React in Time: Ereignisbasierter Entwurf zeitgesteuerter verteilter Systeme*

Florian Franzmann[1], Tobias Klaus[1], Fabian Scheler[2],
Wolfgang Schröder-Preikschat[1] und Peter Ulbrich[1]

[1] Lehrstuhl für Verteilte Systeme und Betriebssysteme
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Erlangen
{franzmann|klaus|wosch|ulbrich}@cs.fau.de
[2] Process Industries and Drives
Siemens AG, Nürnberg fabian.scheler@siemens.com

Zusammenfassung. Die Analyse und Verifikation von Echtzeitsystemen profitiert von einer zeitgesteuerten Implementierung. Jedoch erweist sich aus Entwicklersicht der ereignisorientierte Entwurf als deutlich flexiblerer und einfacher. Diese Arbeit stellt einen Ansatz zur automatisierten Analyse von existierenden Echtzeitsystemen, deren Überführung in eine abstrakte Zwischendarstellung sowie die anschließende Abbildung auf zeitgesteuerte (verteilte) Mehrkernsysteme vor.

1 Einleitung

Entwurfsentscheidungen haben typischerweise starke Auswirkungen auf das zeitliche Verhalten (Termintreue, Latenzen, Jitter, ...) des zu entwickelnden Echtzeitsystems. Neben der Abbildung der eigentlichen Echtzeitanwendung auf Arbeitsaufträge und deren Koordinierung, betrifft dies vor allem die Wahl eines der beiden Echtzeitparadigmen sowie eines geeigneten Echtzeitbetriebssystems. Gerade diese frühen Entwurfsentscheidungen bestimmen den weiteren Verlauf der Entwicklung und die Eigenschaften des Gesamtsystems maßgeblich.

Die Entscheidung zwischen dem *ereignis-* und dem *zeitgesteuerten* Paradigma kann eine philosophische sein. Unbestreitbar haben jedoch beide Ansätze ihre individuellen Vor- und Nachteile: Zeitgesteuerte Echtzeitsysteme bieten beispielsweise einen konstruktiven Ansatz für die Verifikation des Systems. Ein gültiger Ablaufplan impliziert bereits die korrekte Erfüllung der zeitlichen Anforderungen wie auch der Abhängigkeiten zwischen Arbeitsaufträgen. Durch die vorgezogene Ablaufplanung entfallen zudem die sonst notwendige komplexe Laufzeitumgebung und Ereignisbehandlung, wodurch letztlich auch die statische Analyse und Vorhersage von nichtfunktionalen Eigenschaften, insbesondere der *maximale Ausführungszeit* (WCET) des Systems, signifikant erleichtert wird. Durch das

* Diese Arbeit wurde durch die Deutsche Forschungsgesellschaft (DFG) unter Nr. SCHR 603/9-1 und dem Bayrischen Staatsministerium für Wirtschaft (EU EFRE) unter Nr. 0704/883 25 gefördert. Autoren erscheinen in alphabetischer Reihenfolge.

für die statische Ablaufplanung notwendige Vorabwissen stellt das zeitgesteuerte Paradigma hingegen typischerweise höhere Anforderungen an den Systementwurf, da kein expliziter Zusammenhang zwischen Ereignis und Behandlung existiert. Im Gegensatz dazu orientiert sich ereignisbasierte Entwurf vorrangig an der Ereignisbehandlung und ist daher typischerweise intuitiver in seiner Umsetzung. Durch die Ablaufplanung und Koordinierung zu Laufzeit ist zudem weniger Vorabwissen notwendig. Die entstehende Flexibilität wirkt sich jedoch nachteilig auf die Vorhersagbarkeit und Verifizierbarkeit aus. Analog zu der Wahl des Echtzeitparadigmas haben auch das Echtzeitbetriebssystem beziehungsweise die Ausführungsplattform einen signifikanten Einfluss auf die nichtfunktionalen Eigenschaften des Echtzeitsystems.

Einen wesentlichen Aspekt dieser Problematik stellen die weitreichenden Auswirkungen und die praktische Unumkehrbarkeit initial getroffener Entwurfsentscheidungen dar: Ein ereignisgesteuertes System lässt sich beispielsweise nicht ohne weiteres in ein zeitgesteuertes System überführen und umgekehrt. Die zuvor genannten Vorteile der einfachen Verifizierbarkeit beziehungsweise des intuitiven und flexiblen Entwurfs schließen sich somit praktisch gegenseitig aus.

Von diesem Standpunkt aus betrachtet ist eine Entkopplung der Entwurfsentscheidungen und ihren Auswirkungen auf die temporalen beziehungsweise nichtfunktionalen Eigenschaften im Allgemeinen wünschenswert. Eine mögliche Lösung liegt in der ereignisbasierten Entwicklung und anschließenden Abbildung auf eine zeitgesteuerte Laufzeitumgebung. Die sich hieraus ergebene Forschungsfrage bezieht sich auf eine Transformation von Echtzeitsystemen unter Beibehaltung der temporalen Invarianten (Abhängigkeiten und Termine). Diese Arbeit widmet sich den hierfür notwendigen Ansätzen zur Analyse und abstrakte Darstellung von Echtzeitsystemen, der Manipulation von nichtfunktionalen Eigenschaften sowie deren Abbildung auf eine gewünschte Zielplattform. In Kapitel 2 wird hierfür das Konzept der *Atomaren Basisblöcke* (ABB) sowie der *Real-Time Systems Compiler* (RTSC) zur automatisierten Analyse von ereignisgesteuerten Echtzeitsystemen eingeführt. Darauf aufbauend widmet sich Kapitel 3 der Abbildung und Kolokalisierung von ereignisgesteuerten Echtzeitsystemen auf ein zeitgesteuertes Mehrkernsystem. Schließlich gibt Kapitel 4 einen Ausblick auf die Problemstellungen und Lösungsvorschläge für eine Verteilung der Echtzeitanwendung in einem zeitgesteuerten verteilten System.

2 Abstrakte Darstellung von Echtzeitsystemen

Um auf Echtzeitsystemen Transformation ausführen zu können, die deren Invarianten – insbesondere den gerichteten und ungerichteten Abhängigkeiten sowie den Terminen – nicht verletzen, sind diese zunächst in eine abstrakte *Zwischendarstellung* zu überführen, welche die zugrundeliegenden strukturellen Eigenschaften des Echtzeitsystems vollständig erfasst. Hierzu muss das Echtzeitsystem in *Atomare Basisblöcke* zerlegt werden, die sich von Synchronisationspunkt zu Synchronisationspunkt erstrecken und so die Interaktion der Echtzeitanwendung mit dem Betriebssystem erfassen. Jeder dieser ABBs setzt sich aus meh-

reren *Basisblöcken* zusammen. Diese wird aus dem Quellcode der Anwendung durch ein Übersetzerwerkzeug extrahiert und mit Hilfe eines *Systemmodells* um zeitliche Informationen und die Besonderheiten des verwendeten *Quellbetriebssystems* ergänzt. ABBs sind durch implizite Abhängigkeiten miteinander verbunden, die in Form gerichteter Kanten den aus den Basisblöcken extrahierten *lokalen Kontrollflussgraphen* nachzeichnen. Explizite Abhängigkeiten, im Quellsystem noch durch Systemaufrufe des Echtzeitbetriebssystems dargestellt, verbinden diese lokalen *Atomarer Basisblock* (ABB)-Graphen zum *globalen Atomarer Basisblock* (ABB)-Graph, der das Echtzeitsystem abstrakt und ohne Abhängigkeiten zum ursprünglichen Quellbetriebssystems beschreibt.

Ein Teil dieses Beitrags ist der *Real-Time Systems Compiler*, ein übersetzerbasiertes Werkzeug, das *Atomarer Basisblock* (ABB)-Graphen aus dem Quellcode der Anwendungssoftware extrahiert und Transformationen auf diesem vornimmt. Ergebnis der Transformation ist ein zeitlich analysiertes, vollständiges und auf der gegebenen Zielarchitektur ausführbares Echtzeitsystem. Ein Beispiel hierfür ist die Umwandlung eines ereignisgesteuerten in ein zeitgesteuertes Echtzeitsystem unter Erhaltung der relevanten Invarianten. Der *Real-Time Systems Compiler* (RTSC) erkennt hierbei auch, ob das Echtzeitsystem tatsächlich planbar ist. Eine Aussage die aus der ereignisgesteuerten Implementierung nicht ohne weiteres hervorgeht.

3 Multicore

Ein Trend der jüngeren Vergangenheit ist der verbreitete Einsatz von Mehrkernarchitekturen auch in sicherheitskritischen Einsatzbereichen. In der Automobiltechnik werden zum Beispiel bislang für sich stehende Echtzeitanwendungen zunehmend auf leistungsfähigere Steuergeräte konsolidiert um Gewicht und Kosten einzusparen. Hierbei ergibt sich das Problem, dass Anwendungen mit unterschiedlichen Anforderungen bezüglich funktionaler Sicherheit nunmehr gemeinsam auf dem selben Rechenknoten zur Ausführung kommen. In der Folge ergibt sich die Sicherheitseinstufung des Steuergeräts aus der kritischsten Anwendung, mit entsprechend negativen Auswirkungen hinsichtlich des Entwicklungsaufwands, der Zertifizierung und der hierdurch entstehenden Kosten. Alternativ ist eine strikte temporale und räumliche Isolation der Anwendungen untereinander erforderlich. Neben der gängigen, jedoch ressourcenintensiven, Partitionierung des Systems, eignet sich hier insbesondere die taktgesteuerte Umsetzung um einerseits die notwendige zeitlichen Abschottung der Anwendungen untereinander zu garantieren und andererseits die zeitliche Verifizierbarkeit des Gesamtsystems zu verbessern.

An dieser Stelle spielt der *Real-Time Systems Compiler* (RTSC) seine Vorteile aus. Er erlaubt die Analyse auch von bereits existierenden ereignisgesteuerten Echtzeitsystemen sowie deren Abbildung auf zeitgesteuerte Echtzeitsysteme. Räumliche Isolation kann z. B. durch Speicherschutzmassnahmen erreicht werden, die ebenfalls vom *Real-Time Systems Compiler* (RTSC) eingewebt werden könnten.

Die vom *Real-Time Systems Compiler* (RTSC) verwendete Zwischendarstellung erleichtert es, Echtzeitanwendungen auf eine Mehrkernarchitektur abzubilden. Das Systemmodell des *Real-Time Systems Compiler* (RTSC) enthält jegliche Information, die für eine *rechtzeitige* Ausführung aller Aufgaben erforderlichen ist (Termine, Perioden, minimale Zwischenankunftzeiten, zeitliche Schwankung), während die *Atomarer Basisblock* (ABB)-Graphen die Abhängigkeiten zwischen einzelnen Aufgaben (gerichtet sowie gegenseitiger Ausschluss) repräsentieren und so den globalen Kontrollflussgraph abbilden.

Zwar erzeugt die Zerlegung in ABBs eine Last, die prinzipiell planbar ist, jedoch stellt der *Real-Time Systems Compiler* (RTSC) noch nicht alle Mittel bereit, die für die Ablaufplanung für Mehrkernsysteme erforderlich sind. Die Auswahl dieser Mittel und ihre Anpassung an die besonderen Gegebenheiten des *Real-Time Systems Compiler* (RTSC) sind Thema dieses Abschnitts. Das Ergebnis ist der *Real-Time Systems Compiler for Multicore* (RTSC:MP).

3.1 Forschungsfrage

Das Ziel des *Real-Time Systems Compiler for Multicore* (RTSC:MP) ist es, eine Last auf einem Mehrkernsystem einzuplanen, die aus mehreren wechselseitig abhängigen Aufgaben besteht. Hierzu sind folgende Bausteine notwendig:

1. Ein Algorithmus, der entscheidet, welcher *Atomarer Basisblock* (ABB) auf welchem Prozessorkern ausgeführt wird. Diese Entscheidung muss so getroffen werden, dass es möglich ist, einen *durchführbaren* Ablaufplan für diese Zuteilung zu finden, d. h. alle Termine müssen eingehalten werden. Der Algorithmus sollte in der Lage sein, die durch die gerichteten und ungerichteten Abhängigkeiten bestimmte Rangordnung der ABBs bei der Zuteilungsentscheidung zu berücksichtigen.
2. Ein Ablaufplanungsalgorithmus, der mit Einzel- und Mehrkernsystemen umgehen kann, und der sich in Zukunft so erweitern lässt, dass die Last auch auf verteilte Systeme abgebildet werden kann.

Beide Algorithmen sollen außerdem für das im *Real-Time Systems Compiler* (RTSC) verwendete Systemmodell geeignet sein oder die Zwischendarstellung des *Real-Time Systems Compiler* (RTSC) sollte entsprechend der Erwartungen der Algorithmen anpassbar sein. Sowohl das Zuweisungs- als auch das Planungsproblem sind NP-hart, weswegen es wichtig ist, dass die Algorithmen akzeptables Verhalten in Bezug auf Zeit- und Speicherbedarf aufweisen. Hierbei ist von Interesse welche Anpassungen am *Real-Time Systems Compiler* (RTSC) und den Algorithmen notwendig sind damit diese hier eingesetzt werden können.

3.2 Theoretischer Hintergrund

Die in der Literatur beschriebenen Algorithmen sind zum Großteil entweder nicht optimal[1] [3] oder sehen nicht vor, dass das Echtzeitsystem Abhängigkeiten enthält [2, 4, 6]. Globales oder gebündeltes *Earliest Deadline First* (EDF)

[1] Sofern eine durchführbare Lösung für das Ablaufplanungs- bzw. Zuweisungsproblem existiert, wird diese durch den Algorithmus stets gefunden.

sind partitioniertem *Earliest Deadline First* (EDF) in Bezug auf die Planbarkeit unterlegen, solange keine Abhängigkeiten vorhanden sind. [2] Um dem Entwickler zuverlässige Rückmeldung geben zu können ist es erforderlich optimale Zuweisungs- und Ablaufplanungsalgorithmen zu verwenden, auch wenn dies bedeutet, dass die Laufzeit und der Arbeitsspeicherbedarf im Vergleich zu Heuristiken größer ist. Deswegen kommt im *Real-Time Systems Compiler* (RTSC) der Zuweisungsalgorithmus von Peng et al. [5] und der Ablaufplanungsalgorithmus von Abdelzaher et al. [**?**] zum Einsatz. Beide Algorithmen erreichen ihre Optimalität dadurch, dass sie nach dem Prinzip der *Verzweigen-und-Begrenzen-Suche* (B&B) funktionieren.

Zuweisung: Der im *Real-Time Systems Compiler* (RTSC) verwendete Zuweisungsalgorithmus erzeugt zunächst eine leere Lösung. Von dieser werden dann nach und nach verfeinerte Lösungen abgeleitet, indem jeweils ein weiterer *Atomarer Basisblock* (ABB) den Prozessoren zugewiesen wird. Innere Knoten des Suchbaums sind somit noch keine Kandidaten für eine vollständige Zuweisung, da erst in den Blatt-Knoten jeder *Atomarer Basisblock* (ABB) genau einem Prozessorkern zugewiesen ist. Als Kostenfunktion zur Bewertung der Lösungen dient die *Normalisierte Aufgaben-Antwortzeit* (NTRT), die sich aus dem Zeitpunkt der vollständigen Abarbeitung c, dem Auslösezeitpunkt r und dem absoluten Termin d zu $\bar{c} := \frac{c-r}{d-r}$ berechnet. Bei der Ermittlung der Kosten einer Lösung passt der Zuweisungsalgorithmus zunächst die Auslösezeiten aller Arbeitsaufträge an, so dass die Rangordnung eingehalten wird. Anschliessend werden die Arbeitsaufträge in nicht-absteigender Reihenfolge ihrer modifizierten Auslösezeiten angeordnet, so dass disjunkte Blöcke entstehen. Nun wird derjenige Arbeitsauftrag, der die geringsten Kosten in Bezug auf die *Normalisierte Aufgaben-Antwortzeit* (NTRT) verursacht entfernt und dessen Kosten erfasst. Die verbleibenden Arbeitsaufträge werden gemäß ihrer modifizierten Auslösezeit unter Beachtung der Rangfolge neu angeordnet. Diese Schritte werden so lange wiederholt, bis keine Arbeitsaufträge mehr verbleiben. Die Kosten einer Lösung lassen sich als das Maximum der Kosten aller Arbeitsaufträge ermitteln. Zusätzlich müssen jedoch auch noch die Kosten der Arbeitsaufträge mit berücksichtigt werden, die noch nicht zugewiesen worden sind. Für diese wird eine Abschätzung für die obere Grenze der verursachten Kosten ermittelt und denen der Lösung zugeschlagen.

Ablaufplanung: Im Unterschied zum Zuweisungsalgorithmus stellt die initiale Lösung des Ablaufplanungsalgorithmus bereits eine *vollständige* Lösung für das *globale Ablaufplanungsproblem* dar. Diese initiale Lösung respektiert zwar schon die gerichteten und ungerichteten Abhängigkeiten, hält jedoch nicht notwendigerweise alle Termine ein. Werden jedoch alle Termine eingehalten, so ist die Suche an dieser Stelle abgeschlossen. Andernfalls werden nach und nach verfeinerte Lösungen abgeleitet, indem Arbeitsaufträge, die ihren Termin nicht einhalten weiter nach vorne im Ablaufplan verschoben werden. Dies wird erreicht, indem zusätzliche gerichtete Abhängigkeiten in das Aufgabensystem eingefügt werden. Einzelne Lösungen des Suchbaums werden erzeugt, indem die gesamte

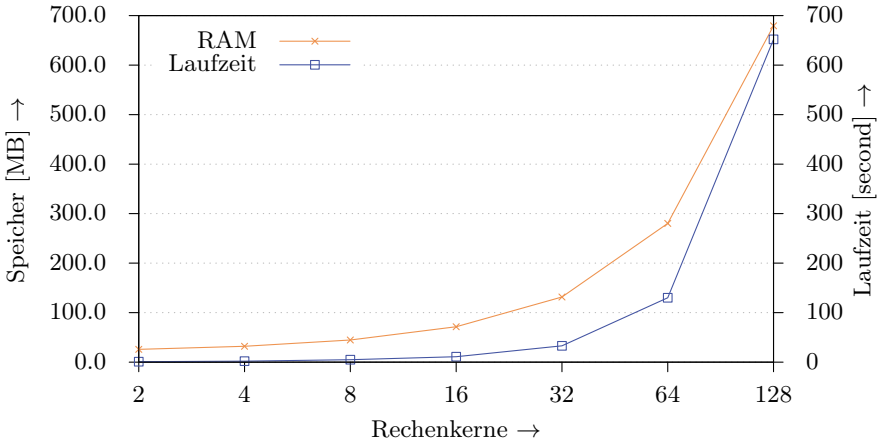

Abb. 1. Anstieg des Ressourcenverbrauchs mit der Anzahl der Kerne des Zielsystems.

Last mit Hilfe des *Earliest Deadline First with Deadline Inheritance* (EDF-DI)-Algorithmus eingeplant wird. Als Kostenfunktion zum Vergleich der einzelnen Lösungen dient hierbei die maximale Verspätung aller Arbeitsaufträge einer Lösung. Bei *Verzweigen-und-Begrenzen-Suche* (B&B)-Algorithmen ist es erforderlich, eine untere Grenze für die Kosten aller ableitbaren Lösungen zu bestimmen. Diese Abschätzung wird verwendet um festzulegen, welche Teilbäume nicht weiter erkundet werden müssen. Im konkreten Fall wird diese untere Grenze errechnet, indem ein vereinfachtes Ablaufplanungsproblem, ohne Abhängigkeiten gelöst wird. Unter diesen Umständen ist der *Earliest Deadline First* (EDF)-Algorithmus optimal und es ist unmöglich, dass das Aufgabensystem, das die Abhängigkeiten berücksichtigt, geringere Kosten verursacht.

3.3 Ansatz

Aus Sicht des Echtzeitsystementwurfs erscheint es auf der einen Seite erstrebenswert das System in möglichst feinkörnige Bestandteile zu zerlegen, die dann zugewiesen und eingeplant werden. Dieser Bestrebung kommt das *Atomarer Basisblock* (ABB)-Konzept entgegen. Jedoch führt diese feinkörnige Zerlegung auch zu schwerwiegenden Problemen: Der sich aus den ABBs ergebende Suchraum ist deutlich größer als derjenige, für den die verwendeten Algorithmen ursprünglich entworfen wurden. Da der Suchraum exponentiell mit der Anzahl der ABBs wächst, besteht die Gefahr, dass bereits bei mittelgroßen Echtzeitsystemen keine durchführbare Lösung mehr in akzeptabler Zeit und mit vertretbaren Betriebsmitteln gefunden werden kann. Mit der Nutzung von Vielkernsystemen wird dieses Problem weiter verschärft. Abbildung 1 zeigt den polynomiellen Zuwachs des Betriebsmittelbedarf beim Abbilden desselben sehr einfachen, also aus wenigen ABBs bestehenden Ausgangssystems auf Systeme mit steigender Anzahl von Kernen.

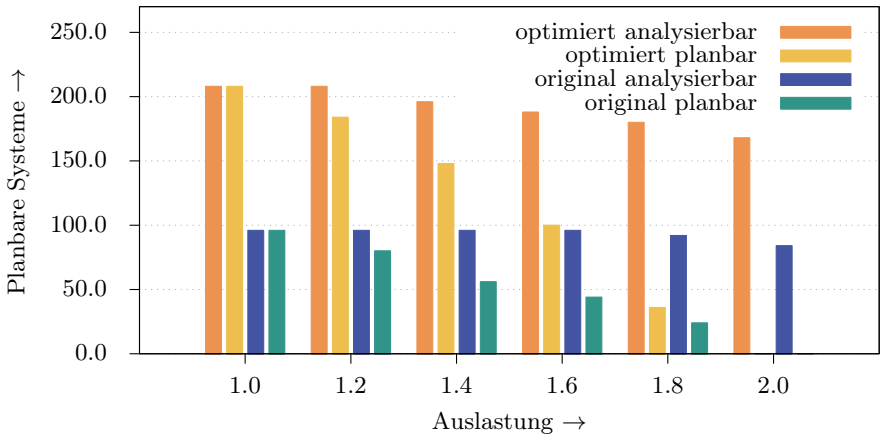

Abb. 2. Die Auswirkung der Optimierungen: Je Auslastungstufe wurden 336 zufällige Ausgangsysteme einmal mit dem originalen Algorithmus und dann mit unserer optimierten Version analysiert und transformiert. Beide Algorithmen hatten jeweils maximal 5,5 GB Arbeitsspeicher zur Verfügung. Sowohl die Anzahl der analysierbaren, als auch die unter Einhaltung aller Termine planbaren Systeme konnten durch unsere Optimierungen teilweise verdoppelt werden. Zusätzlich konnte durch die Optimierungen auch die Laufzeit von durchschnittlich 1794 s auf 852 s mehr als halbiert werden.

Komplexität: Aus diesem Grund waren verschiedene Optimierungen bei der Integration der Algorithmen in den *Real-Time Systems Compiler for Multicore* (RTSC:MP) notwendig. Hierbei wurde darauf geachtet die Optimalität der Algorithmen nicht zu verletzen. Der Ansatzpunkt der Optimierung ist die Ersetzung bislang nicht spezifizierter und daher zufälliger Entscheidungsprozesse in den Kostenfunktionen durch systematische und schneller terminierendere Funktionen. Im Falle des Zuweisungsalgorithmus heißt dies, dass bei ansonsten gleichen Kosten diejenige Lösung bevorzugt wird, in der bereits mehr ABBs zugewiesen sind. Schlimmstenfalls könnte eine zufällige Entscheidung an dieser Stelle dazu führen, dass immer die Lösung bevorzugt wird, die weiter von der Terminierung entfernt ist. Dies würde dazu führen, dass ohne Notwendigkeit weite Teile des Suchbaums erforscht werden. Abbildung 2 zeigt den Effekt dieser Optimierungen. Unter der Annahme eines auf 5,5 GB beschränkten Arbeitsspeichers konnte die Zahl der analysierten und planbaren Systeme teilweise verdoppelt werden. Auch die durchschnittliche Laufzeit sank von 1794 s auf 852 s um mehr als die Hälfte.

Abbildung von Modell auf Modell: Der Zuweisungsalgorithmus von Peng et al. berücksichtigt gerichtete Abhängigkeiten, indem Auslösezeitpunkte so verschoben werden, dass alle Abhängigkeiten eingehalten werden. Explizite Abhängigkeiten werden folglich in eine implizite Form umgewandelt. Dies ist jedoch für die Last, die der *Real-Time Systems Compiler* (RTSC) dem Algorithmus als Eingabe übergibt, nicht hinreichend. Im *Real-Time Systems Compiler* (RTSC) erben alle zu einer Aufgabe gehörenden Arbeitsaufträge, und somit auch deren ABBs, den

Termin der Aufgabe. Da dieser jedoch mit in die Berechnung der Kosten einer Lösung eingeht, werden hierdurch Lösungen, die eigentlich unterschiedliche Kosten verursachen sollten, ununterscheidbar. Dies führt dazu, dass weite Teile des Suchraums erkundet werden, ohne dass dies notwendig ist. Schlimmer noch, die Kostenfunktion verliert ihre Monotonieeigenschaft, die für die Korrektheit des Algorithmus zwingend erforderlich ist. Deswegen wurde der Zuweisungsalgorithmus so angepasst, dass zusätzlich auch Termine entlang der Abhängigkeiten der ABBs verschoben werden, so dass diese bereits durch die Auslösezeitpunkte zum Ausdruck kommen.

Einbeziehung der Ausführungsumgebung: Die bis hier durchgeführten Maßnahmen ermöglichen es bereits, den Zuweisungs- und den Ablaufplanungsalgorithmus im *Real-Time Systems Compiler for Multicore* (RTSC:MP) zu verwenden. Jedoch bringt die Verwendung des *Real-Time Systems Compiler for Multicore* (RTSC:MP) für echte Hardware weitere Herausforderungen mit sich: Der Ablaufplanungsalgorithmus wandelt in seiner ursprünglichen Form explizite Abhängigkeiten in verschobene Termine und Auslösezeitpunkte um, was dem Verhalten des Zuweisungsalgorithmus entspricht. Im Falle der endgültigen Ablaufplanung ist diese Herangehensweise jedoch nicht zweckdienlich. Zwar würden auch so alle Abhängigkeiten eingehalten und es würde ein System entstehen, das alle Termine einhält. Das Verschieben von Zeiten zur Einhaltung von Abhängigkeiten begünstigt jedoch Kontextwechsel, was unter Umständen nicht hinnehmbare Verwaltungsgemeinkosten bedingt. Deswegen werden ABBs im *Real-Time Systems Compiler for Multicore* (RTSC:MP) in einer expliziten „Bereit"-Warteschlange verwaltet, was die Abhängigkeiten durchsetzt, ohne dass Zeiten verschoben werden müssen. Dies ermöglicht es in Situationen, in denen mehrere ABBs die gleiche Kosten verursachen so zu entscheiden, dass ABBs bevorzugt werden, die keinen Kontextwechsel verursachen. Auch im Zuweisungsalgorithmus musste eine Anpassung zur Vermeidung von Kontextwechseln vorgenommen werden: Die Kostenfunktion wurde dahingehend erweitert, dass, wenn zwei Lösungen in Bezug auf die bisherige Kostenfunktion gleichwertig erscheinen, diejenige Lösung bevorzugt wird, die weniger Kontextwechsel verursacht. Weiterhin werden im Ablaufplanungsalgorithmus ABBs, die an einem Kontextwechsel beteiligt sind, mit einem Malus auf ihre *maximale Ausführungszeit* (WCET) beaufschlagt.

4 Verteilte Systeme

Ein weiterer Schritt in der Entwicklung von Echtzeitsystemen stellt die Verteilung von Anwendungen über die Grenzen von Rechenknoten hinweg dar. Ein aktuelles Beispiel liefert Airbus [7] mit der Verteilung einer bestehenden und für ein leistungsfähiges Rechensystem entwickelten Fluglageregelungsanwendung. Diese wurde für ein großen Passagierflugzeug entwickelt, soll jedoch in kleineren Produkten wiederverwendet werden. Da in diesen Zielprodukten allerdings nicht die gleichen leistungsfähigen Rechensysteme eingebaut werden können, müssen die Teile der Anwendung die wiederverwendet werden effizient auf mehrere leistungsschwächere Recheneinheiten mit einer vom Ausgangssystem verschiedener

Architektur verteilt werden. Dieses Beispiel zeigt auch, dass die Probleme bei der Verteilung nicht nur ereignisgesteuerte Systeme betreffen sondern eben auch schon zeitgesteuert entwickelte Anwendungen betreffen.

Auch in einem solchen Szenario beeinflussen die Entscheidungen, die beim Entwurf der Flugregelung gemacht wurden, stark die Möglichkeiten der Weiterverwendung und Verteilung, obwohl die eigentliche Implementierung der Algorithmen unabhängig von der Ausführungsplattform des Systems erfolgen sollte. Zusätzlich rückt eine neue Frage in den Fokus: An welchen Stellen kann und sollte die bestehende Anwendung aufgeteilt und damit verteilt werden.

Um auch dieses Problem mithilfe der Darstellung mit *Atomarer Basisblock* (ABB)-Graphen zu lösen, muss der *Real-Time Systems Compiler for Multicore* (RTSC:MP) geeignet zum *Real-Time Systems Compiler for Distributed Systems* (RTSC:DS) erweitert werden.

Modell für das Zielsystem: Bisher modelliert der *Real-Time Systems Compiler for Multicore* (RTSC:MP) die Zusammensetzung des Zielsystems sehr einfach. Da das Zielsystem nur ein Rechensystem mit mehreren symmetrischen Knoten ist, wird das Zielsystem lediglich durch die Zielarchitektur und die Anzahl der Kerne abgebildet. Um in Zukunft auch verteilte Systeme betrachten zu können, muss der *Real-Time Systems Compiler for Distributed Systems* (RTSC:DS) auch komplexe Netzwerktopolgien abgebildet werden. Diese können aus mehreren Teilnetzen bestehen, die wiederum unterschiedliche Netzwerkprotokolle nutzen. Solche Netzwerke sind beispielsweise in Automobilen schon weitverbreitet. Hier werden Netzwerkprotokolle wie LIN, CAN oder Flexray gemeinsam genutzt und über Netzübergangseinheiten miteinander verbunden. Um die zeitlichen Parameter der Kommunikation auch nach den Transformationen des *Real-Time Systems Compiler* (RTSC) feststellen zu können müssen die Eigenschaften, insbesondere zeitliche, aller unterstützten Netzwerkprotokolle innerhalb des *Real-Time Systems Compiler* (RTSC) nachgebildet werden. Zusätzlich muss bei der Zuweisung von Aufgaben auf die Rechenknoten die angeschlossene Peripherie beachtet werden.

Aufteilung von Anwendungen: Bei der Verteilung bestehender Funktionen auf mehrere Knoten kommt der sinnvollen Aufteilung in einzelne Einplanungs- und Allokationseinheiten eine größere Bedeutung zu als bisher. Ungeschicktes Aufteilen und Verteilen von Codeblöcken mit starker Bindung verursacht jetzt nicht nur zusätzliche Speicherzugriffe, sondern belastet auch das globale Kommunikationsnetz und beeinflusst damit Funktionen, die auf allen anderen Knoten eingeplant werden. Dies kann zur überlast des Netzwerks und damit der Unplanbarkeit des kompletten Systems führen. Um eine solche ungeschickte Aufteilung zu verhindern müssen erweiterte Analysen auf den zu planenden Anwendungen ausgeführt und auch die implizite Kommunikation von Anwendungsteilen untereinander betrachtet werden. Damit können Anwendungen dann statt wie bisher nur an Synchronisationspunkten, auch an Stellen, die Kommunikation minimieren, aufgeteilt werden.

Extraktion von anwendungsübergreifenden Abhängigkeiten: Bisher extrahiert der *Real-Time Systems Compiler for Multicore* (RTSC:MP) die Abhängigkeiten innerhalb einer Anwendung und zwischen dem Betriebssystem. Werden jedoch Anwendungen zunächst konsolidiert und anschließend wieder auf mehrere Knoten verteilt, ergeben sich neue Optimierungsmöglichkeiten für die Kommunikation. Anwendungen und Anwendungsteile, die auf dasselbe Datum zugreifen oder die in einer, bisher netzwerkbasierten, ProduzentKonsument Beziehung stehen, können auf die selben Knoten zugewiesen werden und so zur Entlastung des Netzwerks beitragen. Um solche Allokationsoptimierungen umsetzen zu können, müssen jedoch zunächst anwendungsübereifende Kommunikationsabhängigkeiten aus den Anwendungen extrahiert und der jeweilige Kommunikationspartner festgestellt werden. Hierfür muss die schon bestehende Extraktion von *lokalen* Abhängigkeiten um ein Verständnis der verbreiteten Kommunikationsstacks erweitert werden und die gewonnenen Abhängigkeiten so aufbereitet werden, dass die beschriebenen Analysen zur Aufteilung von Anwendungen davon profitieren können.

5 Abschluss

Die starke Kopplung zwischen Implementierungsentscheidungen und nichtfunktionalen Eigenschaften tritt gerade bei den meist ressourcenbeschränkten Echtzeitsystemen hervor. Verstärkt wird dies durch ihre Einordnung in die Paradigmen *ereignisgesteuert* und *zeitgesteuert*, die sehr früh in der Entwicklung stattfindet und beträchtlichen Einfluss auf die Weiterentwicklung hat. Die abstrakte Darstellung von Echtzeitsystemen durch *Atomarer Basisblock* (ABB)-Graphen bietet eine Möglichkeit diese Entscheidungen später im Entwicklungsprozess treffen zu können. Verwendung finden *Atomarer Basisblock* (ABB)-Graphen in dem übersetzerbasierten Werkzeug *Real-Time Systems Compiler* (RTSC), der verschiedene Analysen, Transformationen und die Abbildung auf konkrete Zielhardware ausführt.

Dieser Artikel beschreibt zunächst die prinzipielle Abbildung von Echtzeitsystemen durch *Atomarer Basisblock* (ABB)-Graphen, um anschließend auf die Erweiterung des *Real-Time Systems Compiler* (RTSC) zum Mehrkernsysteme unterstützenden *Real-Time Systems Compiler for Multicore* (RTSC:MP) einzugehen. Davon ausgehend werden die wichtigsten Herausforderungen für die zukünftige Weiterentwicklung zum *Real-Time Systems Compiler for Distributed Systems* (RTSC:DS) wie die erweiterte *Modellierung der Zielplattform*, die *Aufteilung von Anwendungen* und die *Extraktion von anwendungsübergreifenden Abhängigkeiten* erläutert.

Diese Weiterentwicklungen sind notwendig da die Komplexität von Implementierung, Verteilung und Verifikation von Echtzeitsystemen durch neue Trends in der Industrie und der Verfügbarkeit von neuer Hardware stetig steigt. Der *Real-Time Systems Compiler* (RTSC) soll dem Entwickler helfen sich auf seine eigentliche Aufgabe, die Behandlung von Ereignissen, zu konzentrieren ohne nichtfunktionale Eigenschaften wie Termine oder die Verteilung der einzelnen

Ereignisbehandlungen auf ein heterogenes verteiltes Echtzeitsystem aus den Augen zu verlieren.

Literaturverzeichnis

1. T. F. Abdelzaher and K. G. Shin, "Combined task and message scheduling in distributed real-time systems," *IEEE TPDS*, vol. 10, no. 11, pp. 1179–1191, 1999.
2. A. Bastoni et al., "An empirical comparison of global, partitioned, and clustered multiprocessor EDF schedulers," in *31st IEEE Int. Symp. on Real-Time Sys. (RTSS '10)*. Washington, DC, USA: IEEE, Dec. 2010, pp. 14–24.
3. R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Comp. Surv.*, vol. 43, no. 4, Oct. 2011.
4. E. Massa, G. Lima, P. Regnier, G. Levin, and S. Brandt, "Optimal and adaptive multiprocessor real-time scheduling: The quasi-partitioning approach," in *26th Eurom. Conf. on Real-Time Sys. (ECRTS '14)*. IEEE, 2014, pp. 291–300.
5. D.-T. Peng et al., "Assignment and scheduling communicating periodic tasks in distributed real-time systems," *IEEE TOSE*, vol. 23, no. 12, pp. 745–758, 1997.
6. P. B. Sousa et al., "Unified overhead-aware schedulability analysis for slot-based task-splitting," *Real-Time Systems Journal*, vol. 50, no. 5–6,Nov. 2014.
7. E. Deroche et al., "Performance evaluation of a distributed ima architecture," in *WiP ECRTS '15*, H. Ramaprasad, Ed., July 2015, pp. 17–20. [Online]. Available: `http://control.lth.se/ecrts2015/wip.html`