Automotive Betriebssysteme

Wolfgang Schröder-Preikschat

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl für Verteilte Systeme und Betriebssysteme
Martensstr. 1, 91058 Erlangen
http://www4.informatik.uni-erlangen.de/~wosch

Zusammenfassung. Ein bedeutender Marksektor für (tiefste) eingebettete Systeme ist die Automobilindustrie. Aus informatischer Sicht sind heutige Automobile eingebettete verteilte Systeme auf Rädern. Automobile mit über 80 vernetzten Steuergeräten sind bereits keine Seltenheit mehr. Derartige Gebilde sind ohne spezielle Systemsoftware nicht mehr betreibbar. Die weiter zunehmende Elektronifizierung von Fahrzeugen erfordert skalierbare Systemsoftware die einerseits maßgeschneidert auf gegebene und andererseits anpaßbar für zukünftige Anwendungsbereiche ist, wobei neben der Quantität auch der Diversität von Prozessoren pro Fahrzeug eine große Bedeutung zukommt. Die Arbeit behandelt exemplarisch Entwurf- und Implementierungsaspekte von Betriebssystemen im Kontext solcher eingebetteten Systeme.

1 Einleitung

Moderne Kraftfahrzeuge bilden höchst komplexe Systeme von Hard- und Software, sie sind aus informatischer Sicht betrachtet „verteilte Systeme auf Rädern“. Zur Regelung und Steuerung des mechatronischen Systems „Automobil“ kommt eine hohe Zahl verschiedenster Mikrocontroller zum Einsatz. Je nach Ausstattung und Funktion sind Kraftfahrzeuge mit über 80 solcher Spezialprozessoren bzw. Steuergeräte längst keine Utopie mehr.

Die Gesamtheit aller Steuergeräte bildet ein hochgradig vernetztes System auf Grundlage unterschiedlicher Bussysteme bzw. Kommunikationseinrichtungen. Das Netzwerk ist typischerweise problemorientiert partitioniert (Abb. 1). Unterschieden werden Subsysteme z. B. für Antrieb, Komfort, Infotainment, Armaturen und Diagnose. Eine weitere Unterteilung ergibt sich hinsichtlich der Betriebsart der Subsysteme: der Antriebsbereich etwa steht für harten Echtzeitbetrieb, wohingegen die anderen Bereiche lediglich festen oder gar nur schwachen Echtzeitbetrieb (wenn überhaupt) erfordern. Die mit einem solchen System gegebene Heterogenität von Hard- und Software hat unterschiedliche Dimensionen angenommen:

Prozessortechnologie Die in den Steuergeräten vorhandenen Mikrocontroller sind von 8-, 16- und 32-Bit Technologie. Diese Spannbreite stellt u. a. große Herausforderungen an die Softwareentwicklung, insbesondere vor dem

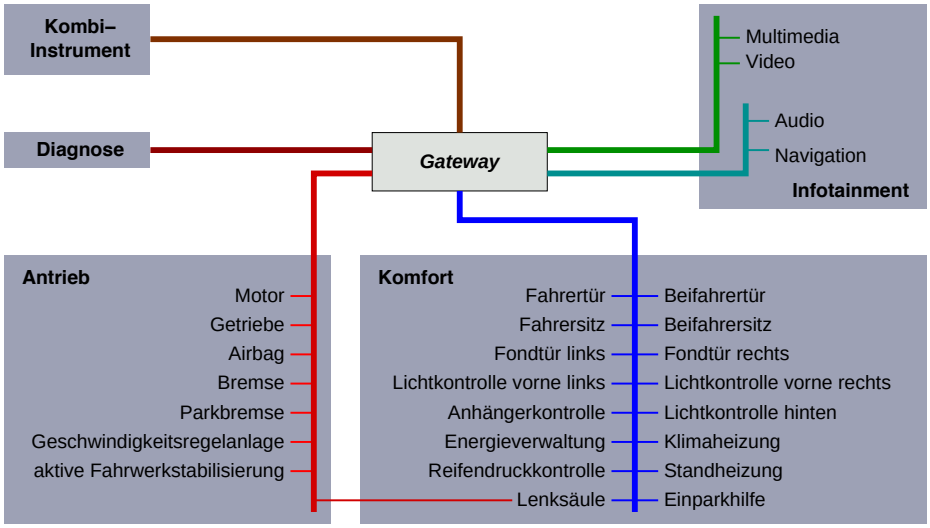

Abb. 1. Beispiel der Netzwerkstruktur innerhalb eines Automobils

Hintergrund der Wiederverwendbarkeit von Softwarekomponenten. Die Leistungsbandbreite hinsichtlich Geschwindigkeit und Speicherplatz kann unterschiedlicher nicht sein. Darüberhinaus kommt die Hardware von verschiedenen Herstellern und damit mit verschiedensten Befehlen und Befehlsformaten, Ein/Ausgabe- und Unterbrechungskonzepten und Entwicklungsumgebungen.

Netzwerkstruktur Vorherrschend ist nach wie vor CAN-Technologie [1], die in unterschiedlichen Leistungsklassen (*low-speed*, *high-speed*) eingebaut wird. Als kostengünstigere Variante zur Verbindung intelligenter Sensoren und Aktoren kommt mehr und mehr LIN [2] zum Einsatz. Im Bereich Infotainment ist MOST-Technologie [3] anzutreffen. Desweiteren werden Automobilhersteller spezifische Technologien eingesetzt, wie z. B. die *byteflight*- bzw. SI-Bus-Technologie von BMW [4]. Eine zukünftige Alternative stellt *FlexRay* [5] dar. Zusammenfassend kann festgehalten werden, dass sich die Komplexität der Netzinfrastruktur moderner Kraftfahrzeuge mit der von herkömmlichen lokalen Netzen gut messen kann.

Softwarearchitektur Hier muss unterschieden werden zwischen den Steuergeräten und den Zusatzgeräten insbesondere des Bereichs Infotainment (Audio, Video, Telefonie, usw.). In den Steuergeräten kommen durchgängig Spezialzweckbetriebssysteme zum Einsatz. Vorherrschend im Automobilsektor (europäischer Hersteller) sind OSEK-konforme Betriebssysteme. OSEK [6] definiert einen Betriebssystemstandard namhafter europäischer Automobilhersteller. Viele Zulieferer bieten Implementierungen dieses Standards an, die sich intern hinsichtlich ihrer nicht-funktionalen Eigenschaften zwar z. T. erheblich unterscheiden (indem Prozessabläufe z. B. ereignis- oder zeitge-

steuert [7] realisiert sind), nach außen aber eine einheitliche funktionale Sicht präsentieren. Die OSEK-Betriebssysteme sind zugeschnitten für Einsätze in Umgebungen äußerster Betriebsmittelknappheit (insbesondere in Bezug auf Speicher), wie sie z. B. typisch für Steuergeräte sind. Einen Kontrast dazu bildet etwa der Infotainmentsektor, in dem die Zusatzgeräte z. T. durch „herkömmliche“ Allgemeinzweckbetriebssysteme kontrolliert werden. So definiert sich die Gesamtheit der in einem Kraftfahrzeug vorliegenden Systemsoftware als „Schmelztiegel“ von Linux, QNX, VxWorks, iTRON, WindowsCE und OSEK-konformen Installationen.

In der Kraftfahrzeugindustrie ist der Anteil für Elektronik und Elektrotechnik an den Gesamtherstellungskosten eines Automobils von 8 % im Jahre 1965 auf heute ca. 35 % gestiegen [8]. Weitere Schätzungen besagen, dass innerhalb der Hard- und Software-Ausstattung eines Automobils die Kosten der Software zukünftig 38 % ausmachen, gegenüber 20 % heute. Dies würde einen Anteil von etwa 13 % an den Gesamtherstellungskosten ausmachen.

Der Umfang der Fahrzeug-Software wächst rasant, ist jedoch abhängig vom Hersteller, vom Modell und von der Ausstattung. Darüberhinaus lastet ein enormer Kostendruck auf diesen Industriezweig, der sich direkt z. B. in der zum Einsatz kommenden Prozessortechnologie niederschlägt. Dies verdeutlicht sich u. a. auch darin, dass im Jahre 2000 etwa 57.6 % aller weltweit hergestellten Prozessoren 8-Bit Technologie waren [9].[1] Mit den Steuergeräte-Stückzahlen pro Automobil und hochgerechnet mit den weltweiten Automobil-Stückzahlen jährlich (weit über 50 Millionen) ist die Kraftfahrzeugindustrie einer der Hauptabnehmer von Prozessortechnologie.

Der Kostendruck führt teilweise auch dazu, dass die zum Einsatz kommende Hardware nicht an der gegenwärtigen oberen Leistungsgrenze anzufinden ist. Allgemein gilt für den Bereich eingebetteter Systeme, dass die Prozessortechnologie typischerweise (mindestens) eine Generation zurückhängt [10]. Die Hardware für diesen Bereich besitzt im Regelfall mehr auf einem Chip integrierte Funktionseinheiten als für einen „nackten“ Prozessor erforderlich sind. Dies treibt die Transistoranzahl nach oben und erfordert, sofern überhaupt verfügbar, den Einsatz aufwendigerer Herstellungsverfahren. Prozessortechnologie der Vorgängergeneration(en) kommt mit weniger Transistoren aus und lässt Spielraum für die zusätzlichen Funktionseinheiten — ist dafür jedoch auch weniger leistungsfähig. Ein typisches Beispiel für diese Situation ist der Mikrocontroller, unabhängig von 8-, 16- oder 32-Bit-Technologie.

Die Dimensionen der Heterogentität der IT-Infrastruktur moderner Kraftfahrzeuge machen eines der Hauptprobleme deutlich: Variabilität der Software. Nicht nur, dass die Kraftfahrzeug-Software mittlerweile sehr große Ausmaße angenommen hat, sie ist darüberhinaus von einer Vielfalt, die ohne spezielle Werkzeuge und Entwicklungsprozesse nicht mehr beherrscht werden kann. Das weitergehende Problem dabei ist, dass der Stand der Kunst in der Softwaretechnologie einerseits (noch) keine Patentlösungen anbietet und er andererseits

[1] Eingebettete Systeme beanspruchten überhaupt 98.2 % der Gesamtproduktion für einen Markt, in dem z. B. Linux oder Windows ohne nennenswerte Bedeutung sind.

auch (noch) nicht durchgängig in allen Entwicklungsbereichen der Kraftfahrzeugdomäne praktiziert wird.

2 Betriebssystemanforderungen

Ein Automobil definiert grob zwei Problemdomänen für Betriebssysteme. Dabei handelt es sich einerseits um den Infotainmentbereich (Audio, Video, Telefonie, Internet, aber auch Navigation), der Anforderungen zum Multimediabetrieb des Automobils vorgibt. Für diesen Betrieb kommen recht leistungsfähige Rechnersysteme (32-Bit Technologie) zum Einsatz, mit QNX und VxWorks als bevorzugte Betriebssysteme. Verschiedentlich sind hier auch Windows CE und Varianten von Linux anzufinden. Von den Betriebssystemen wird die adäquate Unterstützung von schwach echtzeitfähigen Anwendungsprogrammen verlangt.

Im Falle der anderen Problemdomäne handelt es sich um den „klassischen" Steuerungsbereich, der das Automobil als mechatronisches System versteht. In diesem Bereich ist zusätzlich fester und harter Echtzeitbetrieb gefordert, je nach Steuergerät bzw. zu kontrollierendem Objekt (wie z. B. Einspritzanlage, ABS, Airbag oder Fensterheber). Hier werden Echtzeitbetriebssysteme eingesetzt, die auch unter äußerster Betriebsmittelknappheit ihre Dienste für die Anwendungsprogramme verrichten können müssen. Weit verbreitet sind OSEK-kompatible Betriebssysteme. QNX und VxWorks sind in diesem Bereich nicht prominent vertreten, ganz zu schweigen von Windows CE oder (embedded) Linux.

Die nachfolgenden Ausführungen beziehen sich auf Anforderungen an Echtzeitbetriebssysteme für eben diesen Steuerungsbereich. Alleinstellungsmerkmale werden diskutiert, die typisch für „automotive Betriebssysteme" sind.

2.1 Prozesseinplanung

Grundlegend für den Echtzeitbetrieb ist, dass die u. a. vom Betriebssystem auszuführenden Arbeiten zeitlich determiniert sein müssen. Daraus leitet sich direkt die Forderung ab nach algorithmischen Verfahren mit konstantem Aufwand. Typische Fälle dafür sind beispielsweise die Signalisierung oder Einplanung (*scheduling*) von Prozessen und die Vergabe von Speicher. Sind konstant aufwändige Verfahren unmöglich bzw. nicht durchgängig praktikabel, so muss mindestens eine feste obere Grenze für die Ausführungszeit (*worst case execution time*, WCET) gegeben sein. Für jeden einzelnen Systemaufruf ist seine Ausführungslatenz anzugeben und als nicht-funktionale Eigenschaft in der Schnittstellenbeschreibung mit aufzunehmen.

Zunehmend an Bedeutung gewinnt ein Sachverhalt, der der Bestimmung der WCET ähnlich ist, sich jedoch auf den Energieverbrauch der ablaufenden Prozesse bezieht. Die Energieversorgung von Automobilen ist je nach Ausstattung eine z. T. sehr kritische Komponente. Die Vergabe von Betriebsmitteln an Prozesse muss verstärkt energiegewahr erfolgen, um Arbeitszeit und Lebensdauer der Batterie zu maximieren. Ist eine feste obere Grenze des Energiebedarfs (*worst case energy demand*, WCED) eines CPU-Stoßes bekannt und bietet der Prozessor

Möglichkeiten der flexiblen Leistungssteuerung, so kann eine energiegewahre Prozesseinplanung die Senkung des Stromverbrauchs bei der Programmausführung zur Folge haben [11]. Hierzu sind ähnliche Analysetechniken erforderlich, wie sie auch zur Bestimmung der WCET Verwendung finden müssen.

2.2 Unterbrechungstransparenz

Asynchrone Programmunterbrechungen (*interrupts*) dienen der Signalisierung von externen Ereignissen, die z. B. Zustandsänderungen in der Umgebung anzeigen oder einem Zeitgebersignal entsprechen. Bei ereignisgesteuerten Systemen hat die Unterbrechung die Erzeugung des Echtzeitabbilds einer Echtzeitentität [12] zur Folge, um eine Zustandsänderung im zu kontrollierendem Objekt zu verfolgen. Bei zeitgesteuerten Systemen wird darüber die interne Zeitskala für die Abarbeitung von Prozessen eng mit der vorgegebenen externen Zeitskala der Umgebung synchronisiert.

Im Falle ereignisgesteuerter Systeme sind die durch die Unterbrechungen hervorgerufenen nebenläufigen Aktivitäten explizit zu koordinieren. Dabei birgt ein zeitweises Sperren der Unterbrechungen (*disable interrupts*) das Risiko in sich, Ereignisse zu verlieren und dadurch Zustandsänderungen eben nicht verfolgen zu können. Ideal sind daher Techniken, die asynchrone Programmunterbrechungen durchlässig erscheinen lassen und eine Unterbrechungstransparenz der Systemsoftware unterstützen [13].

In wie fern solche Sperrvorgänge jedoch kritisch in Bezug auf ein korrektes Verhalten des Gesamtsystems ist, hängt auch sehr stark von der Dynamik des zu kontrollierenden Objektes und der Frequenz der Zustandsänderungen ab. Bei hartem Echtzeitbetrieb ist Ereignisverlust nicht zu tolerieren. Weniger strikt verhält es sich bei festem bzw. weichem Echtzeitbetrieb, der einen gelegentlichen Verlust von Ereignissen zulässt. Das zu kontrollierende Objekt gibt die Art (weich, fest, hart) des Echtzeitbetriebs vor und damit auch die Anforderung (sperren tolierbar/nicht tolerierbar) an das Synchronisierungsverfahren. Entsprechend müssen Betriebssysteme für die jeweilige Problemdomäne maßgeschneiderte Lösungen zur Verfügung stellen.

2.3 Wiedereintrittsfähigkeit

Wiedereintritt bedeutet, dass ein Programm zu einem Zeitpunkt überlappt (im Falle von Monoprozessorsystemen) oder parallel (im Falle von Multiprozessorsystemen) in mehreren Inkarnationen aktiv sein kann. Um die korrekte Ausführung der anstehenden Arbeiten zu unterstützen, sind Programmzustände daher lokal in Bezug auf die jeweilige Inkarnation zu verwalten. Zur Unterstützung müssen Stapelkonzepte, eine ausgefeilte Entwurfs-/Implementierungstechnik wie auch spezielle Übersetzer Verwendung finden. Wo globale Programmzustände verwaltet werden müssen, ist für eine geeignete Koordination der Abläufe zu sorgen. Soweit anwendbar sollte dabei wartefreie Synchronisation [14] den Vorzug erhalten vor nicht-blockierender Synchronisation, die wiederum Techniken zur blockierenden Synchronisation ggf. vorzuziehen ist. Je nach Anwendungsfall müssen

die Betriebssysteme mit den entsprechenden (nicht-funktionalen) Eigenschaften versehen sein.

Wiedereintrittsfähigkeit verhilft dazu, die Latenzen bis zur Ereignisbehandlung bzw. Arbeitseinplanung und/oder -abfertigung auf ein Minimum zu reduzieren. Unterbrechungstransparenz ist eine unterstützende Maßnahme dafür, Verdrängung (*preemption*) von Prozessen eine andere. Bezüglich des zuletzt genannten Punktes ist jedoch hervorzuheben, dass Wiedereintrittsfähigkeit eine volle bzw. durchgängige Verdrängbarkeit (*full preemption*) impliziert: die ereignisgesteuerte Prozessumschaltung muss jederzeit möglich sein. Ein Betriebssystemkern, der sich z. B. wie im Falle von UNIX oder Linux umfassend wie ein Monitor verhält und Verdrängung „großflächig“ unterbindet, ist nur sehr schwach wiedereintrittsfähig und mit hohen Latenzen belastet.

2.4 Konzentration auf das Wesentliche

Betriebssysteme für (tiefste) eingebettete Systeme sollen nur jene Funktionen enthalten, die auch von dem jeweiligen Anwendungsfall verlangt werden. Diese Anforderung begründet sich in vielen eingebetteten Systemen gerade durch die z. T. nur in sehr knappen Umfang zur Verfügung stehenden Betriebsmittel (z. B. für Speicher und Energie). Überflüssige Software ist ein Luxus, der nur in eher seltenen Fällen akzeptiert wird. Automotive Betriebssysteme sind Spezialzweckbetriebssysteme, die anwendungsgewahre und speziell auf bestimmte Einsatzszenarien zugeschnittene Funktionen enthalten sollen.

Auch wenn sich die technologische Basis dahingehend verschieben sollte, dass zukünftig Rechnerbetriebsmittel nicht mehr in all zu spärlicher Form vorliegen, so ist dadurch noch längst nicht die Forderung nach minimalistischer Software aufgehoben. Ein anderes gewichtiges Argument für eine Konzentration auf die wesentlichen Funktionen, die ein Betriebssystem zu erbringen hat, ergibt sich allgemein in Hinblick auf die Korrektheit und Sicherheit (*safety*, *security*) der Software. Systeme, die nur die jeweils notwendigen Funktionen enthalten, sind vergleichsweise kleine Systeme und kleine Systeme sind einfache Systeme. Einfache Systeme wiederum sind verständlicher und leichter (formal) auf ihre Korrekheit hin zu überprüfen als komplexe Systeme. Darüberhinaus sind einfache Systeme effizient und besser erweiterbar. Gerade für sicherheitskritische eingebettete Systeme sind aus den Gründen „schlanke“ und „federgewichtige“ Softwarestrukturen von äußerst großer Relevanz.

Die Systemsoftware muss in jeder Beziehung leicht anpassbar sein, d. h., sie muss sich einerseits als erweiterbar und andererseits als „zusammenziehbar“ erweisen. Diese an sich gegenläufigen Ziele werden erreicht, wenn die Software als Programmfamilie [15,16] ausgelegt ist. Innerhalb einer Programmfamilie wird zwischen den einzelnen Familienmitgliedern ein hohes Maß an Wiederverwendbarkeit ermöglicht, wohingegen nach außen jedes Familienmitglied ein auf spezielle Bedürfnisse hin maßgeschneidertes Softwareprodukt darstellt. Hochspezialisierte Varianten von Betriebssystemen koexistieren innerhalb einer Softwareproduktlinie, wobei automotive Betriebssysteme dann einen eigenen Stammbaum innerhalb der Familie begründen.

3 Fallbeispiel PURE

PURE [17] ist eine Betriebssystemfamilie für eingebettete Systeme. Familienmitglieder liegen als leichtgewichtige Bibliotheksbetriebssysteme vor, die zum Übersetzungs- und/oder Bindezeitpunkt in sehr feinkörniger Weise an die Erfordernisse der Anwendung angepasst werden können und so in ihrem Betriebsmittelverbrauch mit den Anforderungen skalieren. Dabei definiert PURE/OSEK [18] den Zweig der OSEK-kompatiblen Betriebssysteme der PURE-Familie

3.1 Entwicklungsprinzipien

PURE Betriebssysteme sind familienbasiert entworfen und (überwiegend) objektorientiert implementiert. Der Entwurf und die Implementierung der Systemsoftware ist dabei jeweils inkrementell ausgelegt. Ausgehend von einer minimalen Teilmenge von Systemfunktionen erfolgt die funktionale Anreicherung um minimale Systemerweiterungen in minimalen Schritten. Der Entwurfsprozess verläuft von unten nach oben (*bottom-up*), er wird jedoch von oben nach unten (*top-down*) gesteuert. Ergebnis des Entwurfs ist eine anwendungsgewahre Softwarestruktur, die im Kern auf eine große Anzahl für viele Anwendungsfälle wiederverwendbare Abstraktionen zurückgreift und an der Oberfläche hoch spezialisierte Funktionen zur Verfügung stellt.

Die dabei entstehende tiefe funktionale Hierarchie von Systemabstraktionen wird auf eine Klassenstruktur abgebildet und objektorientiert implementiert. Je nach Eignung kommt dabei Vererbung wie auch Komposition zur Hierarchiebildung zum Einsatz. Bei konsequenter Umsetzung entsteht eine Vielzahl hierarchisch angeordneter und sehr feinkörnig ausgelegter Funktionen bzw. Klassen. Große Bedeutung kommt bei diesem Ansatz einer Variantenverwaltung zu, die die Gemeinsamkeiten wie auch Unterschiede der Mitglieder einer Betriebssystemfamilie auf Basis funktionaler und nicht-funktionaler Eigenschaften erfasst. Diese Eigenschaften bilden letztlich die Merkmale (*features*) von Betriebssystemfunktionen. Unter Ausnutzung solcher Merkmale können dann für die jeweils vorgegebenen Anwendungsfälle passende Betriebssysteme aus vorgefertigten Bausteinen (semi-) automatisch zusammengestellt und konfiguriert werden [19,20].

3.2 Ergebnisse

PURE ist in C++ implementiert und wurde auf x86-, sparc-, alpha-, m68k-, ppc60x, C167-, AVR- und ARM-basierenden Plattformen portiert. Ein gutes Beispiel für die Feinkörnigkeit des Entwurfs gibt der Nukleus, der aus über 100 Klassen mit insgesamt mehr als 600 exportierten Methoden besteht. Jede dieser Klassen implementiert dabei einen abstrakten Datentyp. Der Nukleus selbst bildet eine „Unterfamilie“ von PURE, die grob sechs Familienmitglieder nachfolgend genannter Eigenschaften umfasst:

interruptedly Reaktive Ausführung von Aufgaben rein auf Basis von Unterbrechungsbehandlungsroutinen.

serialize Unterbrechungstransparent synchronisierte, reaktive Ausführung von Aufgaben. Minimale Erweiterung von *interruptedly*.

exclusive Ausschließlich durch ein (einfädiges) Anwendungsprogramm kontrolliertes System.

cooperative Kooperative Ausführung von Aufgaben eines mehrfädigen Anwendungsprogramms. Minimale Erweiterung von *exclusive*.

non-preemptive Kooperative Ausführung von Aufgaben eines mehrfädigen, Ereignisse verarbeitenden Anwendungsprogramms. Minimale Erweiterung von *cooperative* und *serialize*.

preemptive Ereignisgesteuerte Ausführung von Aufgaben eines mehrfädigen Anwendungsprogramms. Minimale Erweiterung von *non-preemptive*.

Wie die Tabelle 1 zeigt, lassen sich trotz der hochmodularen Struktur des Nukleus sehr kleine und kompakte Systeme erzeugen [18]. Die Daten beziehen sich auf die x86-Portierung (übersetzt mit egcs 1.0.2).

Tabelle 1. Speicherverbrauch der Nukleusfamilie

Familienmitglied	Größe (in Bytes)			
	text	*data*	*bss*	gesamt
interrupedly	812	64	392	1268
serialize	1882	8	416	2306
exclusive	434	0	0	434
cooperative	1620	0	28	1648
non-preemptive	1671	0	28	1699
preemptive	3642	8	428	4062

Der Familienzweig für automotive Betriebssysteme entsteht durch die OSEK-Erweiterungen zum Nukleus. Dieser PURE-Zweig unterstützt die von einem OSEK-Betriebssystem erwarteten Konformitätsklassen und Einplanungsstrategien. Nicht unterstützt wird der erweiterte Fehlermodus. Anhand der vom OSEK-Konsortium definierten Konformitätstests wurde die volle Kompatibilität von PURE/OSEK zum OSEK-Standard bestätigt. Tabelle 2 zeigt einen Auszug der erzielten Ergebnisse. Beispielhaft sind die Ergebnisse der komplexesten OSEK-

Tabelle 2. Laufzeitergebnisse von PURE/OSEK

Einplanung	Synchronisation	*text*	*data*	Fadenwechsel
ECC2	Ereignismaske	2871	1052	94
	Semaphor	3351	1076	141
	handover	2095	1052	61
cooperative	Ereigniszähler	3242	2248	148
preemptive	Ereigniszähler	3674	2248	202

Konformitätsklasse ECC2 (Mehrfachaktivierung von Fäden, Ereignisverwaltung) dargestellt, in Abhängigkeit von verschiedenen Arten der Fadenkoordinierung. Im unteren Teil der Tabelle sind zum Vergleich erweiterte PURE-Funktionen gegenübergestellt. Abermals diente die x86-Portierung als Ausgangspunkt. Die Spalte für den Fadenwechsel listet die Anzahl der Taktzyklen, die der zu Grunde liegende Pentium II jeweils zum Kontextwechsel benötigte. Auch diese Ergebnisse demonstrieren, dass hochmodulare und erweiterungsfähige sowie portable und wartbare Systemsoftware keinesfalls im Widerspruch zu kompakten und effizienten Lösungen stehen muss.

4 Zusammenfassung

Echtzeitbetriebssysteme liegen in einer für herkömmliche (nicht echtzeitfähige) Betriebssysteme geradezu undenkbaren Manigfaltigkeit vor. Wesentlichen Beitrag an dieser Situation haben die eingebetteten Systeme, die eine enorme Spezialisierung und Plattformabhängigkeit der Systemsoftware bedingen. Etwa die Hälfte aller im Jahr 2004 entstandenen „einbettbaren“ Echtzeitbetriebssysteme sind proprietär und gehen nur als integraler Bestandteil eines eingebetteten Systems in den Markt. Die andere Hälfte teilen sich eine Vielzahl von Betriebssystemprodukten jenseits von Windows, UNIX, Linux oder MacOS. QNX oder VxWorks, als zwei sehr prominente Vertreter von kommerziell erfolgreichen Echtzeitbetriebssystemen, bilden dabei nur die Spitze des Eisbergs. Der Kraftfahrzeugbereich definiert dabei eine eigene Problemdomäne.

Alleinstellungsmerkmale funktionaler Natur finden sich bei der Einplanung von Prozessen, Verwaltung und Vergabe von Betriebsmitteln, Koordination nebenläufiger Aktivitäten, Unterbrechungstransparenz, Wiedereintrittsfähigkeit sowie Ablaufinvarianz der Systemsoftware. Für jedes dieser Themenkomplexe existieren unterschiedlichste Lösungsansätze, die ihre Bedeutung evtl. nur in sehr speziellen Anwendungsfällen besitzen. Kraftfahrzeuge im Großen und Steuergeräte im Kleinen geben solche Anwendungsfälle vor und die Systemsoftware muss den daraus erwachsenen Anforderungen entsprechen. Entwurf und Implementierung dieser Software muss die Wiederverwendbarkeit von Systemfunktionen fördern und gleichzeitig die Option zur Spezialisierbarkeit bieten. Programmfamilien, Objektorientierung und aspektorientierte Programmierung bietet dafür die adäquate Unterstützung, gepaart mit einer Merkmal basierten Modellierung und Konfigurierung der Softwarevarianten.

Literaturverzeichnis

1. International Standards Organization: Road vehicles—Interchange of digital information—Controller area network (CAN) for high-speed communication. ISO 11898, 1993.
2. Specks JW, Rajnak A: LIN — Protocol, Development Tools, and Software Interfaces for Local Interconnect Networks in Vehicles. In *Proceedings of the 9th International Conference on Electronic Systems for Vehicles*, Baden-Baden, Germany, 2000.

3. MOST Cooperation: Media Oriented Systems Transport—Multimedia and Control Networking Technology. `http://www.mostcooperation.com`, 2002. MOST Specification, Rev 2.2.
4. BMW AG et al.: Sicherheits- und Informationsbussystem (SI-BUS) — *byteflight*. `http://www.byteflight.com`.
5. Belschner R, et al.: FlexRay Requirements Specification, Version 2.0.2. Technical report, BMW AG, DaimlerChrysler AG, Robert Bosch GmbH, General Motors Opel AG, April 2002.
6. OSEK/VDX Steering Committee: OSEK/VDX Operating System, September 2001. Specification 2.2.
7. Kopetz H: *Event-Triggered versus Time-Triggered Real-Time Systems*, Lecture Notes in Computer Science 563. Springer, 1991.
8. Broy M, von der Beeck M, Krüger I: SOFTBED: Problemanalyse für ein Großverbundprojekt „Systemtechnik Automobil — Software für eingebettete Systeme“, 1998.
9. Tennenhouse D: Proactive Computing. Communications of the ACM, 43(5):43–50, 2000.
10. Heath S: *Embedded Systems Design*. Newnes, Butterworth-Heinemann, 1997.
11. Weißel A, Bellosa F: Process Cruise Control: Event-Driven Clock Scaling for Dynamic Power Management. In *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES 2002)*, Grenoble, France, 2002.
12. Kopetz H: *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, 1997.
13. Schön F, Schröder-Preikschat W, Spinczyk O, Spinczyk U: On Interrupt-Transparent Synchronization in an Embedded Object-Oriented Operating System. In *The Third IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2000)*, 270–277, Newport Beach, California, 2000. IEEE Computer Society.
14. Herlihy MP: Wait-Free Synchronization. *ACM Transactions on Programming Languages and Systems*, 13(1):123–149, 1991.
15. Parnas DL: On the Design and Development of Program Families. IEEE Transactions on Software Engineering, SE-5(2):1–9, 1976.
16. Parnas DL: Designing Software for Ease of Extension and Contraction. IEEE Transactions on Software Engineering, SE-5(2):128–138, 1979.
17. Beuche D, Guerrouat A, Papajewski H, Schröder-Preikschat W, Spinczyk O, Spinczyk U: The PURE Family of Object-Oriented Operating Systems for Deeply Embedded Systems. In *Proceedings of the 2nd IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'99)*, St Malo, France, 1999.
18. Papajewski H, Schröder-Preikschat W, Spinczyk O, and Spinczyk U: Die PURE-OSEK-API — Spezialisierung einer objektorientierten Betriebssystem-Familie. *PRAXIS Profiline — IN-CAR-COMPUTING*, 36–41, 2000.
19. Spinczyk O: *Aspektorientierung und Programmfamilien im Betriebssystembau*. PhD thesis, Otto-von-Guericke-Universität Magdeburg, 2003.
20. Beuche D: *Composition and Construction of Embedded Software Families*. PhD thesis, Otto-von-Guericke-Universität Magdeburg, 2004.