im Automobilbereich existieren verschiedene Bestrebungen, Standards für den Systementwurf zu etablieren. Hierunter fällt auch EAST-ADL2 [ATE] als modellbasierte Architekturbeschreibungssprache, die eine Integration der komponentenbasierten Architektur von AUTOSAR erlaubt, als eine vielversprechende durchgängige Möglichkeit für den Architekturentwurf im Bereich Automotive. Hiermit wird der durchgängige Entwurf eines statischen Systems auf UML-Basis ermöglicht. Um jedoch eine Adaptivität und damit eine Dynamik zur Laufzeit zu ermöglichen, müssen diese Entwurfsmethodiken erweitert werden. Dabei müssen insbesondere die frühe Absicherung des dynamischen Verhaltens, die Laufzeitbeschreibung sowie deren Ausleitung aus dem Entwurf und die Möglichkeit zur dynamischen Integration von neuen Systemkomponenten Berücksichtigung finden.

Der zulässige Grad an Adaptivität muss durch den Entwurf festgelegt werden, um die Anforderungen an die Sicherheit des Systems zu gewährleisten. Eine uneingeschränkte Selbstorganisation kann in einem solchen System nicht zugelassen werden, da bestimmte Anforderungen zu jedem Zeitpunkt von dem zugrunde liegenden System zu erfüllen sind. Hierbei ist auch die Abstraktion der Definition von Adaptivität entscheidend. Um das Gesamtsystem und sein Verhalten ausreichend abzusichern, sollte dieses auf hoher Abstraktionsebene bereits definiert sein. Dadurch kann das zulässige Soll-Verhalten, auch das der möglichen Adaptivität, begrenzt werden.

Auf hoher Abstraktionsebene können sogenannte *Features* die vom Benutzer wahrnehmbaren Funktionen darstellen. Zusätzlich zu statischen Features, die in einem Produkt vorhanden sind, können dynamische Features definiert werden. Diese stellen adaptive Funktionalität abstrakt dar. Sie besitzen Abhängigkeiten untereinander und gewisse Selektionskriterien, die ihre Auswahl festlegen. Erkannte Kontexte können dazu genutzt werden, dynamische Features der Fahrsituation entsprechend zu wählen. Automobilhersteller können

hierzu konkrete Szenarien festlegen, in denen bestimmte Funktionen des Automobils notwendig sind. Hierdurch können adaptive automobile Systeme durchgängig entworfen und beschrieben werden. Schließlich muss der Entwurf eines solchen Systems inklusive der definierten Adaptivität zur Laufzeit realisiert werden.

4.3 Laufzeitarchitektur für dynamisch, selbstorganisierende Softwaresysteme im Automobil

Eine selbstorganisierende Laufzeitumgebung muss zum einen Mechanismen bereitstellen, um Ressourcen dynamisch zu verwalten und die Adaptivität des Systems zu ermöglichen. Zum anderen müssen die im Entwurf spezifizierten Anforderungen und Einschränkungen an das System überwacht und eingehalten werden, um Selbstorganisation und Emergenz zu kontrollieren. Dies geschieht mittels eines Regelkreises, gemäß dem AC/OC-Paradigma. Das System wird kontinuierlich überwacht, Änderungen werden erkennt und bei Bedarf werden Änderungen am System geplant und ausführt.

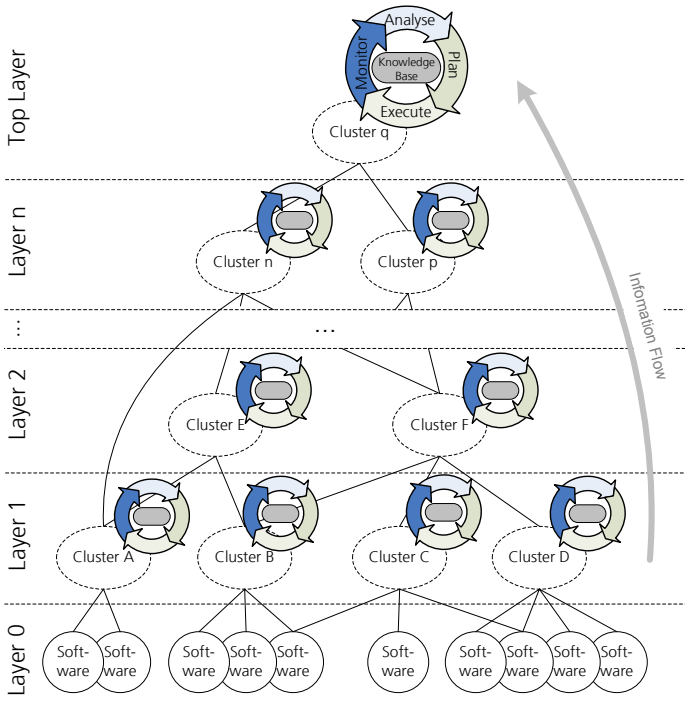

Abbildung 2: Hierarchische Kontrollarchitektur

Durch die hohe Komplexität, die softwarebasierte eingebettete Systeme in der Automobilindustrie heute erreicht haben, ist die Kontrolle des gesamten Fahrzeugs durch einen einzi-

gen Regelkreis nur schwer möglich. Zusätzlich gibt es bedingt durch die Vielfalt an Features, die durch Software realisiert werden, sehr unterschiedliche Anforderungen an die einzelnen Funktionen im Fahrzeug. Allgemein teilt man die unterschiedlichen Fahrzeugfunktionen nach ihrer Sicherheitsrelevanz in sogenannte Safety Integrity Level (SIL) [Int98] ein. Jede Sicherheitsstufe hat unterschiedliche Anforderungen und muss dementsprechend von einer Kontrollinstanz auf unterschiedliche Art und Weise berücksichtigt werden. Eine sicherheitskritische Funktion wie beispielsweise die Airbag-Steuerung darf zu keinem Zeitpunkt in ihrer Funktion beeinträchtigt sein. Eine Funktion aus dem Infotainment-Bereich, wie z.B. eine Freisprechanlage, kann hingegen jederzeit deaktiviert werden, ohne das Leben von Menschen zu gefährden. Die Laufzeitarchitektur muss diese Anforderungen berücksichtigen. Dazu wird das Gesamtsystem in *Cluster* aufgeteilt. Ein Cluster ist hier definiert als eine logische Gruppe von Software-Funktionen sowie einer Menge von Anforderungen und Zielen, die alle Elemente des Clusters erfüllen müssen. Jedes Cluster wird durch einen eigenen Regelkreis kontrolliert. Die Aufteilung in Cluster kann auf Grund funktionaler Abhängigkeiten, nicht-funktionaler Abhängigkeiten, der physikalischen Lage oder auf Grundlage von gemeinsamen Anforderungen und Zielen erfolgen. Durch wiederholtes Unterteilen des Systems ergibt sich eine mehrstufige, hierarchische Kontrollarchitektur (Abbildung 2).

Die unteren Ebenen der Hierarchie bilden kleine Cluster, in denen schnell und mit geeigneten Mitteln auf veränderte Bedingungen reagiert werden kann. Können Anforderungen, Beschränkungen oder Systemziele innerhalb eines Clusters nicht erfüllt werden, wird ein Regelkreis auf der nächst höheren Hierarchieebene angestoßen. Cluster auf höheren Ebenen kontrollieren eine höhere Anzahl an Elementen und umfassen einen größeren Lösungsraum. Somit ist die Möglichkeit eine neue valide Systemkonfiguration zu erreichen größer. Allerdings sind diese Cluster auch komplexer und aufwendiger zu verwalten. Dieser hierarchische Ansatz bietet jedoch den Vorteil, dass die Kontrollarchitektur sehr genau auf die Bedürfnisse des zu kontrollierenden Systems angepasst werden kann.

Jeder Regelkreis der Kontrollarchitektur besteht gemäß dem AC-Paradigma aus den vier Phasen:

Überwachung: Das System bzw. bestimmte Systemparameter müssen kontinuierlich überwacht werden, um schnell und zuverlässig Änderungen im System bzw. Änderungen der Umwelt feststellen zu können. Um Selbstheilung zu ermöglichen ist zudem eine automatisierte Erkennung eines Fehlverhaltens notwendig, um weitere Schritte anzustoßen. Klassischerweise beschäftigt sich die Überwachung und Fehlererkennung mit der Erkennung von Fehlfunktionen einzelner Komponenten. Dabei wird ein erwartetes Soll-Verhalten mit dem gemessen Ist-Verhalten verglichen. Weicht das Ist-Verhalten vom Soll-Verhalten ab, so kann man von einem Fehlerfall ausgehen. Die Darstellung des Soll-Verhaltens bzw. die Messung des Ist-Verhaltens ist dabei in den meisten Fällen sehr spezifisch auf die einzelne Komponente zugeschnitten. Die wachsende Komplexität, Abhängigkeit und Verteilung der verschiedenen Funktionen stellt dabei folgende zu lösende Probleme:

- Überwachung des gesamten Systemverhaltens: Obwohl alle einzelnen Komponenten korrekt arbeiten, weist das Gesamtsystem ein falsches Verhalten auf.

- Überwachung des dynamischen Systemverhaltens: Ein adaptives System kann verschiedene Systemkonfigurationen einnehmen. Damit ist es schwer, alle möglichen neuen Konfigurationsmöglichkeiten vorherzusagen (State Explosion) und mittels statischer Überwachungsmechanismen zu überprüfen.

- Erkennung von unbekannten Fehlern: Heutige Überwachungsmechanismen können nur sehr eingeschränkt zum Entwicklungszeitpunkt unbekannte Fehler erkennen. Dies ist vor allem darin begründet, dass in den meisten Überwachungstechniken bestimmte Fehlerbilder hinterlegt sind, die auftretende Fehler deklarieren. Treten Fehler auf, welche nicht in ein vorher bekanntes Muster passen, so werden diese nicht erkannt.

Analyse: Die Analyse muss den aktuellen, den gewünschten und den zukünftigen Zustand des Systems erkennen und vorhersagen. Dabei ist die Analyse des Systems eng gekoppelt mit der Überwachung, da die Ergebnisse aus der Überwachung (Beobachtung) des Systems direkt in die Analyse eingehen. Im Gegensatz zur Überwachung bezieht die Systemanalyse zusätzliche Informationen wie den Umgebungszustand und die definierten Systemziele in die Auswertung des Systemzustandes mit ein. Hierzu kann das 1996 von der vorgestellte Livingstone-Model [CPC03] herangezogen werden. Dieses beschreibt ein modellbasiertes Diagnoseverfahren für selbstkonfigurierende autonom agierende Raumsonden, welches vorhergesagtes Soll-Verhalten mit dem tatsächlichen Ist-Verhalten vergleicht und auf Basis des Systemmodels Aussagen über einzuleitende Maßnahmen treffen kann. In der Diagnose soll über die Fehlerursachenerkennung hinaus eine Aussage getroffen werden (in Abhängigkeit des Umgebungszustandes und der Systemziele), welche Funktionen für einen weiteren Betrieb des Systems zur Verfügung stehen müssen [WNN96]. Mit den so gewonnen Aussagen über die verfügbare Ressourcen und dem zukünftigen gewünschten Funktionen kann im nächsten Schritt (Planung) eine neue Systemkonfiguration hergestellt werden.

Planung: Die Laufzeitplanung erstellt oder wählt eine Menge von Aktionen aus, die Änderungen an der überwachten und verwalteten Ressource vornehmen. In einem Fahrzeug erstellt die Laufzeitplanung eine Zuordnung von Software-Funktionen zu Steuergeräten, die alle gestellten Anforderungen erfüllt. Dieses Zuordnungsproblem lässt sich sowohl als „Generalized Assignment Problem“ (GAP) [CVW90], als auch als „Constraint Satisfaction Problem“ (CSP) [DB07] formulieren. Da es sich jeweils um ein $\mathcal{NP}$-hartes Optimierungsproblem handelt, ist ein heuristischer Lösungsansatz notwendig, um das Problem zur Laufzeit zu lösen. Die Herausforderung in der Laufzeitplanung liegt darin, ein Trade-Off zwischen Rechendauer und Genauigkeit der Lösung zu finden, um den Anforderungen im Automobilbereich zu genügen.

Durchführung: Die Laufzeitdurchführung stellt die nötigen Mechanismen bereit, um die von der Laufzeitplanung vorgegebenen Aktionen auszuführen und so das System anzupassen. Im Softwaresystem des Fahrzeugs beziehen sich diese Änderungen auf das Aktivieren und Deaktivieren von Funktionen, sowie auf die Migration von Funktionen auf andere Steuergeräte. Hierbei ist wichtig, dass das System auch während der Anpassung noch seine Funktionalität erfüllt. Dies gilt besonders für sicherheitskritische Funktionen. Bei der Migration von Software kann unterschieden werden, ob der Kontext der Funktion

(Variablen, Stack, etc.), der Programmcode (binär oder als Quellcode) oder beides auf ein anderes Steuergerät transferiert werden soll. So kann beispielsweise eine sicherheitskritische Funktion auf mehreren Steuergeräten vorhanden sein, so dass bei der Migration nur der aktuelle Kontext der Funktion auf die andere ECU übertragen werden muss. Andere Funktionen, die nicht zu einem sicherheitskritischen Funktionsmerkmal gehören, können bei der Migration für eine neue Hardwareplattform neu kompiliert und dann als Binärcode an das entsprechende Steuergerät geschickt werden. Je nach Anforderungen der Funktion sind so verschiedene Methoden einsetzbar.

Eine solche Laufzeitarchitektur kann die im Entwurf spezifizierten Anforderungen und Systemziele zur Laufzeit überwachen und das System entsprechend an neue Bedingungen anpassen und so Dynamik durch Selbstorganisation zur Laufzeit umsetzen. Hiermit werden in dieser Arbeit das Potential selbstorganisierender Software im Automobil und die damit einhergeheneden Herausforderungen aufgezeigt.

5 Zusammenfassung und Ausblick

Die wachsende Komplexität in automobilen Softwaresystemen wird zunehmend unbeherrschbar. In dieser Arbeit haben wir aufgezeigt, wie die Methoden der Selbstorganisation zur Lösung dieses Problems beitragen können und gleichzeitig eine höhere Flexibilität und Effizienz erlauben. Des Weiteren wurden die domänenspezifischen Herausforderungen beschrieben, die sich bei der Umsetzung eines selbstorganisierenden Softwaresystems im Fahrzeug stellen. Um den Sicherheits- und Echtzeitanforderungen an automobile Software gerecht zu werden, müssen der Freiheitsgrad der Selbstorganisation und die resultierende Emergenz eingeschränkt werden. Dazu müssen Anforderungen und Beschränkungen des Systems im Entwurf beschrieben und zur Laufzeit überwacht und kontrolliert werden. Dies geschieht in einem durchgängigen Entwurfsprozess, der darüber hinaus eine kontinuierliche Absicherung des dynamischen Systems ermöglicht. Zur Laufzeit wird die Einhaltung der im Entwurf spezifizierten Anforderungen und Beschränkungen durch eine mehrstufige, hierarchische Kontrollarchitektur überwacht. Bei Bedarf wird das System neu konfiguriert, und die Zuordnung von Software zu Steuergeräten dynamisch angepasst.

Ungeachtet des aufgezeigten Potentials selbstorganisierender Softwareplattformen im Automobilbereich, müssen die hier vorgestellten Lösungsansätze für eine breite Akzeptanz ressourcen- und kosteneffizient umgesetzt werden. Hierbei muss insbesondere die Abwägung zwischen der Beherrschung der Laufzeitkomplexität durch selbstorganisierende Methoden und der mit diesen einhergehende Aufwand für die Integration von Selbstorganisation untersucht werden. Zudem muss die Umsetzung der Selbstorganisation den im Automobilbereich geforderten strengen Sicherheitsansprüchen genügen. Aus diesem Grund müssen für eine erfolgreiche Integration und Verbreitung selbstorganisierender Techniken in reale automobile Systeme, diese Methoden zur Beherrschung der wachsenden Komplexität von Automobilsoftware in Standardisierungsprozessen definiert werden.

Danksagung

Das Projekt DynaSoft wird durch das Bayrische Staatsministerium für Wirtschaft, Infrastruktur, Verkehr und Technologie gefördert.

Literatur

[ATE] ATESST. EAST-ADL 2.0 Specification. http://www.atesst.org/.

[Aut] Automotive Open System Architecture (AUTOSAR). http://www.autosar.org.

[Bro05] M. Broy. Automotive software and systems engineering. In *MEMOCODE '05: Proceedings of the 2nd ACM/IEEE International Conference on Formal Methods and Models for Co-Design*, Seiten 143–149. IEEE Computer Society, 2005.

[CAR] CAR 2 CAR Communication Consortium. http://www.car-to-car.org.

[CE00] K. Czarnecki und U.W. Eisenecker. *Generative programming: methods, tools, and applications*. Addison-Wesley, 2000.

[CPC03] Alessandro Cimatti, Charles Pecheur und Roberto Cavada. Formal verification of diagnosability via symbolic model checking. In *In Proceedings of the 18th International Joint Conference on Artificial Intelligence IJCAI03*, Seiten 363–369, 2003.

[CVW90] D.G. Cattrysse und L.N. Van Wassenhove. *A survey of algorithms for the generalized assignment problem*. Erasmus University, Econometric Institute, 1990.

[DB07] Michael Dinkel und Uwe Baumgarten. Self-Configuration of Vehicle Systems - Algorithms and Simulation. In *WIT '07: Proceedings of the 4th International Workshop on Intelligent Transportation*, Seiten 85–91, 2007.

[Int98] International Electrotechnical Commission (IEC). *IEC 61508: Functional safety of Electrical/Electronic/Programmable Electronic (E/E/PE) safety related systems*, 1998.

[MOS] MOST Cooperation. http://www.mostcooperation.com/home/index.html.

[MS04] C. Müller-Schloer. Organic computing: on the feasibility of controlled emergence. In *CODES+ISSS '04: Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, Seiten 2–5. ACM, 2004.

[MSvdMW04] C. Müller-Schloer, C. von der Malsburg und R.P. Würt. Organic Computing. *Informatik-Spektrum*, 27(4):332–336, 2004.

[MWJ+07] G. Mühl, M. Werner, M.A. Jaeger, K. Herrmann und H. Parzyjegla. On the Definitions of Self-Managing and Self-Organizing Systems. In *KiVS 2007 Workshop: Selbstorganisierende, Adaptive, Kontextsensitive verteilte Systeme (SAKS 2007)*, 2007.

[OSE] OSEK VDX Portal. http://www.osek-vdx.org.

[PBKS07] A. Pretschner, M. Broy, I.H. Kruger und T. Stauner. Software Engineering for Automotive Systems: A Roadmap. *Future of Software Engineering*, Seiten 55–71, 2007.

[PH05] M. Parashar und S. Hariri. Autonomic computing: An overview. *Lecture Notes in Computer Science*, 3566:257–269, 2005.

[Sch05] Hartmut Schmeck. Organic Computing - A New Vision for Distributed Embedded Systems. In *ISORC '05: Proceedings of the Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, Seiten 201–203. IEEE Computer Society, 2005.

[Wal04] K. Waldschmidt. Adaptive system architectures. In *18th International Parallel and Distributed Processing Symposium*, Seiten 147–, 2004.

[WH04] Tom De Wolf und Tom Holvoet. Emergence and Self-Organisation: a statement of similarities and differences. In *Lecture Notes in Artificial Intelligence*, Seiten 96–110. Springer, 2004.

[WNN96] Brian C. Williams, P. Pandurang Nayak und Urang Nayak. A Model-based Approach to Reactive Self-Configuring Systems. In *In Proceedings of AAAI-96*, Seiten 971–978, 1996.

[Zad63] L.A. Zadeh. On the definition of adaptivity. *Proceedings of the IEEE*, 51(3):469–470, 1963.