Modellbasierte Generierung von statischen Schedules für sicherheitskritische, eingebettete Systeme mit Multicore Prozessoren und harten Echtzeitanforderungen

Robert Hilbrich, J. Reinier van Kampenhout, Hans-Joachim Goltz

Fraunhofer-Institut für Rechnerarchitektur und Softwaretechnik FIRST
Kekuléstr. 7, 12489 Berlin
{robert.hilbrich,j.r.van.kampenhout,hans-joachim.goltz}@first.fraunhofer.de

Zusammenfassung. In dieser Arbeit wird anhand des neuen Werkzeugs PRECISION PRO das Potential der automatisierten Generierung von statischen Schedules für sicherheitskritische, eingebettete Systeme beschrieben. Dessen Erstellung ist häufig mit einem hohen manuellen Aufwand verbunden, da er von hoher Bedeutung für einen erfolgreichen Zertifizierungsprozess ist. Eine modellbasierte Generierung ermöglicht ein schnelles Feedback, eine frühe Bewertung der Hardware- und Softwarearchitekturen, sowie erweiterte Möglichkeiten zur Optimierung der Systemauslastung. Dabei ist immer sichergestellt, dass alle spezifizierten Anforderungen erfüllt sind.

1 Einführung

Software-intensive, eingebettete Systeme übernehmen heute viele zentrale Steuerfunktionen in zahlreichen Anwendungsbereichen. Besonders deutlich wird diese Entwicklung im Automotive-Bereich, wo die Bedeutung von Fahrerassistenzsystemen, wie prädiktiver Kollisionsvermeidung, adaptiver Geschwindigkeitsregelung und intelligenter Navigationsführung, signifikant zunimmt. Gartner Analysten prognostizieren den Anteil der Kosten für die Fahrzeugelektronik eines Neuwagens im Jahr 2018 auf etwa 30% [6]. Neben steigenden funktionalen Anforderungen, müssen diese Systeme auch den aktuellen Sicherheitsstandards und Zertifizierungsanforderungen gerecht werden, z.B. gegeben durch die ISO 61508, ISO 26262 oder DO-178B.

Insbesondere in sicherheitskritischen Bereichen mit hohem Gefährdungspotenzial für Mensch und Umwelt ist ein Höchstmaß an Zuverlässigkeit nötig. Für eine erfolgreiche Zertifizierung in den höchsten Sicherheitsstufen, z.B. DAL A im Luftfahrtbereich, muss das Verhalten sowohl einfacher, als auch hochkomplexer Systeme zu jedem Zeitpunkt der Laufzeit genau bekannt sein, so dass harte Echtzeitanforderungen nachweisbar erfüllt werden können.

Neben verschiedenen Hardware-Eigenschaften wird dies durch die Festlegung der Ablaufmuster von einzelnen Teilaufgaben - dem *Scheduling* des Echtzeit-Betriebssystems - bestimmt. Bei vielen Computeranwendungen, wie z.B. dem

Heim-PC, entscheidet der Nutzer spontan, welches Programm er nutzen möchte. Im Gegensatz dazu führen Systeme in sicherheitskritischen Bereichen geordnete und bereits zum Entwicklungszeitpunkt genau festgelegte Arbeitsschritte aus, deren Ausführung sich im Rahmen eines betrachteten Zeitfensters *kontinuierlich* wiederholt (das *Scheduling Intervall*).

2 Erstellung von statischen Schedules für Multicore Prozessoren

Bei der Konstruktion eines festen Scheduling Intervalls wird oft noch manuell, z.B. mit Hilfe von Tabellenkalkulationsprogrammen, festgelegt, wann einzelne Applikationen ausgeführt werden sollen. Dabei stoßen die Entwickler schnell an ihre Grenzen, da sehr viele Variationsmöglichkeiten und Randbedingungen beachtet werden müssen.

Beispielsweise müssen bei der Erstellung unterschiedliche Periodenlängen von Applikationen, diverse Abhängigkeiten zwischen einzelnen Applikationen und auch deren Zugriffe auf externe Ressourcen, wie Kommunikationskanäle oder Aktuatoren, beachtet werden. Gleichzeitig sollte ein Prozessor einen Arbeitsschritt möglichst zusammenhängend bearbeiten können, um teure Umschaltzeiten zu vermeiden.

Während dies bei Single-Core Prozessoren noch weitgehend manuell durchgeführt werden konnte, fügt die Nutzung von Multi- und Manycore Prozessoren neue Freiheitsgrade bei der Erstellung eines Schedules hinzu, so dass die resultierende Komplexität des *Scheduling-Problems* für komplexe, eingebettete Systeme nicht mehr manuell zu bewältigen ist.

Die wichtigen Ziele im Engineering Prozess eines eingebetteten Systems: *optimale Ausnutzung der vorhanden Ressourcen* und *korrektes Echtzeitverhalten* lassen sich durch manuelle Planungsverfahren nicht mehr mit vertretbarem Aufwand erreichen. Gleichzeitig schränkt der hohe Planungsaufwand aber auch die Flexibilität der Softwareentwickler ein. Für sie sind die Auswirkungen von Änderungen an der eigenen Softwarekomponenten auf das Echtzeitverhalten des Gesamtsystem nur schwer überschaubar. Größere Änderungen sind wegen des Zeitaufwands für Umplanungen häufig unmöglich.

3 Modellbasierte Generierung von statischen Schedules – *Correctness by Construction*

Im Rahmen aktueller Arbeiten wird für diese Problematik ein Generator für statische Schedules von periodischer Anwendungen auf sicherheitskritischen, eingebetteten Systemen entwickelt: *PRECISION PRO*.

Als Eingabe benötigt der Generator:

- das *Hardware-Angebot* - ein vereinfachtes Modell der zur Verfügung stehenden Hardware-Architektur

- die *Software-Anforderungen* - Ausführungseigenschaften der Applikationen, wie Worst-Case Execution Time, Periode, Abhängigkeiten zu anderen Applikationen, ...
- ein *feste Zuordnung* von Applikationen zu Prozessoren.

Damit ist das Werkzeug in der Lage, einen statischen Schedule zur Entwicklungszeit zu generieren, der alle geforderten Bedingungen *by construction* erfüllt, oder bei Nicht-Erfüllbarkeit eine Fehlermeldung zurückgibt. Im Vergleich zu anderen Werkzeugen im Bereich Echtzeit-Scheduling (siehe [3]), adressiert PRECISION PRO das Scheduling Problem mit Hilfe von speziellen Heuristiken *direkt* und *konstruiert* einen Schedule, der die gewünschten Anforderungen zeigt.

Klassische Ansätze beschränken sich entweder auf die Analyse eines erstellten Schedules oder können nur einfache Scheduling-Probleme lösen. PRECISION PRO bietet zudem die Möglichkeit zur Modellierung von speziellen Ressourcen (zum Beispiel die Netzwerkbandbreite eines Kommunikationsbusses im System), so dass Schedules generiert werden können, die eine Ressourcenüberlastung von vornherein vermeiden. Der grundlegende Ansatz zur Modellierung von Scheduling-Problemen ist detaillierter im Abschnitt 4 beschrieben.

Mit Hilfe eines statischen Schedules kann ein deterministisches Echtzeitverhalten des Gesamtsystems - insbesondere bei Multicore Prozessoren - erreicht werden, da eine nicht-deterministische Auflösung von Ressourcenkonflikten zur Laufzeit bereits a-priori im Entwicklungsprozess verhindert wurde. In [5] wurden diese Konflikte auch als *Interference Channels* bezeichnet. Mit einer entsprechenden Modellierung in PRECISION PRO können diese nicht-deterministischen *Interferences* bereits zur Entwicklungszeit vermieden werden. Dies führt zu einer global synchronisierten Ausführung aller Applikationen auf einem Multicore Prozessor.

Statische Schedules bieten ein hohes Maß an Vorhersagbarkeit und Determinismus des resultierenden Systemverhaltens, so dass auf den Einsatz von Scheduling-Analysewerkzeugen zum Nachweis zeitlicher Eigenschaften für eine Zertifizierung verzichtet werden kann. Gleichzeitig reduziert die Festschreibung eines Ausführungsplans aber auch die Flexibilität im Entwicklungsprozess und schränkt auch den Umgang mit Event-getriebenen Funktionen ein. Änderungen am Zeitverhalten einzelner Softwarekomponenten führen meist auch zu aufwändigen Änderungen am statischen Schedule, so dass die Auswirkungen kleiner Änderungen oft nicht unmittelbar bestimmt und evaluiert werden können. Der Preis einer hohen Vorhersagbarkeit des Systemverhaltens äußert sich zur Laufzeit im Fehlen von umfangreichen Möglichkeiten zur Adaption beim Eintreffen von unvorhergesehenen Ereignissen. Dynamische Schedules bieten hier mehr Flexibilität. In [7] werden verschiedene Scheduling Algorithmen für den Einsatz in sicherheitskritischen Systemen analysiert. [4] diskutiert den Trade-Off zwischen Flexibilität und Determinismus beim Scheduling.

Einen interessanten Mittelweg im Konflikt zwischen Determinismus und Flexibilität stellen die *Mode-basierten* statischen Schedules dar (zum Beispiel für die Avionik in [2] spezifiziert). Für unterschiedliche Betriebsmodi eines Gerätes werden unterschiedliche statische Schedules a priori generiert. Zur Laufzeit kann

dann *dynamisch* von einem Modus in einen anderen gewechselt werden, so dass eine Anpassung des Ausführungsplans an die veränderten Umgebungsbedingungen zur *Laufzeit* möglich ist.

Die strukturellen Nachteile eines statischen Schedules, also die fehlende Flexibilität zur Laufzeit, können durch unser Werkzeug PRECISION PRO nicht beseitigt werden. Durch eine modellbasierte Generierung können aber die Aufwände bei der Erstellung und Wartung von umfangreichen Schedules signifikant reduziert werden, so dass die Entwicklungsprozesse durch schnellere Iterationen und unmittelbares Feedback auf Änderungen in der Software profitieren können. Für die Konstruktion eines statischen Schedules mit einem Intervall von 2000 ms für 30 Anwendungen werden nur wenige Minuten auf einem aktuellem PC benötigt. Eine effiziente Konstruktion bietet damit das Potential, den Integrationsprozess komplexer Systeme signifikant zu verbessern. Das resultierende Systemverhalten von unterschiedliche Zeitkontingenten für verschiedene Applikationen kann bereits am Anfang des Entwicklungsprozesses simuliert und unter verschiedenen Gesichtspunkten optimiert werden.

Ein korrektes Timing-Design und eine optimierte Ressourcenauslegung eines komplexen Systemen sind damit nicht mehr *nur* das Ergebnis von Erfahrungen des Systemarchitekten aus vorangegangenen Projekten, sondern das Resultat der Konstruktion eines *korrekten* Designs unter Einbeziehung zeitlicher Anforderungen, das im Laufe der Entwicklung schrittweise durch genauere Feststellung des Zeitverhaltens der einzelnen Applikationen auf der Hardware präzisiert werden kann.

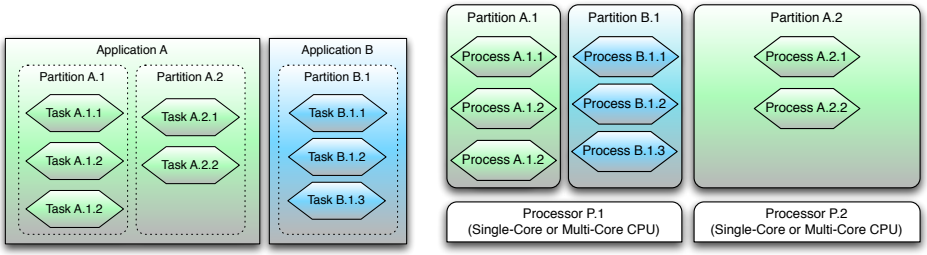
4 Modellierung eines Scheduling Problems

Bei der Modellierung eines Scheduling Problems in PRECISION PRO orientieren wir uns an der Terminologie im Bereich der Avionik [1]. Diese ist als Überblick in Abbildung 1 dargestellt. Eine *Applikation* besteht dabei aus mindestens einer *Partition*. Jede Partition enthält mindestens einen (potentiell nebenläufigen) *Prozess*, der eine Funktion realisiert. Wir sehen einen Prozess als grundlegende *Ausführungseinheit*, die auf einem Core gestartet werden kann. Falls erforderlich, kann die Ausführung eines Prozesses geplant unterbrochen werden, so dass ein Prozess zur Laufzeit aus mehreren *Sices* bestehen kann.

4.1 Ein Beispielszenario

In diesem fiktiven Beispiel soll ein statischer Schedule für ein hypothetisches System (siehe Abbildung 2) entwickelt werden. Das System besteht aus zwei Prozessoren: einem Single-Core Prozessor *P.1* und einem Dual-Core Prozessor *P.2*. Beide Prozessoren sind miteinander über gemeinsam genutzten Speicher (*R.1*) verbunden. Dieser Speicher darf nur *exklusiv* von einem Prozessor pro Zeitslot verwendet werden.

Anwendung *A* liest Sensordaten ein und berechnet, ob bestimmte Aktuator-Aktionen notwendig sind. Anwendung *A* wird durch die Partition *A.1* implementiert, die fest dem Prozessor *P.1* zugeordnet ist. Die Partition *A.1* besteht



(a) Software Architektur (logische Sicht) (b) Abbildung der Software auf die Hardware

Abb. 1. Die Terminologie von PRECISION PRO.

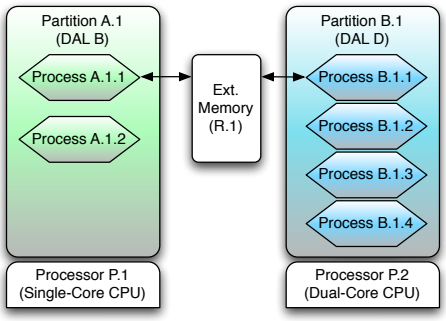


Abb. 2. Beispiel mit zwei Partitionen, sechs Prozessen, sowie einer externen Ressource

aus zwei Prozessen *A.1.1* und *A.1.2*. Ersterer holt die Sensordaten und stellt diese im Speicher *R.1* für andere Prozesse in anderen Partitionen zur Verfügung. Prozess *A.1.2* liest die neuen Sensordaten periodisch von *R.1* ein und führt die interne Auswertung der Daten durch.

Die Anwendung *B* besteht aus einer Partition (*B.1*), mit vier Prozessen *B.1.1* bis *B.1.4*. *B.1.1* holt die Daten aus *R.1* und stellt sie den Prozessen *B.1.2* bis *B.1.4* intern zur Verfügung. Die Partition *B.1* wird fest dem Dual-Core Prozessor *P.2* zugeordnet.

Weitere Echtzeitanforderungen sind in Tabelle 1 festgehalten.

Weiterhin gilt für dieses Beispiel:

- Immer wenn ein *Slicing* im Schedule auftritt, muss eine zusätzliche Verzögerung von $1\,ms$ im statischen Schedule einkalkuliert werden, damit der Kontextwechsel durchgeführt werden kann.
- Hier genügt es, die Granularität des Schedules auf der Basis von $1\,ms$ zu berechnen (*base scheduling unit*).

Prozess	**WCET** $[ms]$	**Periode** $[ms]$
A.1.1	5	20
A.1.2	10	40
B.1.1	5	20
B.1.2	11	20
B.1.3	7	40
B.1.4	3	20

Tabelle 1. Weitere Echtzeitanforderungen

4.2 Globale Anforderungen

Zunächst werden verschiedene globale Anforderungen des Schedules definiert. Dies betrifft die Wartezeiten bei *Slices*, die Länge der Hyperperiode und andere globale Eigenschaften. In dem Code-Beispiel 1.1 sind die Definitionen für das oben beschriebene Beispiel aufgeführt.

```
def_global
    scheduling_period(200),
    unit_based(1:ms),
    change_delay(1).
```

Listing 1.1. Globale Anforderungen

4.3 Hardware Architektur

Die Architektur der Hardware wird abstrakt modelliert. *Cores* werden als generische Ausführungseinheiten angesehen, die pro Zeit nur einen Task ausführen können. Dieser Ansatz adressiert homogene Multicore Prozessoren mit konstanter Taktrate und vernachlässigt Aspekte wie *Pipelining* oder *Speculative Execution*. Auch die Speicherhierarchie ist nicht Bestandteil des Modells, sondern implizit in den Ausführungszeiten der verschiedenen Prozesse enthalten. Weiterhin wird auch eine *Uniform Memory Architecture* angenommen, d.h. die Ausführungszeiten und Speicherzugriffszeiten sind unabhängig von den Cores. In PRECISION PRO können Prozessor*systeme* beschrieben werden, die aus Single- und Multicore Prozessoren bestehen. Damit können auch Abhängigkeiten zwischen Prozessen definiert werden, die auf unterschiedlichen Prozessoren ausgeführt werden. Das Code-Beispiel 1.2 enthält die Modellierung der Hardware Architektur für das besprochene Beispiel.

```
def_processor
    id('P.1'),
    cores(1).

def_processor
    id('P.2'),
    cores(2).
```

Listing 1.2. Die Hardware Architektur

4.4 Anwendungen und ihre zeitlichen Eigenschaften

PRECISION PRO legt ein einfaches Prozess Modell zugrunde. Jedem Prozess werden zeitliche Eigenschaften bei der Ausführung auf der modellierten Hardware zugeordnet (siehe Abbildung 3). Prozesse werden als *synchron* mit vorher bekannter *Periode* angenommen.

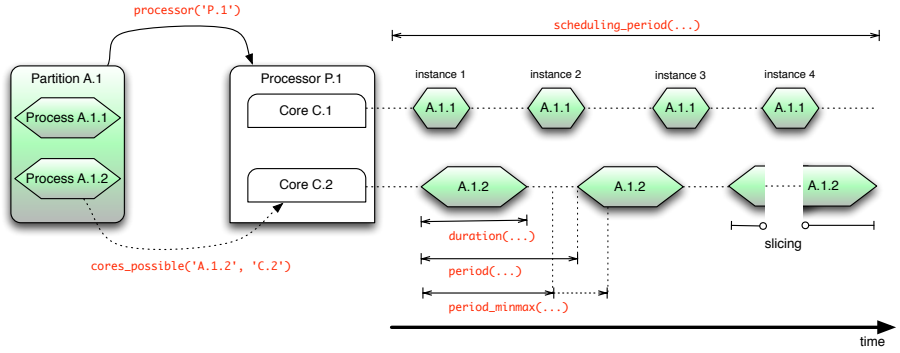


Abb. 3. Definition der Zuordnung von Partitionen zu Prozessoren und die Spezifikation von zeitlichen Anforderungen von Prozessen.

Die Spezifikation für die Partition *B.1* im Beispiel ist in Listing 1.3 dargestellt. Die Anzahl der parallelen Prozesse in einer Partition wird durch `parallel` konfiguriert. Zeitliche Anforderungen werden durch `duration` und `period` festgelegt. Falls diese Eigenschaften nicht für alle parallelen Prozesse gelten sollen, kann im ersten Argument die *Id* des gewünschten Prozesses angegeben werden.

```
def_partition
    id('B.1'),
    parallel(4),
    duration(1, 5), duration(2, 11), duration(3, 7), duration(4, 3),
    period(20), period(3, 40).
```

Listing 1.3. Modellierung der Software

4.5 Externe Ressourcen

Die Modellierung des Zugriff auf gemeinsam genutzte Ressource, wie Netzwerk Bandbreite oder der Zugriff auf I/O Geräte, ist eine wichtige Eigenschaft von PRECISION PRO, um ein zertifizierbares, statisches Scheduling auf Multicore Prozessoren zu erreichen (siehe [5]). Gemeinsam genutzte Ressourcen werden als abstrakte *Container* mit einer definierten Kapazität modelliert. Mit einer Kapazität von 1 verhalten sie sich *exklusiv*, während eine größere Kapazität zu einem kumulativen Verhalten führt. Exklusive Ressourcen können den Zugriff auf I/O Geräte abbilden, die keinen Mehrfachzugriff unterstützen. Mit kumulativen

Ressourcen werden häufig eher Netzwerkbandbreiten modelliert und damit in die Erstellung eines Schedules einbezogen.

Im Quellcode-Beispiel 1.4 ist die Definition für die Ressource *R.1* angegeben. Die Kapazität wird mit `max` spezifiziert. In diesem Beispiel wird auch die Nutzung von *R.1* durch *A.1.1* und *B.1.1* angegeben.

```
def_resource
   id('R.1'),
   max(1).

def_partition
  id('A.1'),
  resource(1, 'R.1', 1),
  ...

def_partition
   id('B.1'),
   resource(1, 'R.1', 1),
   ...
```

Listing 1.4. Gemeinsam genutzte Ressourcen

4.6 Zuordnung der Software auf die Hardware

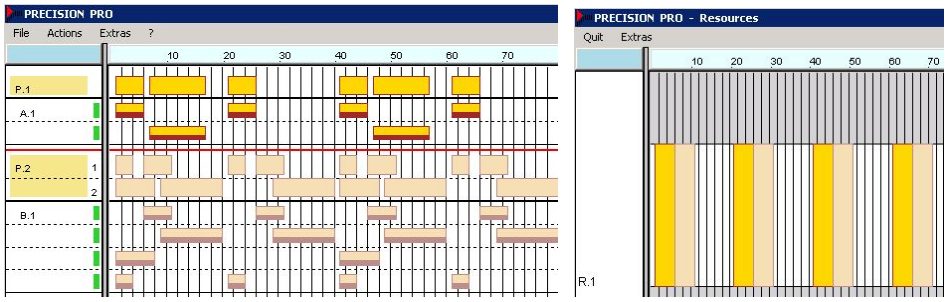
Die Suche nach einer korrekten und effizienten Zuordnung von Partitionen zu Prozessoren ist eine anspruchsvolle Aufgabe, die wesentlich die Suche nach gültigen Schedules beeinflusst. Aktuell muss die Zuordnung manuell vorgenommen werden (siehe Listing 1.5 für die Verteilung von Partition *A.1* auf Prozessor *P.1*), da neben kapazitiven Aspekte oft auch nicht-funktionale Aspekte bei der Verteilung von Funktionen eine große Rolle spielen. PRECISION PRO bietet allerdings eine automatische Verteilung von Prozessen einer Partition auf die Cores eines Prozessors an.

```
def_partition
   id('A.1'),
   processor('P.1'),
   ...
```

Listing 1.5. Verteilung der Software auf die Hardware

4.7 Ergebnis

Falls ein Schedule für die spezifizierten Anforderungen gefunden werden kann, wird dieser von PRECISION PRO in kurzer Zeit generiert - sogar für komplexe Systeme. Zur Generierung des Beispiels werden auf einem aktuellen PC etwa $1050\,ms$ benötigt. Das Ergebnis wird graphisch in einer Benutzeroberfläche dargestellt (siehe Abbildung 4a). Abhängigkeiten zwischen Prozessen werden durch Linien gekennzeichnet. Auch die Zugriffmuster auf die gemeinsam genutzten Ressourcen werden dargestellt (siehe Abbildung 4b). Diese Scheduling-Informationen können exportiert werden und anschließend für die Konfiguration eines Echtzeit-Betriebssystems verwendet werden.



(a) Der generierte Schedule (b) Zugriffe auf die Ressource

Abb. 4. Der erzeugte Schedule für das beschriebene Beispiel

5 Aktuelle Arbeiten: Generierung von Hierarchischen Schedules in der Automotive Domäne

Im VirtuOS Projekt [8] untersuchen die Autoren die Einsatzmöglichkeiten von Virtualisierungstechnologien in sicherheitsrelevanten Anwendungsfällen anhand eines konkreten Fallbeispiels aus dem Automobil-Bereich. Angenommen wird die Nutzung des automotiven Betriebssystems *COQOS* auf einem *Instrument-Cluster* zur Integration von Infotainment Applikationen (*"car-apps"*) und sicherheitskritischen Automotive-Funktionen.

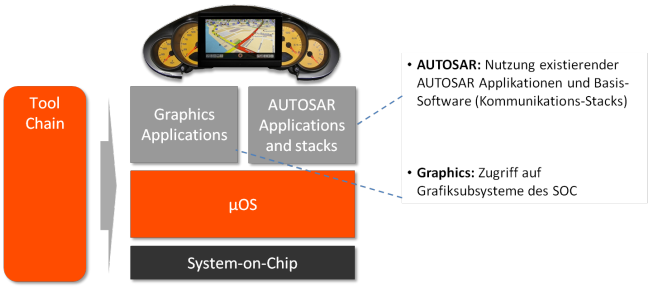


Abb. 5. Architektur der Fallstudie und wichtige Eigenschaften

Die Software-Architektur des *Instrument-Clusters* ist in Abbildung 5 dargestellt. Auf einer gemeinsamen Hardware-Plattform werden die verfügbaren Ressourcen mit Hilfe eines Mikrokernels in verschiedene Partitionen unterteilt. Die konfigurierten Partitionen sind dabei voneinander isoliert, so dass Fehler in einer Partitionen nicht die Ausführung von sicherheitskritischen Funktionen beeinflussen können.

Eine genaue Analyse der Architektur und der jeweiligen zeitlichen Anforderungen zeigt, dass der bisher präsentierte Ansatz zur Modellierung von Partitionen nicht ausreichend ist, um die zweistufige Scheduling-Hierarchie abzubilden. Die Infotainment Applikationen laufen in einer Partition, die insgesamt nur einen bestimmten Anteil der System-Ressourcen konsumieren darf. Diese Applikationen selbst sind jedoch nicht sicherheitskritisch und werden dynamisch gescheduled. In der AUTOSAR-Partition laufen ebenfalls verschiedene Prozesse (*AUTOSAR Runnables*), die jedoch harten Echtzeitanforderungen unterliegen und spezifizierte Deadlines zu erfüllen haben.

Aktuell arbeiten die Autoren daher an einer Erweiterung des beschriebenen Modellierungskonzeptes. Ziel ist es, die Erstellung von statischen Schedules zu ermöglichen, die sowohl eine feste Aufteilung der Systemressourcen sicherstellen, aber auch die Deadlines von Prozessen innerhalb einer Partition berücksichtigen. Gleichzeitig müssen auch die Umschaltzeiten zwischen den Partitionen minimiert werden, um den resultierenden Overhead zu reduzieren.

Acknowledgements

Teile dieser Arbeit wurden im Rahmen des VirtuOS Projektes erarbeitet. Das VirtuOS Projekt wird durch die TSB Technologiestiftung Berlin aus Mitteln des Zukunftsfonds des Landes Berlin gefördert, kofinanziert von der Europäischen Union - Europäischer Fonds für Regionale Entwicklung. Investition in Ihre Zukunft!

Literaturverzeichnis

1. ARINC. ARINC Specification 653P1-2: Avionics Application Software Standard Interface Part 1 - Required Services. Technical report, Aeronautical Radio Inc., Maryland, USA, December 2005.
2. ARINC. ARINC Specification 653P2-1: Avionics Application Software Standard Interface Part 2 - Extended Services. Technical report, Aeronautical Radio Inc., Maryland, USA, December 2008.
3. Vicent Brocaly, Miguel Masmanoy, Ismael Ripolly, Alfons Crespoy, Patricia Balbastrey, and Jean-Jacques Metge. Xoncrete: a scheduling tool for partitioned real-time systems. In *Proceedings of the Embedded Real Time Software and Systems Conference (ERTS^2 2010)*, May 2010.
4. Gerhard Fohler. Flexible Reliable Timing - Real-Time vs. Reliability. In *Keynot Address, 10th European Workshop on Dependable Computing*, May 1999.
5. R. Fuchsen. How to address certification for multi-core based IMA platforms: Current status and potential solutions. In *Digital Avionics Systems Conference (DASC), 2010 IEEE/AIAA 29th*, pages 5.E.3–1 –5.E.3–11, oct. 2010.
6. Manne Kreuzer. Modellbasiert die Echtzeit im Griff. *Markt & Technik - die unabhängige Wochenzeitung*, Nr. 24:35, Jun 2011.
7. John Rushby. Partitioning in Avionics Architectures: Requirements, Mechanisms, and Assurance, 1999.
8. VirtuOS Project Summary. http://www.technologiestiftung-berlin.de/data/files/tsb-zukunftsfonds/VirtuOS__Web_.pdf, 2010.