Frühzeitiges Absichern von Entwurfsentscheidungen durch Virtuelle Deployments

Thomas Kuhn

Embedded Software Development
Fraunhofer IESE
Fraunhofer-Platz 1
67663 Kaiserslautern
thomas.kuhn@iese.fraunhofer.de

Abstract: Multi-Core ECUs ermöglichen eine Vielzahl neuartiger Software- und Systemarchitekturen, fordern Architekten jedoch auch viele grundlegende Entscheidungen ab, die oft weichenstellend für die spätere Implementierung sind. Diese Entscheidungen sind oft schwierig zu revidieren, müssen jedoch oft basierend auf Entscheidungen getroffen werden, da es nicht möglich ist deren Auswirkungen frühzeitig zu untersuchen. In diesem Paper stellen wir das Simulationsframework FERAL vor, dass es ermöglicht bestehende oder prototypische Verhaltenskomponenten in einer simulierten Umgebung auszuführen, die die relevanten Eigenschaften der geplanten Softwarearchitekturen widerspiegelt. Die Verhaltenskomponenten müssen dafür nicht modifiziert werden. Auswirkungen von Design- und Deploymententscheidungen können so frühzeitig vorhergesagt und gegebenenfalls korrigiert werden.

1 Einleitung

Multi-Core Prozessoren und die damit mögliche Konsolidierung von unabhängigen Steuergerätefunktionen auf einer gemeinsamen Hardware stellen Software und Funktionsentwickler vor Herausforderungen. Bisher wurde Software in der Regel auf föderierten Architekturen betrieben; jede Funktion wird auf einem unabhängigen Steuergerät ausgeführt. Daten werden über Netzwerke ausgetauscht. Dies vereinfacht die Entwicklung von Funktionen, da die netzwerkbasierte Kommunikation klare Schnittstellen vorschreibt. Interferenzen zwischen Funktionen treten in der Regel nur dann auf wenn mehrere Funktionen den gleichen Aktuator nutzen. Diesem Aspekt wird dadurch Rechnung getragen, dass Funktionen, die die gleichen Aktuatoren nutzen, normalerweise auf demselben Steuergerät ausgeführt werden, und daher während der Steuergeräteentwicklung frühzeitig miteinander integriert werden können.

Verglichen mit den für die Steuergeräteentwicklung bisher genutzten Prozessoren bieten Multi-Core ECU[1]s deutlich mehr Ressourcen. Diese können dazu genutzt werden um

[1] ECU = Electronic Control Unit, diese Abkürzung bezeichnet eine Steuergerätehardware für Automotive-Anwendungen

eine Vielzahl von Funktionen auf einem einzigen Steuergerät zu konsolidieren. Ebenfalls denkt man darüber nach, Regelkreise auf mehrere Steuergeräte zu verteilen, um zum Beispiel eine effizientere Datenverarbeitung in der Nähe von Sensoren zu ermöglichen und freie Prozessor- und Speicherkapazitäten besser nutzen zu können.

Software-Plattformen wie zum Beispiel AUTOSAR [KF09] unterstützen die Entwickler teilweise bei diesen Aktivitäten. AUTOSAR trennt Plattformdienste und Applikationsfunktionen, mehrere Funktionen können auf einer Steuergerätehardware ausgeführt werden und auch über Steuergerätegrenzen hinaus miteinander kommunizieren. Diese Kommunikation wird durch AUTOSAR automatisch auf das genutzte Kommunikationsnetzwerk, zum Beispiel CAN, abgebildet. Ebenso bildet AUTOSAR Anwendungsfunktionen auf Tasks ab, die im Fall einer Multi-Core ECU gleichzeitig ausgeführt werden können. Auch bei der Nutzung von AUTOSAR müssen Entwickler sicherstellen, dass bei der parallelen Ausführung ihrer Applikationen keine Dateninkonsistenzen auftreten und geteilte Ressourcen konfliktfrei genutzt werden können. Ebenfalls müssen auf mehrere Steuergeräte verteilte Algorithmen robust sein gegenüber Kommunikationsfehlern, sowie absichtlichem oder unabsichtlichem Fehlverhalten anderer Busteilnehmer.

Daher ist es erforderlich, dass sowohl Software- als auch Plattformkonzepte frühzeitig und unabhängig voneinander getestet werden, um die notwendige Betriebssicherheit zu gewährleisten und teuren Fehlentscheidungen vorzubeugen. In dieser Arbeit illustrieren wir, wie Parallelisierungs- und Deployment-Entscheidungen frühzeitig mittels des Simulationsframeworks Fraunhofer FERAL (FramEwork for Simulator Coupling on Requirements and Architecture Level) durch virtuelle Deployments getestet werden können. Wir zeigen, wie Entwickler die Auswirkungen ihrer Entscheidungen durch Simulationstechnologien frühzeitig evaluieren und bewerten können. Im Gegensatz zu häufig genutzten Hardware-in-the-Loop Lösungen können Architekturkonzepte daher bereits getestet werden, bevor diese für die Zielplattform umgesetzt wurden. Ebenfalls illustrieren wir wie diese Evaluation ohne eine Modifikation des evaluierten Applikationscodes nur durch umgebende Simulationskomponenten durchgeführt werden kann um Designfehler frühzeitig zu erkennen und den Aufwand für Simulationen signifikant zu senken.

2 Software im Automobilbereich

Im Fokus dieser Arbeit steht Software im Automobilbereich, die steuernde und regelnde Aufgaben wahrnimmt. Diese Software realisiert verschiedene Funktionen, wobei eine Funktion eine größere Einheit darstellt. Funktionen sind in der Regel weiter in Unterfunktionen strukturiert, und werden schließlich durch Runnables implementiert. Häufig wird eine Funktion daher durch mehrere kommunizierende Runnables realisiert. Dieses kommunizierende Netzwerk bezeichnet man als Funktionsnetz.

2.1 Automotive Software

Diese Kommunikation wird Beispielsweise mittels globaler Variablen realisiert. Basierend auf den von einem Runnable gelesenen und geschriebenen Daten lassen sich Funktionsnetze konstruieren, die die Abhängigkeiten zwischen Funktionen darstellen.

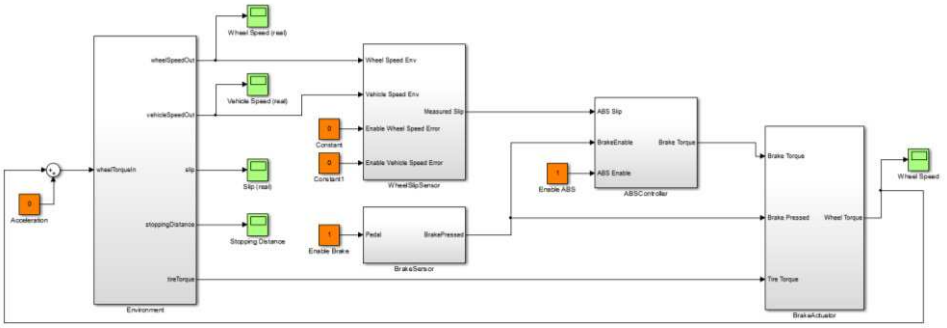


Abbildung 1: Beispielsystem in Simulink

Abbildung 1 illustriert ein in Simulink erstelltes Funktionsnetz das ein vereinfachtes ABS System realisiert. Es werden vier Funktionen und die Umgebung simuliert:

- Der Wheel Slip Sensor überwacht ein Rad des Autos und prüft ob dieses blockiert und rutscht.
- Der Brake Sensor überwacht das Bremspedal des Brake-By-Wire Systems und signalisiert dem ABS Controller elektrisch einen Bremswunsch des Fahrers.
- Der ABS Controller steuert das ABS System und entscheidet ob mit der vollen Bremskraft gebremst werden soll, oder ob diese aufgrund blockierender Räder reduziert werden muss.
- Der Brake Actuator nimmt die Befehle des ABS Controllers und des Fahrers entgegen und steuert die Bremse. Der ABS Controller kann den Bremsdruck abschwächen, jedoch keinen eigenen Bremsdruck aufbauen.

2.2 Simulation auf Funktionsebene mit FERAL

Während der Funktionsentwicklung muss es möglich sein ein System zu testen. Wird die Entwicklung wie bei dem in Abbildung 1 gezeigten Beispielsystem vollständig in einem Werkzeug durchgeführt, so ist es natürlich möglich das Verhalten der Algorithmen innerhalb dieser Entwicklungsumgebung ohne zusätzliche Werkzeuge zu testen. Immer häufiger ist dies jedoch nicht möglich, da entweder Applikationskomponenten die das Verhalten von Runnables realisieren nicht im Quelltest vorliegen, oder diese mit anderen Werkzeugen entwickelt wurden. Das Testen eines Funktionsnetzes erfordert in diesem Fall eine semantische Integration von verschiedenen Runnables. Diese Integration sollte

ermöglich werden ohne dass Applikationskomponenten verändert werden müssen. Dies ermöglicht die Integration von Komponenten, deren Quelltext oder Designmodell nicht verfügbar ist.

Das Fraunhofer FERAL Simulationsframework ermöglicht die integrierte Simulation von verschiedenen Runnables in einem Kontext. Dieser wird komponentenbasiert realisiert. Das Kernkonzept von FERAL ist dabei die Trennung von Simulationsmodellen und dem simuliertem Verhalten. Simulationsmodelle, d.h. Berechnungs- und Kommunikationsmodelle (Models of Computation and Communication - MOCC) die zur Ausführung von Funktionen und Simulationsmodellen erforderlich sind werden wiederverwendbar von Direktoren implementiert. Berechnungsmodelle definieren wann eine bestimmte Komponente ausgeführt wird, und wie lange diese Ausführung in Simulationszeit dauert und dauern darf. Kommunikationsmodelle definieren wann welche Daten wie zwischen Simulationskomponenten ausgetauscht werden. Das konkrete simulierte Verhalten einer Funktion wird durch einen Worker bereitgestellt. Worker erfordern bestimmte Verhaltensmodelle, daher können diese nur zusammen mit kompatiblen Direktoren eingesetzt werden. Diese Trennung ermöglicht eine effiziente Simulatorkopplung: diese erfordert eine Integration der unterschiedlichen Ausführungs- und Kommunikationsmodelle, die wiederverwendbar durch Direktoren bereitgestellt wird. Die Integration eines konkreten Simulators wird mittels eines Workers realisiert, der lediglich eine technische Schnittstelle zum Simulationsmodell implementieren muss.

Angelehnt an Ptolemy [Ek03] definieren sowohl Direktoren als auch Worker eine gemeinsame Schnittstelle. Dies ermöglicht die Schachtelung von Direktoren und Workern in hierarchische Szenarien. Jede Komponente bis auf die Wurzel hat exakt einen übergeordneten Direktor, der deren Ausführung kontrolliert. Worker treten nur als Blätter auf. Auf diese Weise werden Simulationsszenarien realisiert, die unterschiedliche Simulationsmodelle vereinigen. Die vollständige Semantik des FERAL Frameworks ist in [Ku13] dokumentiert. Für die Ausführung von Komponenten im Kontext von virtuellen Deployments mit FERAL sind insbesondere die folgenden Komponenten relevant:

- TimeTriggeredDirector: Dieser Direktor führt die kontrollierten Komponenten in festen zeitschritten aus. Optional kann eine Reihenfolge vergeben werden.

- EventTriggeredDirector: Dieser Direktor führt enthaltene Komponenten dann aus, wenn ein Ereignis für eine Komponente aufgetreten ist. Dies kann ein Timer oder ein Nachrichtenempfang sein.

- SimulinkWorker: Integriert ein Simulink System als Verhaltensmodell. Dieser Worker benötigt eine Ausführsemantik, die zu der des TimeTriggeredDirector kompatibel ist.

- FMUWorker: Analog zu dem SimulinkWorker, für Functional Mockup Units.

- TimeTriggeredWorker: Worker dessen Ausführung zu festen Zeitpunkten durch dessen Direktor angestoßen wird. Ein- und Ausgangsports verhalten sich wie

Variablen die während der Ausführung beliebig gelesen und geschrieben werden. Dies ist die Basisklasse des Simulink- und FMU Workers.

- EventTriggeredWorker: Ereignisbasierter Worker, der von seinem Direktor bei Nachrichtenempfang oder Ablauf eines selbst gesetzten Timers ausgeführt wird. Eingabeports verhalten sich wie Nachrichtenqueues, Nachrichten an Ausgangsports werden vom Direktor zu definierten Zeitpunkten an verbundene Eingabeports anderer Komponenten weitergeleitet.

- Constant: Spezieller TimeTriggeredWorker der lediglich eine Konstante an einem Ausgangsport bereitstellt.

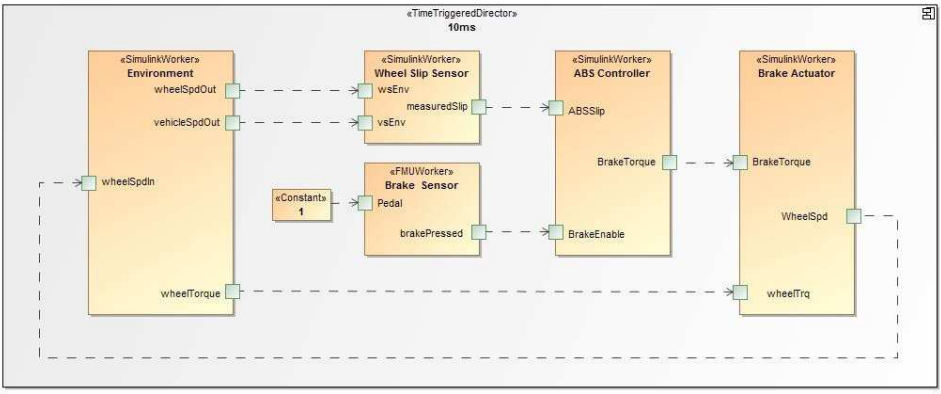


Abbildung 2: Modell eines Simulationsszenarios des FERAL Simulationsframeworks

Abbildung 2 illustriert den Aufbau solches Szenario mit Komponenten des FERAL Frameworks. Genutzt wird ein TimeTriggeredDirector, die alle seine kontrollierten Komponenten in äquidistanten Zeitschritten aufruft und den Datenaustausch über Ports und Links kontrolliert. Verschiedene Worker binden Simulink und FMU Subsysteme ein und ermöglichen die gemeinsame Ausführung des Systems in einem integrierten Szenario.

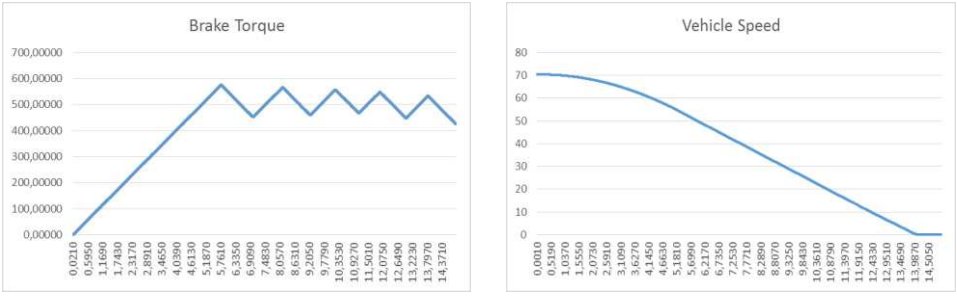


Abbildung 3: Verhalten des Beispielsystems in einem simulierten Szenario

Abbildung 3 illustriert das Verhalten des simulierten Systems. Man erkennt deutlich das Verhalten das ABS Systems der die Räder, sobald sie aufgrund einer zu hohen Bremskraft zu blockieren beginnen wieder freigibt und so das Bremsverhalten des Automobils optimiert.

3 Virtuelle Deployments mit Fraunhofer FERAL

Runnables werden auf einer realen Zielplattform durch ein Netzwerk von Steuergeräten (Electronic Control Unit - ECU) ausgeführt. Dabei spielen Deployment Entscheidungen eine wesentliche Rolle. Diese legen fest auf welcher ECU ein Runnable ausgeführt wird, welcher Task diese Ausführung übernimmt, und welche Daten aufgrund des Deployments über Kommunikationsnetze ausgetauscht werden müssen. Das Deployment hat einen wesentlichen Einfluss auf die Ausführung eines Algorithmus. Wartezeiten (Delays) und deren Varianzen (Jitter) die z.B. durch Multitasking und die Nutzung von Kommunikationsnetzwerken bei der Ausführung von Runnables entstehen, beeinflussen das Verhalten eines Algorithmus signifikant. Um die Auswirkungen dieser Entscheidungen vorherzusagen muss das in Abbildung 2 gezeigte Szenario daher um die Simulation von Deployments erweitert werden. Ziel ist hierbei, die Komponenten des FERAL Frameworks zu nutzen um Eigenschaften der Zielplattform in die Simulation einzubringen, ohne hierfür die Applikationskomponenten anzupassen.

3.1 Virtuelle Deployments auf ECUs

Um ein Deployment zu simulieren müssen in einem ersten Schritt die Tasks der Zielplattform mit FERAL nachgebildet werden. Hierbei sind zwei Arten von Tasks besonders relevant:

- Zeitgesteuerte Tasks wechseln zu bestimmten Zeitpunkten in den Zustand „Bereit“. Abhängig von ihrer Priorität werden diese danach mehr oder weniger zeitnah ausgeführt. Diese Tasks werden für die meisten Runnables genutzt, die steuernde und regelnde Aufgaben übernehmen.

- Ereignisgesteuerte Tasks werden durch ein externes Ereignis in den Zustand „Bereit“ versetzt. Dies ist häufig ein Interrupt, kann aber auch durch den Empfang einer Nachricht über ein Netzwerk oder den Aufruf einer bereitgestellten Funktion durch einen anderen Task ausgelöst werden. Ein Beispiel für einen solchen Task ist die Steuerung eines Einspritzvorgangs, der mit der Kurbelwellenposition und Geschwindigkeit synchronisiert werden muss, und für den daher keine feste Periodendauer definiert werden kann.

Mit FERAL werden Deployments zum Beispiel mit Hilfe des HybridScheduler Direktors realisiert. Es handelt sich hierbei um eine spezialisierte Form des EventTriggeredDirector. Der HybridScheduler realisiert das prioritätsbasierte Scheduling eines OSEK konformen Betriebssystems und kann entweder präemptiv oder nicht präemptiv konfiguriert werden. Runnables werden weiterhin über Worker

eingebunden, die Ausführung der Worker wird bei Verwendung dieses Direktors durch Tasks gesteuert, auf die die Worker allokiert wurden.

Abbildung 4 zeigt ein virtuelles Deployment mit FERAL, dass eine AUTOSAR-konforme Ausführplattform simuliert. Die Kommunikation zwischen dem Worker und der Plattform findet über Signale statt. Die Ausführung von Workern wird bei Nutzung dieses Direktors über Tasks gesteuert. Es sind zwei Tasks definiert: ein periodischer Task führt den ABSController alle 10ms aus. Ein ereignisbasierter Task führt die RTE aus und reagiert dabei auf Scheduling Events des simulierten Betriebssystems. Die RTE ermöglicht den Datenaustausch zwischen Tasks und steuert die Netzwerkkommunikation. Ausführzeiten die für Taskwechsel oder den atomaren Zugriff auf Speicheradressen benötigt wird, wird durch die Ausführzeiten der RTE simuliert.

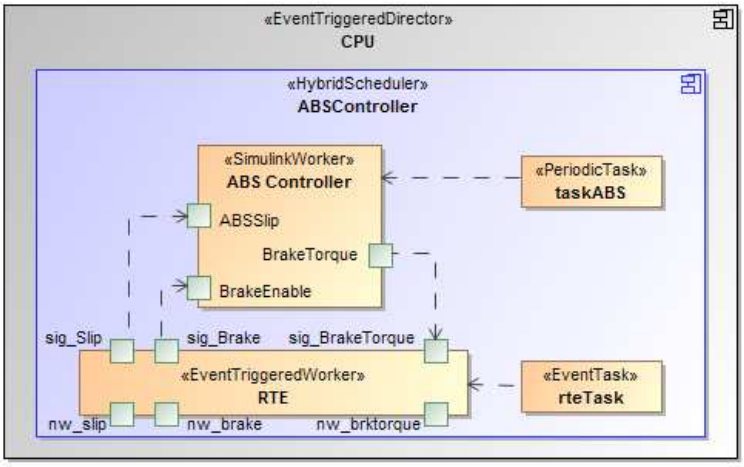


Abbildung 4: Modell eines Simulationsszenarios für virtuelles Deployment auf ECU

Die Runnables des simulierten Szenarios werden hier bereits mit einer Ressourcenbeschränkung ausgeführt. Im Gegensatz zu dem vorher genutzten TimeTriggeredDirector implementiert der hier genutzte Scheduler ein Ausführmodell das Tasks auf einer festen Anzahl Rechenkernen ausführt. Jeder Task benötigt eine bestimmte Menge Rechenzeit, für die er den Kern blockiert. Die Reihenfolge in der bereite Tasks ausgeführt werden wird durch deren Priorität bestimmt.

AUTOSAR nutzt ein preemptives Scheduling, daher können höherpriore Tasks solche mit niedriger Priorität verdrängen. Dies kann die Ausführzeit eines Tasks signifikant verlängern. FERAL bildet dieses Verhalten nach, indem die Ausführung eines Tasks mehrere definierte Zeitintervalle dauern kann. Nach jedem Zeitintervall wird geprüft ob der Task noch weiter rechnen darf, oder von einem Task mit höherer Priorität verdrängt wurde. Abbildung 5 zeigt dafür ein Beispiel. Hier wird ein Task mit einer Gesamtlaufzeit von 800µs in Zeitschlitzen von 100µs ausgeführt und während der Ausführung von einem Task mit einer höheren Priorität verdrängt. Die Länge der Zeitschlitze kann für jeden Task abhängig von der geforderten Genauigkeit individuell

definiert werden, ebenfalls müssen diese nicht äquidistant sein. FERAL führt die Berechnung eines Tasks in diesen Zeitschlitzen aus. Da bestehende Runnables unmodifiziert ausgeführt werden sollen ist es möglich, dass diese nur einen atomaren Zeitschritt realisieren. Dies ist zum Beispiel bei Code der aus Simulink generiert wurde der Fall. In diesem Fall muss der Task festlegen, in welchem Zeitschlitz das Runnable tatsächlich ausgeführt werden soll. Ferner muss definiert werden in welchen Zeitschlitzen Eingangsdaten gelesen- und Ausgangsdaten geschrieben werden. Das Standardverhalten des Direktors definiert alle lesenden Operationen und die atomare Berechnungen während des ersten Zeitschlitzes, die schreibenden Operationen während des letzten Zeitschlitzes.

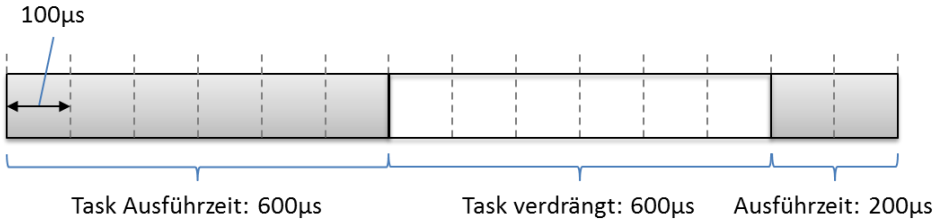


Abbildung 5: Simulationsszenario des FERAL Simulationsframeworks

Mit dem gezeigten Ansatz wird ein Deployment auf eine ECU simuliert, ohne das die eigentlichen Runnables dafür modifiziert werden müssen. Einfache Parallelisierungsfehler können so bereits erkannt werden: greift ein Task schreibend auf Variablen zu, die durch einen verdrängten Task ebenfalls genutzt werden, so handelt es sich dabei in der Regel um ein ungewolltes Verhalten. Außerdem ist es möglich den Einfluss eines Schedulings auf das Verhalten eines Systems zu bewerten.

3.2 Virtuelle Deployments auf Multi-Core ECUs

Die parallele Ausführung von Funktionen wird aufgrund der sich immer weiter verbreitenden Multi-Core Prozessoren immer wichtiger. FERAL simuliert diese Deployments wie in Abbildung 6 illustriert durch zusätzliche Kerne, die simuliert parallel ausgeführt werden. Jeder dieser Kerne wird durch einen eigenen Direktor vom Typ HybridScheduler repräsentiert. Zusätzlich zu den beiden bereits allokierten Funktionen WheelSlipSensor und BrakeActuator wird nun eine weitere Funktion PowerSteeringActuator auf einen zweiten Kern der ECU allokiert. Abbildung 6 zeigt diese Multi-Core ECU als Erweiterung des Deployments aus Abbildung 4. Aus Gründen der Übersicht wurden nur die Worker ohne Tasks modelliert.

Um Dateninkonsistenzen bei der gleichzeitigen Ausführung von Runnables auf Multi-Core Prozessoren zu vermeiden werden zum Beispiel Softwaretransaktionen genutzt (siehe [ST97]). Diese kopieren Eingabedaten zu einem definierten Zeitpunkt in einen lokalen Puffer. Das ausgeführte Runnable arbeitet nur auf diesen Daten, die zu einem weiteren Zeitpunkt mit den globalen Daten wieder synchronisiert werden. So muss das Runnable nicht geändert werden und die Synchronisationskonzepte können unabhängig

von der Implementierung der Runnables realisiert werden. Finden die Lese/Schreib-Transaktionen in definierten Zeitschlitzen statt in denen nur eine Transaktion aktiv ist, kann ein konfliktfreier Zugriff garantiert werden. Aufgrund dieser Eigenschaften ermöglichen Softwaretransaktionen sehr elegante Parallelisierungskonzepte.

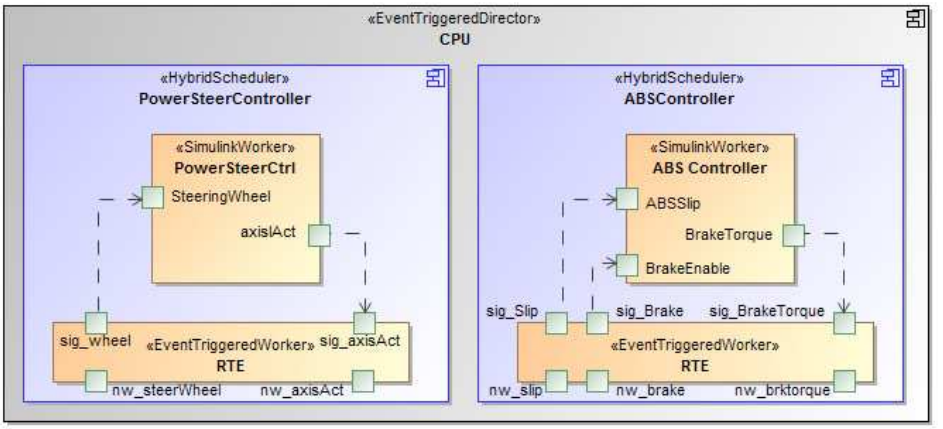


Abbildung 6: Simulationsszenario für eine Multi-Core ECU

Mittels FERAL können diese Konzepte validiert werden. Abbildung 7 illustriert die Modellierung einer Softwaretransaktion mit dem FERAL Simulationsframework. Auf diese Weise ist es möglich, frühzeitig den Einfluss von Softwaretransaktionen zu überprüfen, sowie zu überprüfen ob Transaktionen basierend auf Tasklaufzeiten korrekt konfiguriert wurden und konfliktfrei ablaufen.

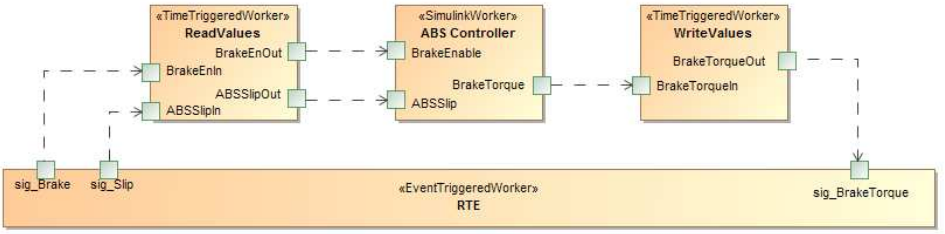


Abbildung 7: Modell der Realisierung einer Softwaretransaktion in FERAL

3.3 Virtuelle Deployments auf Steuergerätenetzwerke

Neben dem Task-Scheduling ist auch die Verteilung von Funktionen auf Steuergerätenetzwerke relevant. Abbildung 8 zeigt das Modell eines solches komplexes Simulationsszenario. Dieses besteht aus drei Single-Core ECUs, sowie einem CAN Bus Netzwerk das die ECUs miteinander verbindet. Das Umgebungsmodell wird von einem TimeTriggeredDirector ohne Ressourcenconstraints ausgeführt. Die Softwarefunktionen

werden erneut durch den HybridScheduler ausgeführt um das Scheduling der ECU zu simulieren. Das Kommunikationsnetz wird durch ereignisbasierte Worker realisiert. Die RTE wurde nicht dargestellt um die Grafik zu vereinfachen.

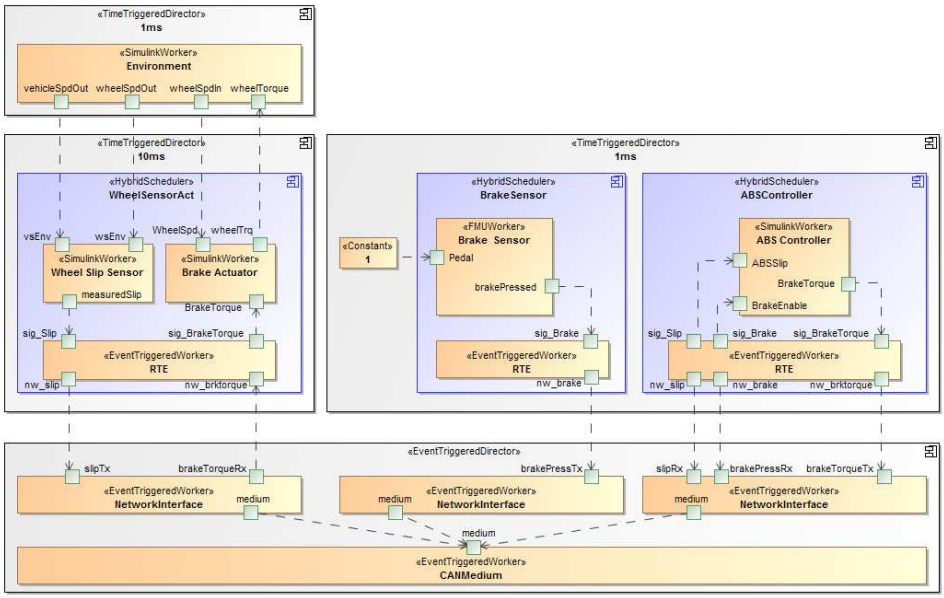
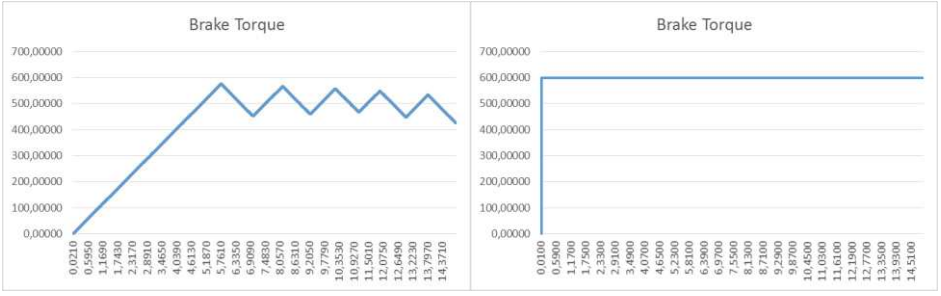


Abbildung 8: Modell eines FERAL Simulationsszenarios für ein Steuergerätenetzwerk

Wie man in Abbildung 9.a sehen kann beeinflusst dieses Deployment die Ausführung der Komponenten nur minimal. Dies liegt daran, dass ausreichend Ressourcen in diesem Szenario zur Verfügung stehen, und daher sowohl die Ausführzeit als auch die Kommunikation nur minimalen Verzögerungen unterliegt, die in dieser Größenordnung für den Algorithmus nicht kritisch sind.



(a) ohne Powersteering-Komponenten (b) mit Powersteering-Komponenten

Abbildung 9: Resultate der Simulation

Abbildung 9.b zeigt das Verhalten dieses Szenario mit zwei zusätzlich verteilten Funktionen: Um Kosten zu sparen wurden Funktionen der Servolenkung auf die ECUs BrakeSensor und WheelSensorAct verteilt. Hierdurch wird ein weiterer Regelkreis in das System eingebracht, der zusätzliche Ressourcen benötigt. Dadurch wird der CAN Bus überlastet und Nachrichten mit niedrigerer Priorität verworfen. Im betrachteten Beispiel sind dies die Nachrichten der Bremskraftsteuerung des ABS, während die Brake-by-Wire und Steer-by-Wire Signale der Bremse und der Servolenkung übertragen werden. Aufgrund der durch die Simulationskomponenten über ihre Schnittstellen publizierten Daten lässt sie die Quelle der Störungen leicht finden und beheben, zum Beispiel in dem Nachrichten zusammengefasst werden, oder mit niedrigerer Frequenz übertragen werden. Dies zeigt, dass eine Fehlersuche in einem virtuellen Deployment schnell möglich ist und eine sinnvolle Indikation bezüglich des späteren Systemverhaltens liefert.

4 Verwandte Arbeiten

Die Kopplung von Simulatoren ist ein aktuelles Forschungsthema das bereits in anderen Arbeiten betrachtet wurde. Hierbei wurden in der Regel spezialisierte Lösungen für einen Anwendungsfall entwickelt: Ein konkrete Lösung ist in [Wü97] beschrieben. Sie wurde bereits 1997 publiziert und beschreibt die Kopplung der Simulationswerkzeuge ANSYS und SABER um die Entwicklung mikroelektronischer Schaltungen zu vereinfachen. ANSYS ist ein Werkzeug das lineare und nichtlineare kontinuierliche Systeme simuliert, SABER simuliert elektronische Schaltungen und Systeme. Die publizierte Simulatorkopplung fokussiert auf die Simulation thermischer Eigenschaften. Die Autoren der als [FW07] referenzierten Veröffentlichung fokussieren im Gegensatz dazu auf die funktionale Evaluation ihres Systems. Durch die Kopplung der Simulatoren SLSim und SMASH wurde eine integrierte Simulation von VHDL Code und kontinuierlichen SPICE-Modellen ermöglicht.

Modelisar [Bl11] ist ein Standard, der die Kapselung von Simulationsmodellen in Functional Mockup Units (FMU) definiert. Hierdurch kann Funktionscode zwischen Stakeholdern mit einer einheitlichen Simulationsschnittstelle weitergegeben werden. FERAL unterstützt diese Schnittstelle um Applikationscode in virtuelle Deployments einzubinden. Das Ptolemy II Framework [Ek03] wird genutzt um die Kombination von unterschiedlichen Ausführmodellen zu erforschen. Es teilt Systeme in unterschiedliche, verschachtelbare semantische Domänen auf, die jeweils die direkt enthaltenen Komponenten kontrollieren. Dieses Framework ist ebenfalls die Basis für den in FERAL genutzten Kopplungsansatz, wurde jedoch an einigen Stellen adaptiert (siehe [Ku13]) um die speziellen Erfordernisse einer Simulation zu realisieren.

Bestehende Lösungen zur Systemsimulation basierend auf SystemC Modellen (z.B. Open Virtual Platforms – www.ovpworld.org) nutzen Instruction Set Simulatoren und Hardwaremodelle zur Erstellung von Simulationsmodellen. Diese ermöglichen es fertige Implementierungen auf einer simulierten Zielhardware zu testen. Im Gegensatz zu FERAL erfordert dies eine fertige Implementierung für die Zielplattform und adressiert daher eher Funktionstests als die frühe Validation von Architekturentscheidungen.

5 Zusammenfassung und Ausblick

Zukünftige Architekturen für eingebettete Systeme werden wesentlich komplexer sein als heute genutzte Architekturen. Um eine sinnvolle Entscheidungsgrundlage für Software- und Systemarchitekten zu bieten sind daher auch neue Ansätze zur Entscheidungsunterstützung erforderlich. Mit FERAL ermöglichen wir die frühzeitige Simulationen auf Systemebene. Dabei wird der Systemkontext durch Simulationskomponenten realisiert, eine Anpassung des Funktionscodes ist nicht notwendig. Verhaltenscode und Verhaltensmodelle können über Worker unverändert in Simulationen eingebunden werden. Dies ermöglicht die schnelle Konstruktion von integrierten Simulationsszenarien und erlaubt die effiziente Evaluation einer Vielzahl von Entscheidungsmöglichkeiten basierend auf Fakten.

Das in diesem Artikel vorgestellte Vorgehen wird im Prognostics Center des Fraunhofer IESE angewandt. Die Ansätze sind praxiserprobt und zeigen, dass eine Verteilung und Parallelisierung bestehender Software auch im Bereich sicherheitskritischer eingebetteter Systeme möglich ist. Um den Aufwand hierfür gering zu halten und die zu treffenden Architekturentscheidungen abzusichern verwenden wir am Fraunhofer IESE Analysewerkzeuge und Softwaremodelle, um zum Beispiel die Auswirkungen der Aufteilung von Runnables in Gruppen frühzeitig vorherzusagen. So ist gewährleistet, dass eine gefundene Parallelisierungsarchitektur robust gegen Änderungen ist und konzeptionelle Entscheidungen Bestand haben, zum Beispiel dann wenn zukünftig leistungsfähigere Prozessoren mit noch mehr Kernen genutzt werden sollen.

Literaturverzeichnis

[KF09] Kindel, O., Friedrich, M.: Softwareentwicklung mit AUTOSAR. Grundlagen, Engineering, Management für die Praxis. dpunkt.verlag, 2009, ISBN 978-3-89864-563-8.

[Ek03] Eker, J., Janneck, J., Lee, E., Liu, J., Liu, X., Ludvig, J., Sachs, S., Xiong, Y.: Taming heterogeneity - the Ptolemy approach. Proceedings of the IEEE, 91(1):127-144, January 2003.

[Ku13] Kuhn, T., Forster, T., Braun, T., Gotzhein, R. FERAL – Framework for Simulator Coupling on Requirements and Architecture Level. Proceedings of the 11th ACM-IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE'13), Portland (Oregon), USA, 2013.

[ST97] Shavit, N., Touitou, D. Software transactional memory. Distributed Computing. Volume 10, Number 2. February 1997.

[Wü97] Wünsche, S., Clauss, C., Schwarz, P., Winkler, F. Microsystem Design Using Simulator Coupling. European Design and Test Conference, 1997. ED&TC 97, IEEE (1997)

[FW07] Frank, F., Weigel, R. Co-Simulation of SPICE Netlists and VHDL-AMS Models via a Simulator Interface. International Symposium on Signals, Systems and Electronics, 2007. ISSSE '07. (2007)

[Bl11] Blochwitz, T., Otter, M., Arnold, M., Bausch, C., Clauß, C., Elmqvist, H., Junghanns, A., Mauss, J., Monteiro, M., Neidhold, T., Neumerkel, D., Olsson, H., Peetz, J., Wolf, S. The Functional Mockup Interface for Tool independent Exchange of Simulation Models. Proceedings of the 8th International Modelica Conference. 8th International Modelica Conference, 20.-22 März 2011, Dresden.