System-Entwicklung basierend auf der DECOS-Architektur

Hauke Fuhrmann, Christian-Albrechts-Universität zu Kiel
Institut für Informatik, Kiel, Deutschland 24098, haf@informatik.uni-kiel.de

Hendrik Geilsdorf, Hamburg University of Technology
Institute of Aircraft Systems Engineering, Hamburg, Deutschland 21129, h.geilsdorf@tuhh.de

Lothar Klein, LIEBHERR-AEROSPACE LINDENBERG GmbH
Software Development, Lindenberg, Deutschland 88153, Lothar.Klein@lli.liebherr.com

Stefan Schneele, EADS Deutschland GmbH, Corporate Research Centre
Electronics & Integration, München, Deutschland 81663, stefan.schneele@eads.net

Abstract

Für Systementwickler stellt es eine immer größere Herausforderung dar, mit der Komplexität eingebetteter und sicherheitskritischer Echtzeitsystemen umzugehen. Im Rahmen des Projekts „Dependable Embedded Components and Systems“ (DECOS), das im Rahmen des sechsten Rahmenprogramms von der Europäischen Kommission gefördert wird, wird eine globale Methodik und Architektur vorgestellt, die diese Herausforderungen adressiert. Dieser Ansatz wurde für eine prototypische Anwendung aus dem Luftfahrtbereich aufgegriffen und im Zusammenspiel mehrerer Projektpartner erfolgreich umgesetzt. Insbesondere werden die notwendigen Schritte zur Erstellung eines plattform-unabhängigen Modells aufgezeigt, sowie notwendige Iterationen beim Erstellen des Cluster Schedules bedingt durch eine Analyse und Bewertung der eingebetteten Echtzeit-Umgebung und den Rahmenbedingungen des Systemreglers. Das vollständig nach DECOS Paradigmen entwickelte und konfigurierte System, wird in den nächsten Wochen in einem realen Teststand integriert, um diesen Erfahrungsbericht mit dem Schwerpunkt auf Systemdesign mit einer Systemvalidierung abzurunden.

1 Einführung

Computersysteme sind mehr und mehr im Vormarsch in sicherheitskritische Anwendungsgebiete der Industrie. Dies betrifft insbesondere Luft- und Raumfahrt und Automobilbau, aber auch industrielle Produktionsverfahren, medizinische Systeme und ähnliche Bereiche.

Entwickler für sicherheitskritische Systeme müssen sich mit mehreren Thematiken auseinandersetzen: Die Systeme sind häufig *eingebettet* in die Umwelt, z.B. in ein Flug- oder Fahrzeug. Sie erfordern die Interaktion mit physikalischen Prozessen, wodurch das *Echtzeitverhalten* eine wichtige Rolle spielt. Zusätzlich werden Funktionen räumlich auf einzelne Steuergeräte getrennt. Dies geschieht zum einen aus Sicherheitsgründen, um die natürlichen Fehlerbarrieren von physikalisch unterschiedlichen Steuergeräten auszunutzen. Zum anderen werden die Steuergeräte im Gesamtsystem *verteilt*, um räumlich dicht an der Schnittstelle zum physikalischen Prozess zu liegen.

Gerade verteilte Systeme bringen neue Chancen aber auch große Herausforderungen mit sich. Zur Koordination des Gesamtsystems ist Kommunikationsaustausch zwischen den einzelnen Komponenten erforderlich, und der Bedarf nach Nachrichtenverkehr steigt mit wachsenden Abhängigkeiten zwischen den Systemteilen überproportional an. Um den Datenaustausch zuverlässig und deterministisch zu gestalten, werden hoch entwickelte Kommunikationsprotokolle eingesetzt, die den sicheren Nachrichtentransport garantieren sollen.

1.1 DECOS Projekt

Das DECOS (*Dependable Embedded Components and Systems*) **[1]** Projekt ist ein Integrated Project im EU Framework Programme 6. Ziel

des Projektes ist es, einen Prozess zur Entwicklung eingebetteter Systeme hervorzubringen, welche auf Standardkomponenten (COTS, commercial-of-the-shelf) aufbauen, sowohl hardware- als auch softwareseitig. Trotzdem soll gewährleistet werden, dass mit diesen Prozessen und Komponenten sichere Systeme auch für sicherheitskritische Anwendungen entwickelt werden können.
Die Lösung dieses Problems liegt in der Entwicklung von fundamentalen Technologien, die vom Einsatzgebiet unabhängig sind und den Paradigmenwechsel weg vom *föderierten* und hin zum *integrierten* Design von zuverlässigen eingebetteten Echtzeitsystemen ermöglichen.
Hauptziel ist es, ein entsprechendes Rahmenwerk zu untersuchen und eine Reihe von generischen Hardware- und Softwarekomponenten zu entwickeln. Das Projekt versteht sich als vom Einsatzgebiet unabhängig und unterstützt daher diverse Plattformen, die nur gewisse Kerndienste zur Verfügung stellen müssen. Dies sind:

- *Deterministischer und zeitgerechter Nachrichtentransport*,
- *Fehlertolerante Uhrensynchronisation*,
- *Starke Fehlerisolation* und
- *Konsistente Diagnose* von ausfallenden Knoten.

Diese Kerndienste erfüllt insbesondere die Zeitgesteuerte Architektur (*Time-Triggered Architecture*, *TTA*) **[2]**, wobei im DECOS Projekt verschiedene Implementierungen berücksichtigt und beispielhaft eingesetzt werden, namentlich das Zeitgesteuerte Protokoll (*Time-Triggered Protocol, TTP*) **[3]**, *FlexRay* **[4]** und zeitgesteuertes Ethernet. In unserem Teilprojekt wurde TTP gewählt, da es konsequent den Paradigmen und Sicherheitsansprüchen der TTA folgt und technisch ausgereift ist.
Das DECOS Projekt betrachtet zwei grundsätzliche Aspekte: Zum einen wird die technische Umsetzung für solche Systeme untersucht und zum anderen werden konkrete Prozesse zur möglichst einfachen und fehlerfreien Entwicklung der Systeme erstellt, wobei modellbasierte Entwicklung eingesetzt wird.

Technische Umsetzung

Technische Aspekte werden untersucht, um die sichere Integration von mehreren Funktionen pro Steuergerät vornehmen zu können. Klassischerweise wurde pro Steuergerät genau eine Funktion vorgesehen, um bei einem möglichen Funktionsausfall die natürlichen Fehlerbarrieren zu nutzen, die durch die getrennte Hardware gegeben ist. Diese Barrieren müssen in integrierten Systemen durch Softwaremechanismen nachgebildet werden. Sie werden so aufgebaut, dass sicherheitskritische Applikationen zusammen mit nicht-sicherheitskritischen Applikationen auf ein- und demselben Steuergerät ablaufen können, ohne sich gegenseitig zu beeinflussen.
Die grundsätzliche Idee ist, nicht eine komplett neue Architektur zu entwickeln, sondern auf den Basisdiensten der Zeitgesteuerten Architektur aufzubauen. Der Fokus liegt also auf einer hoch entwickelten Zwischenschicht (*Middleware*) zwischen den Basisdiensten und den eigentlichen Applikationen wie in Abb. 1.1 dargestellt. Die Plattformunabhängigkeit gewährleistet, dass eine Portierung einer bestimmten Implementierung der Zeitgesteuerten Architektur zu einer anderen problemlos möglich ist, und erhöht somit deutlich die Widerverwendbarkeit der eigentlichen Anwendung, auch wenn sich die Basistechnologie ändern sollte.
Die technische Umsetzung der Integrationsidee mündet in einer eingekapselten Ausführungsumgebung, der sogenannten *Encapsulated Execution Environment* (*EEE*). Diese wird in einem Betriebssystem für die verteilten Knoten umgesetzt und bietet sowohl räumliche als auch zeitliche Trennung von Applikationen in *Partitionen*. Auf der einen Seite wird hardwareunterstützter Speicherschutz eingesetzt, damit unterschiedliche Partitionen nur über die explizit definierten Schnittstellen miteinander interagieren. Auf der anderen Seite läuft das Betriebssystem rein zeitgesteuert und garantiert damit den Partitionen bestimmte Rechenzeiten, die schon zur Design-Zeit festgelegt werden.

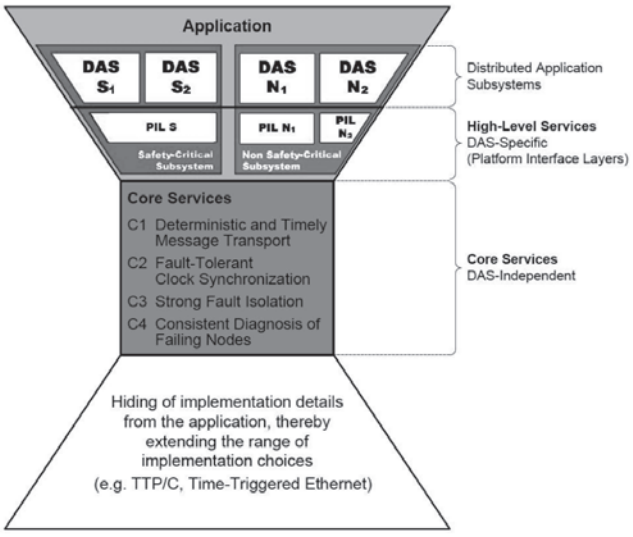

Abb..1.1 Die Diensthierarchie von DECOS

Um die DECOS Architektur in ihren Eigenschaften zu vervollständigen, werden noch weitere Aspekte realisiert: Um die Hauptprozessoren zu entlasten, wird eine Zwischenschicht für Behandlung von Fehlertoleranz in Hardware implementiert, die den Nachrichtentransfer in separate physikalische Kommunikationsleitungen vervielfachen kann und dies beim Empfänger wieder zusammenführt. Zusätzlich können mehrere virtuelle Netzwerke auf einem einzigen physikalischen Netz abgebildet werden. Ein virtuelles Netz bildet logische Verbindungen zwischen beliebigen Teilmengen von verteilten Knoten und insbesondere auch übergreifend zwischen eigenständigen Netzen, die nur durch Gateways miteinander verbunden sind. Die konkrete Netzwerktopologie soll hierbei für die Anwendung unsichtbar sein und erst in einem sehr späten Entwicklungsschritt festgelegt werden, was auch die Wiederverwendbarkeit erneut berücksichtigt. Die virtuellen Verbindungen können sogar von ereignisgesteuertem Charakter sein, die letztendlich jedoch über das zeitgesteuerte Medium übertragen werden. Das kann helfen, alte Programme, die beispielsweise für den CAN-Bus ausgelegt sind, in die DECOS Architektur zu integrieren. Desweiteren hilft ein integriertes Diagnose-System in der DECOS Architektur kontinuierlich Statusinformationen über den Nachrichtentransport und den Systemzustand zu ermitteln.

Die oben zusammengefasste technische Umsetzung soll die Machbarkeit einer solchen Architektur an sich demonstrieren. Parallel dazu wird ein Prozess entwickelt, der die Benutzbarkeit durch Entwickler im Vordergrund sieht. Wichtige Aspekte dabei sind Zertifizierbarkeit, Wiederverwendbarkeit und Entlastung der Entwickler an möglichst vielen Stellen durch Automatisierung und die Unterstützung durch Werkzeuge. Dafür wird eine komplette Werkzeugkette entwickelt, welche einen nahtlosen Entwicklungsprozess gewährleisten soll. Diese Werkzeugkette setzt in hohem Maße auf modellbasierte Systementwicklung.
Der Prozess ist dabei weitestgehend zweigeteilt. Der eine Teil beschäftigt sich mit der eigentlichen funktionalen Softwareentwicklung und setzt die in den Branchen üblichen Werkzeugketten Matlab/Simulink/Stateflow von Mathworks **[5]** und SCADE von Esterel Technologies ein und verknüpft diese mit der restlichen DECOS Werkzeugkette. Dieser Teil wird hier nur am Rande betrachtet.

Der andere Teil der Werkzeugkette betrifft die Hardware-Software-Integration und umfasst den gesamten Prozess angefangen bei der Aufnahme der Systemanforderungen über die Entwicklung von Konfigurationsmodellen und die Verarbeitung durch automatische Konfigurationswerkzeuge. Dies sind beispielsweise Scheduler für die zeitgesteuerten Betriebssysteme und die Kommunikation. Dies geht bis hin zur Verteilung der fertigen Anwendungs-Binärdateien auf die verteilten Knoten.
Dieser Teil der Werkzeugkette beschreibt das System in stereotypierten UML Klassendiagrammen. Dabei wird eine grundsätzliche Aufteilung vorgenommen: Der Entwickler definiert zunächst ein plattformunabhängiges Modell (*Platform Independenet Model, PIM*), welches nur anwendungsspezifische Konfigurationsdaten des Systems enthält, aber auf Architekturdetails explizit verzichtet. Die funktionale Software kann im nächsten Schritt dann schon entwickelt werden, während plattformabhängige Einstellungen in der Werkzeugkette erst in einem recht späten Entwicklungsschritt hinzugefügt werden in das Plattform Spezifische Modell (*Platform Specific Model, PSM*). Erst das PSM enthält dann beispielsweise Informationen über die Verteilung von Software-Einheiten (Jobs) auf konkrete Steuergeräte und die errechneten Schedules. Diese Aufteilung trennt Softwareentwicklung von der Plattformkonfiguration und erleichtert somit die Migration von einer Plattform auf eine andere und ermöglicht Validierung der Software über Simulationen und formale Methoden (z.B. Model-Checking **[7]**) in einem frühen Entwicklungsstadium, unabhängig von der konkreten Verteilung.

Luft- und Raumfahrt Teilprojekt
Das DECOS Projekt sieht neben der oben zusammengefassten Architektur die Entwicklung von industriellen Demonstratoren vor, bei denen die DECOS Architektur und die Prozesse eingesetzt und validiert werden sollen.
Die Autoren selbst arbeiten an einem solchen Demonstrator für die Luft- und Raumfahrtindustrie. Es soll gezeigt werden, dass die DECOS Architektur auch bei höchst sicherheitskritischen Anwendungen einsetzbar ist. Umgesetzt wird ein elektronisch synchronisiertes Hochauftriebssystem. Ein solches Landeklappensystem erhöht den Auftrieb der Flügel und ermöglicht somit Start und Landung bei verhältnismäßig niedrigen Geschwindigkeiten. Ein Ausfall des Systems hätte zur Folge, dass sich hauptsächlich die Landestrecke aufgrund der notwendigerweise höhe-

ren Anflug- und Landegeschwindigkeiten deutlich verlängern würde, zusätzlich zu einer wesentlich höheren Belastung des Fahrwerks. Ein ungewollter asynchroner Zustand beider Flügelseiten erzeugt ein Rollmoment, das unter Umständen nicht mehr durch die primäre Flugsteuerung kompensiert werden kann und im schlimmsten Fall zum Absturz führt. Insofern ist das System höchst sicherheitskritisch und wurde wie in Abb.1.2 gezeigt bisher rein mechanisch synchronisiert **[8]**: Eine starre Welle durch den Rumpf verbindet mechanisch die Antriebsstationen der Landeklappen beider Flügel. Die fehlende Modularität und Flexibilität dieses Ansatzes lassen eine rein elektronisch synchronisierte Lösung sehr vorteilhaft erscheinen. Eine ähnliche Sicherheit wie bei der mechanischen Synchronisierung zu erreichen, stellt aber eine große Herausforderung dar.

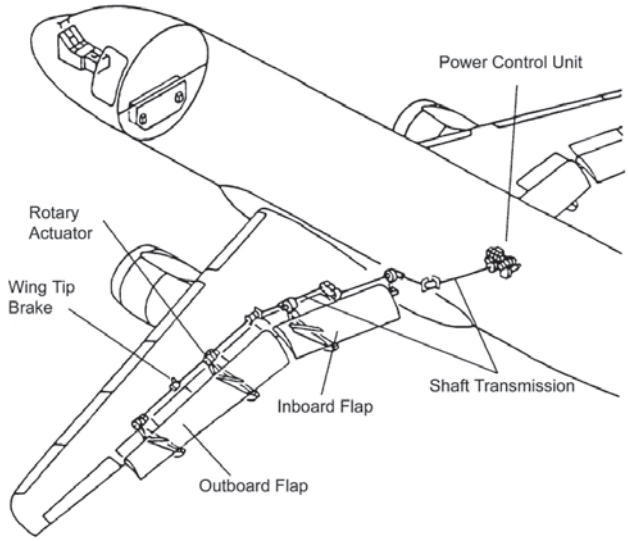

Abb.1.2: Mechanisch synchronisiertes Hochauftriebssystem[8]

Die Architektur des Demonstrators ist in Abb. 1.3 dargestellt. Ein physikalischer Test-Stand beinhaltet alle relevanten mechanischen und elektronischen Komponenten eines einzelnen Landeklappen-Segments: Zwei Elektromotoren treiben über eine Welle verbunden die Landeklappe an. Gesteuert werden die Motoren über Motorelektroniken (*Motor Control Electronics, MCE*) und eine elektromechanische Bremse (*Cross Shaft Brake, CSB*) ist in der Lage, das System stillzulegen. Die eigentlichen Regelkreise bilden Sensoreinheiten (*Position Pick-Off Unit, PPU*) und Regler, die die Motorelektroniken ansteuern (*Actuator Control Electronics, ACE*). Eine zentrale Steuereinheit (*System Control Electronics, SCU*) regelt und überwacht das Gesamtsystem und stellt über den ereignisgesteuerten Bus AFDX (*Avionics Full Duplex Switched Ethernet*) Verbindungen zu anderen Systemen her, die hier durch eine rudimentäre Flugzeugsimulation auf einem PC ersetzt werden. Die äußeren Regelkreise sind über den zeitgesteuerten Bus der DECOS Architektur miteinander verbunden. Der Test-Stand besteht aus nur einem Landeklappen-Segment. Ein weiteres Segment zur Synchronisierung wird durch eine Echtzeit-Simulation in Matlab/Simulink auf einem dSpace-System hinzugefügt.

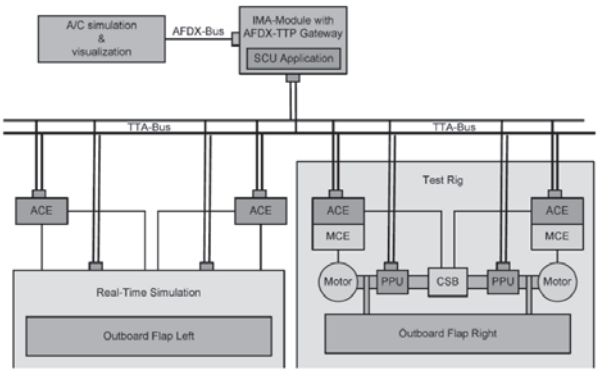

Abb.1.3: Architektur des Demonstrators

Im Folgenden berichten wir von Erfahrungen bei der Entwicklung und Konfiguration des Systems, insbesondere bei der HW-SW-Integration.

2 Partitionierung und WCET Analyse

2.1 Das Encapsulated Execution Environement

Das in Kapitel 1 eingeführte Encapsulated Execution Environment (EEE) ermöglicht die Kapselung von Partitionen – auch als Jobs bezeichnet - von unterschiedlicher Kritikalität, und deren Betrieb auf derselben Hardware Plattform. Diese Partitionierung erfolgt durch Zugriffschutz von Ressourcen (Speicher, Peripherie, etc.) und durch eine a priori festgelegte zeitliche Abfolge. Neben den Partitionen der jeweiligen Anwendungen gibt es drei obligatorische Jobs, die Grundfunktionalitäten für die DECOS Architektur zur Verfügung stellen:

- SA
- Sync
- CCCP

Es handelt sich dabei um den so genannte Sensor-Aktuator Job, die Communication Controller

Control Partition und um die Synchronisations-Partition. Der Sensor-Aktuator Job verwaltet alle lesenden und schreibenden Zugriffe auf die Peripherie durch dedizierte Treiber. Dazu gehören nicht nur analoge und digitale IOs der Zielplattform, sondern auch andere Kommunikationssysteme wie z.B. der CAN-Bus. Eine Inter-Partitions-Kommunikation, entweder als so genannter Kommunikationskanal oder als shared memory implementiert, ermöglicht das Austauschen von Daten zwischen den einzelnen Jobs. Der Synchronisations-Job führt die notwendige Uhrensynchronisation zwischen den globalen und lokalen Uhren durch und sollte abhängig von den Drift Raten und der notwendigen zu erreichenden Genauigkeit aufgerufen werden. Bedingt durch diese sicherheits- und funktionsrelevanten Jobs, erzeugt dies selbstverständlich einen Overhead, den es bei der Systementwicklung zu berücksichtigen gilt. Im nächsten Absatz werden an dem Beispiel der ACE und PPU Entwicklung mögliche Konfigurationen eingeführt. Dies ist ein essentieller Input für die spätere Entwicklung des Plattform Independent Models (PIM). Um den Overhead für diese Anwendungen zu bestimmen, benötigt man die Aktivierungs- und Deaktivierungszeit von Partitionen, sowie die Zugriffszeit für Lesen und Schreiben durch die Inter-Partitions-Kommunikation. Diese Werte können der folgenden Tabelle entnommen werden.

- T_{act_part} = 66 µs
- T_{deact_part} = 64 µs
- T_{lesen} = 32 µs
- $T_{schreiben}$ = 29 µs

2.2 Partitionierung der ACE

Die Actuator Control Electronik (ACE) hat einen funktionalen Job, später als ACE_FUNC bezeichnet. Bevor diese Partition aktiviert werden kann, müssen Werte von der Motor-Kontroll-Einheit (MCE) gelesen werden durch einen Sensor-Aktuator Job (ACE_SA). Danach werden Soll- und Istwerte anderer Knoten via LTTP empfangen (ACE_CCCP). Nach Beendigung der Kalkulationen (ACE_MFUNC) werden die Werte über eine CAN-Schnittstelle (ACE_SE) an die Motor-Kontroll-Einheit (MCE) gesendet, sowie auch über den LTTP-Bus übertragen (ACE_CCCP). Dies ergibt - mit den bereits erwähnten obligatorischen Partitionen - folgende Konfiguration:

- ACE_SA
- ACE_CCCP
- ACE_FUNC
- ACE_SA
- ACE_CCCP
- ACE_SYNC

Zieht man die oben erwähnten Parameter in Betracht, berechnet sich für diese Anwendung ein Overhead von **963** µs. Die Ausführungszeiten für die einzelnen Partitionen sind hier selbstverständlich noch nicht berücksichtigt.

2.3 Partitionierung der PPU

Die Position Pick-Off Unit besteht aus einem funktionalen Job (PPU_FUNC) und einem Job für eine Build-In Test Funktionalität (PPU_BITE). Eine Sequenz beginnt mit dem Auslesen der Sensoren (PPU_SA) und der Auswertung der Messwerte (PPU_FUNC). Danach wird durch den Build-In Test die Korrektheit der Kalkulation, sowie der ordnungsgemäße Zustand der Sensoren überprüft. Das Ergebnis wird über LTTP an andere Knoten übertragen (PPU_CCCP). Dies ergibt folgenden Ablauf:

- PPU_SA
- PPU_FUNC
- PPU_BITE
- ACE_CCCP
- ACE_SYNC

Diesmal beläuft sich der Overhead auf **833** µs. Das folgende Bild zeigt den Zeitverbrauch exklusive der Ausführungszeit der einzelnen Jobs.

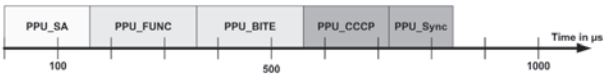

Abb.2.1:Overhead der PPU

Hier zeigt sich eine Problematik des statischen Schedulings. In einer frühen Designphase müssen Nachrichten-, Task-, und Partitionsschedules definiert werden, die nicht nur konsistent sein müssen, sondern auch funktionalen Gesichtspunkten genügen müssen. Dies sind vor allem die Widerholungsrate (Abtastzeit) von Signalen – essentiell für den Systemregler – und die Ausführungszeit der Software. Mit den oben aufgezeigten Berechnungen lässt sich deutlich aufzeigen, dass eine Wiederholungsrate um *1 ms* nicht zu erreichen sein wird, da bereits der Overhead in dieser Größenordnung liegt. Im nächsten Kapitel wird ein Ansatz eingeführt, um worst case

Laufzeiten abschätzen zu können, ohne eine komplette Integration durchzuführen. Darüber hinaus werden die notwendigen Timing Constraints des Reglers hergeleitet, um im Vorfeld eine Abschätzung über die Realisierungsmöglichkeiten des Gesamtsystems abzugeben.

2.4 WCET Analyse

Die Bestimmung der Ausführungszeit von Programmen unter den schlechtesten Rahmenbedingungen, die sogennante *Worst Case Execution Time (WCET)*, ist ein komplexes akademisches Problem, dem sich viele Forschungsinstitute und einige wenige kommerzielle Unternehmen verschrieben haben. Gleichwohl wird aber die Angabe von maximalen Ausführungszeiten als Voraussetzung in der zeitgesteuerten Architektur für den Task Schedule und dem eng damit verbundenen Message Schedule benötigt. Eine Abschätzung durch Laufzeitmessung kann erste Anhaltspunkte geben. Dies setzt allerdings das Vorhandensein der späteren Zielhardware voraus und gibt darüber hinaus keine Garantien, ob der gewählte Input wirklich dem worst case entsprach. Statische Analysen, bzw. hybride Ansätze scheinen viel versprechende Aussagen zu liefern [11]. Dabei ist zu beachten, dass das Ergebnis der Analysen sicher genug ist, das heißt eine Sicherheitstoleranz ein Überschreiten der Deadline unmöglich macht. Allerdings sollte das Ergebnis auch „eng“ genug sein, da zu hoch geschätzte Ausführungszeiten entweder mit aufwendiger Hardware beantwortet werden, oder eben in einem ineffizienten System enden.
Im Folgenden wird aus industrieller Sicht die tool-unterstütze WCET Analyse basierend auf dem C-Code und dem Objekt-Code vorgestellt. Die dabei verwendeten Tools sind **„Calc_Wect_167”** [12] von der Universität Wien, und **„aiT“** [13] von der Firma Absinth. Diese Tools werden beispielhaft am Code der ACE angewendet. Der Worst Case Aufrufbaum ist in Abbildung 2.2 dargestellt.

Abb.2.2: Worst Case Aufrufbaum der ACE

Nun gilt es die Ausführungszeiten der einzelnen Funktionen mit **„Calc_Wect_167”** zu bestimmen.

In Tabelle 2.1 ist das Ergebnis der ACE Quellcode Analyse aufgelistet.

Nr.	C Files	Nr.	Output WCET Files	WCET CPU Cycles
1	ACE	1	ACE	28460 incomplete
		2	ACE_init	2500 incomplete
2	ACE_const			0
3	ACE_control	3	ACE_control	25340 incomplete
		4	ACE_control_init	3460 incomplete
4	ACE_frequency_monitor	5	ACE_frequency_monitor	10260 incomplete
		6	ACE_frequency_monitor_init	2380 incomplete
5	ACE_monitor	7	ACE_monitor	21120 incomplete
		8	ACE_monitor_init	2380 incomplete
6	ACE_status_mce	9	ACE_status_mce	7700
		10	ACE_status_mce_init	1180
7	BoolToInt	11	BoolToInt	2840
		12	BoolToInt_init	820
8	ClockCounter	13	ClockCounter	5180
		14	ClockCounter_init	1180
9	FlipFlopReset	15	FlipFlopReset	5800
		16	FlipFlopReset_init	1180
10	Gain	17	Gain	4860 incomplete
		18	Gain_init	820
11	LimiterUnSymmetrical_0	19	LimiterUnSymmetrical_0	5300
		20	LimiterUnSymmetrical_0_init	820

Tab.2.3: Worst Case Execution Time (High Level)

Wie man deutlich der Tabelle entnehmen kann, wird die WCET in Clock Cylces angegeben. Allerdings können nicht alle Funktionen komplett analysiert werden („incomplete“) auf Grund von Einschränkungen dieses Tools. Es werden Minimalwerte angegeben, die durchaus eine erste Abschätzung der WCET erlauben, gleichwohl nur der Prozessor C167 unterstützt wird, und nicht der TC1796, der in diesem Projekt zum Einsatz kommt.
Das kommerzielle Tool **„aiT“** liefert basierend auf komplexen Prozessormodellen absolute Aussagen. Ein Beispiel kann Tabelle 2.4 entnommen werden.

Nr.	Test Setting	WCET CPU Cycles	WCET Time
1	Clock: 150 Mhz; Compiler: "ppc-diabdata"; interproc flexible; loop "_f_div" + 1: >=1, <= 50;	9450	63 us
2	Clock: 150 Mhz; Compiler: "ppc-diabdata"; interproc flexible; loop "_f_div" + 1: >=0, <= 1000;	9460	63 us

3	Clock: 150 Mhz;
	Compiler: "ppc-diabdata";
	interproc flexible, max-length=5, default-unroll=4, max-unroll=4;
	loop "_f_div" + 1: >=0, <= 1000;
		59364	0.39 ms

Tab.2.4:Worst Case Execution Time (Low Level)

Auch hier war zum Zeitpunkt der Analysen noch keine Unterstützung des TC1796 verfügbar, so dass auch der C167 gewählt wurde. Allerdings lassen die ermittelten Zahlen die Schlussfolgerung zu, dass sowohl der ACE als auch der PPU Applikations-Code Ausführungszeiten deutlich unter *200 µs* hat, und damit in etwa 20 % des Overheads der DECOS Architektur entspricht. Als Cluster Cycle dürfte *1,5 ms* möglich sein, *2 ms* werden sicher erreicht werden unter den oben genanten Rahmenbedingungen.

3 Anpassung der Cycle Time

Die schärfsten Anforderungen an die Cycle Time des verwendeten Datenbusses stellt, wie bereits angesprochen, die notwendige Abtastrate für die implementierte Synchronisationsregelung. Ziel dieser Synchronisationsregelung ist zum einen der möglichst exakte Gleichlauf beider Antriebe einer Landeklappe, um eine unnötige Verspannung der Wellentransmission und dadurch einen so genannten „force fight" zu vermeiden sowie zum anderen eine Synchronisation der Antriebe an verschiedenen Landeklappen, um ein Rollmoment um die Flugzeuglängsachse zu verhindern. Erreicht wird diese Synchronisation mittels einer Positionsregelung mit Geschwindigkeitsvorsteuerung, wie sie schematisch in Abbildung 3.1 dargestellt ist.

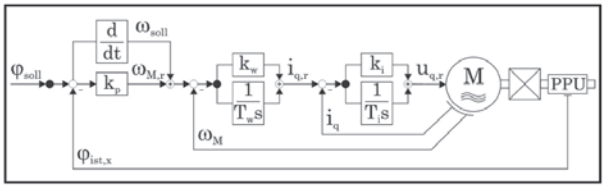

Abb. 3.1: Signalflussgraph des positionsgeregelten Antriebs

Man erkennt zunächst drei Reglerkaskaden, die im Folgenden kurz vorgestellt werden. Die innerste Schleife (Stromregler) ist in der Regel fest vom Motorenhersteller vorgegeben und lässt sich nicht verändern.
Die Drehzahlregelung ist ebenfalls in die Umrichter (MCE) integriert und nutzt motorinterne Sensorik. Sie sollte für diese Anwendung so eingestellt sein, dass die Sprungantwort möglichst geringes Überschwingen bei kurzer Einschwingdauer aufweist.
Der äußerste Regelkreis (Positionsregelung) wird nun über ein spielbehaftetes Getriebe, einen absoluten Drehwinkelgeber (PPU), den Datenbus sowie die ACE, in der die eigentliche Regelung abläuft, geschlossen. Neben der Proportionalverstärkung der Positionsdifferenz k_p ist die Übertragungsrate des Busses ein entscheidender Parameter, den es im Sinne eines akzeptablen Systemverhaltens sinnvoll zu wählen gilt.

Neben den klassischen regelungstechnischen Anforderungen an das Systemverhalten (Stabilität, Störkompensation und Sollwertfolge) ergeben sich für diesen speziellen Anwendungsfall weitere Anforderungen:

- Die Abtastrate muss das so genannte SHANNON-Kriterium erfüllen, so dass eine fehlerhafte Abtastung durch *Aliasing* verhindert wird.
- Das Systemverhalten sollte möglichst robust sein gegenüber Parameteränderungen wie zum Beispiel eine Vergrößerung des mechanischen Spiels durch längeren Gebrauch, temperaturbedingte Änderungen der Reibverhältnisse, etc.
- Die Synchronität der Antriebe sollte unter allen Randbedingungen (zum Beispiel unterschiedliche Luftlasten) gewährleistet sein.

Untersucht man für das vollständige System des elektromechanischen Aktuators nun, bei welchem Wert des P-Anteils im Positionsregler der Regelkreis instabil wird (kritische Verstärkung $k_{p,krit}$), so ergibt sich für $\Delta T_\varphi \in [0.5\ ms; 0.1\ s]$ der in Abbildung 3.2 dargestellt Verlauf.
Zusätzlich wurde die maximal auftretende Frequenz im Positionssignal bestimmt; sie entspricht der Eigenfrequenz des Getriebes, so dass nach SHANNON daraus die maximale Abtastzeit für das betrachtete System bestimmt werden kann: $\Delta T_{\varphi,max} = 4.4\ ms$.

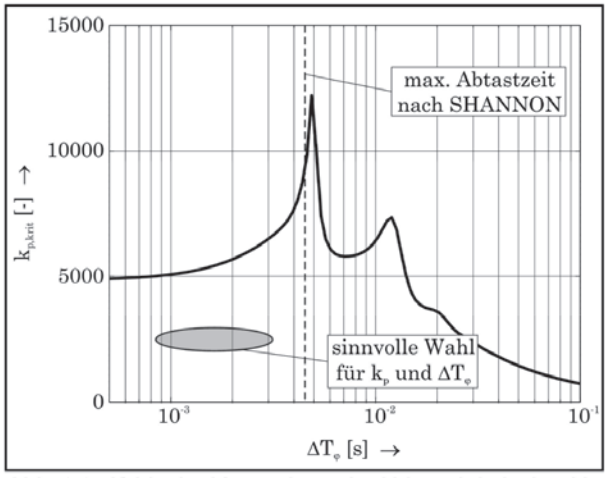

Abb. 3.2: Kritische Verstärkung in Abhängigkeit der Abtastrate

Man erkennt zunächst für schnelle Abtastungen ($\Delta T_\varphi \rightarrow 0$) eine Annäherung an den Wert der maximalen Verstärkung des kontinuierlichen Systems, der den wählbaren Bereich für k_p und ΔT_φ nach oben hin begrenzt. Nach rechts wird dieser Bereich durch die bereits bestimmte maximale Abtastzeit begrenzt.

Um nun einen guten Kompromiss zwischen einem möglichst robusten Systemverhalten einerseits und guten dynamischen Eigenschaften andererseits zu finden, wurde im ersten Schritt die Verstärkung zu k_p = 2500 und die Abtastzeit zu ΔT_φ = 1 ms gewählt. Da letztere sich aus bereits genannten Gründen jedoch nicht realisieren lässt, wurde sie daraufhin auf ihren endgültigen Wert von ΔT_φ = 2 ms erhöht. Wie in Abbildung 3.2 zu erkennen, ist dies durchaus noch möglich, jedoch muss auch hier schon ein weniger robustes Systemverhalten in Kauf genommen werden.

Eine weitere Vergrößerung von ΔT_φ, auch über die hier angegebene maximale Abtastzeit hinaus, lässt sich dann jedoch nur noch unter der Annahme einer ausreichend gedämpften Eigenbewegung des Getriebes und unter Verwendung eines Anti-Aliasing-Filters sowie einer eventuellen Reduktion des Verstärkungsfaktors realisieren.

4 PIM Erstellung

Das PIM (Platform Independent Model) ist der zentrale Einstiegspunkt, um ein DECOS System zu erstellen. Für das PIM werden die Aufgaben, die die Software zu erledigen hat, auf verschiedene Jobs aufgeteilt, die über Messages miteinander kommunizieren. Diese Jobs werden dann als Elemente im PIM definiert. Um dies sinnvoll erledigen zu können, muss sich der Ersteller natürlich im Vorfeld bereits genügend Gedanken über den logischen Aufbau der Software gemacht haben, und auch eine klare Vorstellung davon haben, welche Interfaces die einzelnen Knoten des DECOS Systems haben und welche Funktionen durch sie erfüllt werden sollen.
Jobs sind deshalb die einzelnen funktionalen Einheiten der Software. Die anderen wichtigen Elemente sind die Messages. Messages werden zwischen den Jobs hin- und hergesendet. Um nun die Messages bestimmten Jobs zuzuordnen und ihre Eigenschaften festzulegen, gibt es noch eine Vielzahl andere Elemente, mit denen Jobs und Messages verbunden werden. Verbindungen zwischen zwei Elementen werden Relations genannt. Einen Überblick über die Jobs und Messages in einem PPU-Knoten kann man in Abb. 4.1 sehen.
Um nun eine Message einem Job zuordnen zu können, muss für diesen Job ein Interface erzeugt werden und dieses Interface mit dem Job verbunden werden. Für das Interface wird nun ein Port erzeugt, der mit dem Interface verbunden wird. Erst jetzt kommt die Message ins Spiel. Sie wird nun mit dem Port verbunden. Nun benötigt man noch eine State Variable, in der der Message Inhalt abgebildet wird, und diese Variable wird jetzt mit dem Job verbunden, dem Port und der Message. Jedem dieser Elemente werden nun bestimmte Parameter zugeordnet, die angeben, in welcher Art und Weise der Message-Austausch stattfindet.

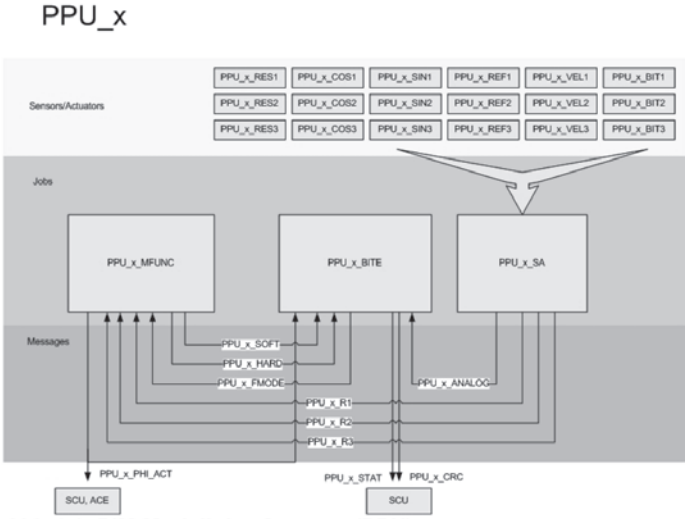

Abb.4.1: PIM Modell des Sensors (PPU)

Von BUTE, dem DECOS Partner für das PIM, wurden zwei Möglichkeiten das PIM zu erstellen entwickelt. Zum einen kann ein Standard UML Editor, wie z.B. das Tool Rational Rose verwendet werden, in das ein Metamodel geladen wird, welches sämtliche PIM-Elemente enthält; zum anderen kann das von BUTE entwickelte Eclipse Plugin DSE (Domain Specific Editor) verwendet werden.

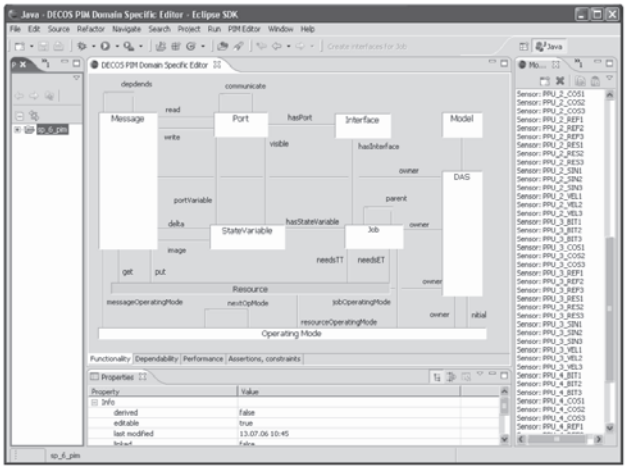

Abb.4.2 DSE Domain Specific Editor

Modellieren der PIM mit einem standard UML Editor läuft auf die manuelle Erstellung von stereotypierten UML Klassendiagrammen hinaus, die sehr viele Elemente enthalten. Ohne weitere Hilfe durch beispielsweise Pattern bei der Modellierung [10] ist diese Methode sehr zeitaufwändig und fehleranfällig.
Im Subproject 6 (Aerospace) von DECOS (SP6) wurde daher der DSE verwendet. Anfangs ließ sich der Editor recht gut bedienen, als jedoch das Modell immer größer wurde, stellte sich heraus, dass auch dieser Editor recht unhandlich zu bedienen ist. Insbesondere Relations im PIM zu erstellen war sehr mühsam. Über Scrollbars müssen Elemente ausgewählt werden, in denen immer nur 5 von teilweise an die 200 aufgelistet sind. Fehler schleichen sich dabei sehr schnell ein. Beim Erstellen des PIM wurde zudem festgestellt, dass die Anzahl der Elemente (und damit auch der Relations) nahezu exponentiell zunahm.
Das System von SP6 besteht aus 9 DECOS-Knoten (4 x ACE, 4 x PPU, 1 x SCU). Für jeweils eine PPU wurden 3 Jobs definiert; für jeweils eine ACE wurden ebenfalls 3 Jobs (2 für die ACE selbst und einer für die Kommunikation mit der MCE) definiert. Für die SCU wurden 2 Jobs definiert (SCU selbst und RAFS). Das ergibt zusammen 26 Jobs. Für diese Jobs waren dann schon ca. 50 Interfaces nötig, ca. 100 Ports, ca. 200 Messages und 400 State Variables. Mit allen anderen noch notwendigen Elementen enthält dieses PIM jetzt ca. 1000 Elemente und etwa doppelt so viele Relations.
Bei dieser Menge von Information muss von Seiten BUTE die Handhabung noch wesentlich verbessert werden. Vor allem muss ein effizientes Errorchecking verwirklicht werden, und eine Gruppierung der Elemente sollte eingeführt werden, um Elemente die logisch zusammengehören, auch zusammen abbilden zu können.
Im DECOS-Prozess soll nun aus dem PIM und weiteren Informationen, die aus der CRD (Cluster Resource Discription) und dem PI (Platform Interface) kommen, automatisch ein Schedule für den Cluster erstellt werden. Da zurzeit diese Toolkette noch nicht vollständig und funktionsfähig ist, wurden die Information des PIMs von Hand in TTPPlan und TTPBuild übertragen (TTPPlan und TTPBuild sind die Tools der Firma TTTech, mit denen das Schedule für den DECOS Cluster und die Konfiguration für das DECOS Betriebssystem EEE erstellt wird). Mit TTPPlan wurde dann ein Schedule erstellt. In TTPPlan werden alle Messages eingebunden, die zwischen den Knoten gesendet werden.

Abb.4.3: DECOS-TTP Schedule

Messages zwischen zwei Jobs in einem Knoten tauchen hier nicht auf. Diese Messages werden in TTPBuild definiert. Für jeden PIM-Job wird ein Subsystem erzeugt. Innerhalb dieses Subsystems können nun verschiedene Tasks definiert werden, in die der Applikationscode eingefügt wird.

5 Schlussfolgerungen

Die Komplexität eingebetteter und sicherheitskritischer Systeme wurde deutlich in den vorangegangenen Kapiteln zum Ausdruck gebracht. Selbst wenn es sich bei dem vorgestelltem System nur um einen Prototypen mit nur wenigen Knoten handelt, zeigt dieses Beispiel deutlich die Grenzen konventioneller Systementwicklung auf, und adressiert die eindringliche Notwendigkeit für globale architektur-bezogene und plattformunabhängige Ansätze, wie sie in DECOS vorgestellt werden. Die Methoden, die innerhalb von DECOS in den letzten zwei Jahren entwickelt wurden, kamen bereits in dieser Anwendung zum Einsatz. Allerdings ist bei einer solchen Herangehensweise eine umfangreiche Unterstützung durch Tools und Prozesse notwendig, um die Menge der Konfigurationsdateien und Schritte sinnvoll verwalten zu können. Hier besteht – in einem Forschungsprojekt durchaus normaler – Bedarf zur Nachbesserung, wie in [10] bereits erkannt wurde. Inwieweit eine spätere Skalierbarkeit ermöglicht werden kann, sollte noch nachgewiesen werden. Die endgültige physikalische Integration wird in den nächsten Wochen stattfinden. Hier wird sich zeigen, ob der a priori investierte, durchaus zeitaufwendige Design-Aufwand den Integrationsaufwand - wie postuliert - deutlich reduzieren wird. Eine durchgehende plattform-unabhängige Softwareentwick-

lung wird nicht möglich sein, da Sachzwänge hervorgerufen durch bestimmte Hardware und der Einsatzumgebung, wie z.B. der Bedarf an langen Kabellängen, essentielle Systemparameter wie in diesem Fall die Bit-Rate (10 MBit/s) beeinflussen, und somit eine Umsetzbarkeit nicht losgelöst von physikalischen Parametern bewertet werden kann. Darüber hinaus sollte die vorgestellte DECOS Architektur im Vergleich zu bereits existierenden Ansätzen wie z.B. der Integrierten Modularen Avionik (IMA) bewertet werden. Für einfache Sensor-Anwendungen scheint die Kombination aus einem sehr leistungsfähigem µ-Controller und einem FPGA überdimensioniert, insbesondere wenn man den Overhead durch das Encapsulated Execution Environment ebenfalls berücksichtigt. Eine abschließende Bewertung der DECOS Architektur im Rahmen dieser Luftfahrtanwendung wird erst nach erfolgter Systemvalidierung erfolgen. Die Fähigkeit zur Unterstützung bei der Zertifizierung komplexer eingebetteter Systeme wird ein besonderer Schwerpunkt sein, da dies einen essentiellen Baustein im Entwicklungsprozess darstellt.
Die praktische Machbarkeit – mit den oben genannten Einschränkungen – des neu entwickelten DECOS Entwicklungsprozesses wurde im Zusammenspiel mehrer Projektpartner erfolgreich durchgeführt, so dass alle notwendigen Hardware- und Softwarekomponenten, und Konfigurationsdateien zur Verfügung stehen, um eine Gesamtsystemintegration in den nächsten Wochen durchzuführen.

6 Danksagungen

Die Autoren möchten sich bei Jörn Rennhack und Jens Koch von Airbus Deutschland GmbH für die Zusammenarbeit in DECOS bedanken, sowie bei Christophe Mital von der TTTech AG und Dr. Gyorgy Csertan von der Universität Budapest, und allen anderen Projektpartnern und Lili Tan von der Universität Duisburg-Essen für die vielen fruchtbaren Diskussionen und Anregungen.

Literatur

[1] DECOS - Dependable Components and Systems, Research project homepage, https://www.decos.at/.

[2] H. Kopetz and G. Bauer, The time-triggered architecture. Proceedings of the IEEE, vol. 91, no. 1, pp. 112-126, 2003.

[3] H. Kopetz and G. Grünsteidl, TTP - a time-triggered protocol for fault-tolerant real-time systems, Institut für Technische Informatik, Technische Universität Wien, Treilstr. 3/182/1, A-1040 Vienna, Austria, Tech. Rep., 1992.

[4] R. Belschner, R. Mores, G. Hay, J. Berwanger, C. Ebner, S. Fluhrer, E. Fuchs, B. Hedenetz, A. Krüger, P. Lohrmann, D. Millinger, M. Peller, J. Ruh, A. Schedl, and M. Sprachmann, FlexRay: The communication system for advanced automotive control systems, in SAE 2001 World Congress. Detroit, USA: Society of automotive Engineers, Mar. 2001.

[5] Mathworks Inc., Simulink - Simulation and Model-Based Design, Mathworks Inc., 2005, http://www.mathworks.com/access/helpdesk/help/pdf_doc/simulink/sl_using.pdf

[6] Esterel Technologies, Company homepage, http://www.esterel-technologies.com.

[7] A. Bouali, B. Dion, and K. Konishi, Using formal verification in real-time embedded software, in JSAE Annual Congress, 2005.

[8] T. Neuheuser, B. Holert, and U. B. Carl, Elektrische Antriebssysteme für ein zentrales Landeklappenelement, in Deutscher Luft- und Raumfahrtkongress, vol. DGLR-JT 2002-192, Stuttgart, 2002.

[9] ARINC 664, Aircraft Data Networks, Part 7 Deterministic Networks, ARINC, Annapolis, Maryland, USA, http://www.arinc.com

[10] W. Herzner, G. Csertan, A. Balogh, Design Patterns for Domain-specific Application Modelling, DECOS/ERCIM Workshop on Dependable Embedded Systems, Cavtat/Dubrovnik, Croatia, August 29, 2006

[11] J. Engblom, Processor Pipelines and Static Worst-Case Execution Time Analysis http://www.coronetbooks.com/books/proc2280.htm, Last access on June 14, 2006.

[12] R. Kirner, The Program Language WCET, 2001.

[13] AbsInt, http://www.absint.com/ait/features.htm, Last access on June 28, 2006.

Hauke Fuhrmann studierte Informatik an der Christian-Albrechts-Universität zu Kiel, wo er 2005 mit dem Diplom abschloss, bereits in dem EU Projekt DECOS involviert. Seit Mitte-2005 ist er wissenschaftlicher Angestellter und Doktorand der Forschungsgruppe für Echtzeitsysteme und Eingebettete Systeme an der CAU Kiel. Dort betreut er praktische Kurse über modellbasierte Systementwicklung verteilter Echtzeitsysteme und arbeitet an Systemmodellierung und Validierung im DECOS Projekt.

Lothar Klein studierte Elektrische Nachrichtentechnik an der Fachhochschule Konstanz, wo er 1997 mit dem Diplom (FH) abschloss. Seit 2002 ist er bei Liebherr Aerospace in Lindenberg als Software Entwickler beschäftigt. Dort befasst er sich hauptsächlich mit Ansteuerungen und Regelungen diverser Hydrauliksysteme im Bahnbereich. Seit Mitte 2005 arbeitet er an der Motoransteuererung im DECOS Project

Hendrik Geilsdorf ist seit Januar 2004 wissenschaftlicher Mitarbeiter im Institut für Flugzeug-Systemtechnik an der Technischen Universität Hamburg-Harburg. Sein Hauptarbeitsgebiet ist der Entwurf und die Analyse neuartiger Konzepte für Hochauftriebssysteme großer Transportflugzeuge, hauptsächlich in Bezug auf Regelung und Systemüberwachung.

Stefan Schneele studierte Medizinische Informatik, und ist seit Februar 2004 Doktorand in der Abteilung Sensor Systeme und Integration des Corporate Research Centre der EADS in München. Seit Arbeitsschwerpunkt liegt in dem Bereich fehlertoleranter Bussysteme für Luftfahrtanwendungen. Im Rahmen von DECOS arbeitet er an der Entwicklung und Integration eines smarten Positionssensors.