FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Lehrstuhl für Informatik 4 · Verteilte Systeme und Betriebssysteme

Florian Güthlein

Entwicklung eines Konzepts zur latenzgewahren dynamischen Umschaltung zwischen Echtzeitparadigmen

Bachelorarbeit im Fach Informatik

29. November 2017

Please cite as:
Florian Güthlein, “Entwicklung eines Konzepts zur latenzgewahren dynamischen Umschaltung zwischen Echtzeitparadigmen” Bachelor’s Thesis, University of Erlangen, Dept. of Computer Science, November 2017.

Friedrich-Alexander-Universität Erlangen-Nürnberg
Department Informatik
Verteilte Systeme und Betriebssysteme
Martensstr. 1 · 91058 Erlangen · Germany

Entwicklung eines Konzepts zur latenzgewahren dynamischen Umschaltung zwischen Echtzeitparadigmen

Bachelorarbeit im Fach Informatik

vorgelegt von

Florian Güthlein

geb. am 24. August 1995
in Erlangen

angefertigt am

Lehrstuhl für Informatik 4
Verteilte Systeme und Betriebssysteme

Department Informatik
Friedrich-Alexander-Universität Erlangen-Nürnberg

Betreuer:	**Dipl.-Inf. Peter Ulbrich** **Tobias Klaus, M.Sc.** **Florian Franzmann, M.Sc.**
Betreuender Hochschullehrer:	**Prof. Dr.-Ing. habil. Wolfgang Schröder-Preikschat**
Beginn der Arbeit:	**1. Juli 2017**
Abgabe der Arbeit:	**1. Dezember 2017**

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Declaration

I declare that the work is entirely my own and was produced with no assistance from third parties. I certify that the work has not been submitted in the same or any similar form for assessment to any other examining body and all references, direct and indirect, are indicated as such and have been cited accordingly.

(Florian Güthlein)
Erlangen, 29. November 2017

ABSTRACT

In real-time systems the choice between event-triggered and time-triggered systems has to be made. The advantages of either are the cause of their respective disadvantages: Implicitly verified deadline guarantees lead to strict schedules with few possibilities for adjustments whereas the verifiability is immensely impeded by dynamic adaptations according to the current system status.

This thesis investigates the possibilities of utilising both systems at the same time. When a change in tasks for a secure environment is required or deadline misses are impending due to the higher load during the event-triggered system, a switch to a time-triggered system will be signalled to accomplish the necessary adjustments. During a transitional phase the tasks will be dispatched to fit in the precomputed schedule of the time-triggered execution until they can be executed solely in a time-driven manner.

The difficulties that need to be tackled are due to the possible dependency changes and synchronisation primitives. The time-driven system may require fewer or more tasks, every one of which could be in a critical section at the time when the modus switch is signalled. It will be be evaluated how tasks may or may not depend on each other and which dependencies can be introduced by additional tasks during the switch to the time-driven system. Several restrictions will be placed upon the usage of synchronisation to enable the realisation. The synchronisation primitives will serve as means to notify the scheduler of the current progress of every task using the abstractions that the RTSC provides.

The proposed algorithm uses a special selection of the previously running tasks and new tasks to continue an adapted event-driven scheduling with time triggered scheduling decisions, until the switch can be completed successfully, at the point when the progress of every task is in line with the time table.

KURZFASSUNG

In Echtzeitsystemen muss die Wahl zwischen ereignis- und zeitgesteuerten Systemen getroffen werden. Die Vorteile der beiden sind gleichzeitig die Ursache ihrer jeweiligen Nachteile: Die implizit garantierten Termineinhaltungen führen zu strikten Ablaufplänen mit wenig Möglichkeiten Anpassungen vorzunehmen, wohingegen die Verifizierbarkeit durch die dynamischen Reaktionen unter Berücksichtigung des Systemzustands enorm behindert werden.

Dieser Arbeit untersucht die Möglichkeiten der zeitgleichen Nutzung beider Systeme. Wenn Änderungen der auszuführenden Ausgaben für die Sicherheit des Systems erforderlich werden oder Terminverletzungen durch einer zu hohen Last bevorstehen, wird ein Wechsel zu einem zeitgesteuerten System signalisiert, um die notwendigen Anpassungen durchzuführen. In einer Transitionsphase sollen die Aufgaben so ausgeführt werden, dass sie in einer vorher berechneten Zeittabelle eingegliedert werden, bis ein vollständiger Übergang in den zeitgesteuerten Modus möglich wird.

Die Schwierigkeiten der Durchführungen liegen bei den möglichen Abhängigkeitsänderungen und den Synchronisationsprimitiven. Das zeitgesteuerte System kann weniger oder mehr Aufgaben benötigen, wobei sich jede einzelne zur Zeit des Moduswechsels in einem kritischen Abschnitt befinden kann. Es wird untersucht wie Aufgaben voneinander abhängen können und welche Abhängigkeiten bei einem Wechsel zum zeitgesteuerten System durch zusätzliche Aufgaben eingeführt werden können. Daraufhin werden einige Einschränkungen bezüglich der möglichen Synchronisation getroffen, um die Umsetzung zu ermöglichen. Die Synchronisationsprimitiven dienen weiterhin dazu den Ablaufplaner über den Fortschritt der aller Aufgaben zu informieren, indem die Abstraktionen des RTSC genutzt werden.

Der vorgeschlagene Algorithmus verwendet eine spezielle Auswahl der vorher laufenden und der neuen Aufgaben, um eine angepasste Ereignissteuerung fortzusetzen, bei der die Entscheidungen zeitgesteuert eingeleitet werden, bis der Wechsel vollständig durchgeführt werden kann, sobald der Fortschritt jeder Aufgabe gleichauf mit dem Zeitplan ist.

INHALTSVERZEICHNIS

1 EINLEITUNG

Computergesteuerte Geräte umgeben uns im täglichen Leben – sei es in der Freizeit oder im Beruf. Der Kühlschrank, das Auto oder Produktionsmaschinen, fast überall sind Mikrocontroller für die Funktionalität unserer Technik verantwortlich, welche immer mehr Aufgaben erfüllen soll. Ursprünglich waren der Zweck und die Aufgaben dieser Systeme eindeutig und singulär festgelegt als die gesicherte Umsetzung der jeweiligen Hauptaufgabe – Automobile müssen Menschen transportieren und der Kühlschrank seine Temperatur halten. Um einen fehlerfreien Ablauf zu ermöglichen, wurden Echtzeitsysteme entworfen, deren Fokus auf der Bearbeitung aller notwendigen Aufgaben innerhalb einer fest vorgegeben Zeit liegt. Um die Güte zu gewährleisten, wurde das gesamte System zur Einhaltung der zeitlichen Forderungen entworfen: Die gewählte Hardware muss die spezielle Software unterstützen, welche für die Umsetzung der physikalisch notwendigen Aufgaben zuständig ist.

Doch mit dem technischen Fortschritt überlagern sich die Aufgabenbereiche vieler dieser Systeme, was den Entwurf eines sicheren Systems erschwert. Beispielsweise sind Automobile nicht mehr ausschließlich für den Transport zuständig, sondern enthalten auch eine Multimedia-Plattform zur Unterhaltung und der Kühlschrank soll nun auch einen Bildschirm und Internetanbindung bieten. Es handelt sich eigentlich um getrennte Aufgabenbereiche, welche nun durch ein gemeinsames System umgesetzt werden sollen, weshalb oftmals die ursprünglichen Anforderungen eines geregelten, sicheren Ablaufs in Konflikt mit den neu hinzukommenden Erwartungen treten. Zufriedenstellende Lösungsansätze teilen die Aufgaben oftmals in unterschiedlichen Kategorien mit entsprechender Priorität ein, sodass weiterhin zuverlässiges und gefahrloses Nutzen gewährleistet ist, aber die zusätzliche Funktionalität ebenfalls ermöglicht wird. [Bro06]

Eine strike, statische Einteilung in verschiedene Klassen ist die sicherste Umsetzung und liefert die in personennahen Umgebungen wie dem Personennahverkehr, bei der lebensbedrohliche Situationen durch technisches Versagen entstehen können, wichtige Korrektheitgarantie des Systems. Wie in diesem Beispiel zu erkennen ist, handelt es sich in einigen Bereichen um den mobilen Betrieb, wodurch die verfügbaren Ressourcen begrenzt sind. [Bro05] Leider ist dieser Ansatz der strikten Einteilung bezüglich der Nutzung der verfügbaren Ressourcen nicht optimal, da die Aufgaben des Systems durch die realen Umstände diktiert werden und so in Normalbetrieb oder Ausnahmesituation unterschiedlich reagiert werden muss. Durch ein genaueres Wissen über diese Dynamik können Ressourcen derart an die Aufgaben verteilt werden, dass die Sicherheit des Systems nicht gefährdet wird, aber so viele Aufgaben wie möglich erfülllt werden.

Einen besonderen Einfluss auf die zeitlichen Eigenschaften hat der Ablaufplaner (engl. scheduler), welcher die Reihenfolge der auszuführenden Aufgaben bestimmt. Bei Echtzeitsystemen unterscheidet man zunächst zwischen zwei Paradigmen: In sicherheitskritischen Anwendungen wird oft ein *zeitgesteuerter* (engl. time-driven) Ansatz herangezogen, bei welchem zu festen, vor der

Ausführung berechneten Zeitpunkten bestimmte Aufgaben ausgeführt werden. Der enorme Vorteil dieses Verfahren ist es, dass ein Einhalten sämtlicher zeitlicher Bedingungen vor der tatsächlichen Nutzung garantiert werden kann, womit ein Versagen des Systems vermieden wird. Für die statische Planung müssen sämtliche Situationen zu jeder Zeit behandelt werden können, womit keine dynamische Verteilung zusätzlicher Ressourcen an Sekundärfunktionen möglich ist, wenn diese nicht zur Gewährleistung der Sicherheit benötigt werden. Dafür eignet sich ein *ereignisgesteuerter* (engl. event-driven) Ansatz besser, der deshalb weniger in kritischen Anwendungen verwendet wird. Bei diesem leiten Ereignisse[1] den Ablaufplaner ein, sodass die Entscheidungen dynamisch getroffen werden.

Die Flexibilität ist einer der Hauptgründe, weshalb die Ereignissteuerung einer Zeitsteuerung wenn möglich vorgezogen wird – es muss nur die notwendige Arbeit geleistet werden und die Verteilung der Ressourcen kann zur Laufzeit je nach Situation angepasst werden. Der große Nachteil ist eine sehr schwierige Verifikation des Systems, was den Einsatz in sicherheitstechnisch gefährlichen Situationen schwierig macht. [Kop91]

Um die Nachteile der jeweiligen Verfahren auszugleichen, wurden Systeme entwickelt, die nach Notwendigkeit durch einen Moduswechsel in verschiedenen Situationen unterschiedliche Aufgaben erfüllen. Viele bisherige Ansätze verwenden entweder feste Zeitpunkte für einen solchen Wechsel, fordern die Unabhängigkeit der Aufgaben, oder vernachlässigen ihre Kontinuität bei einem Moduswechsel, indem die Aufgaben ersetzt anstatt fortgesetzt werden. Unabhängig von der Behandlung der Aufgaben bei einem Wechsel wird die Art der Einplanung bisher stets beibehalten. Diese Arbeit untersucht die Möglichkeiten, wie ein Moduswechsel von einer ereignisgesteuerten zu einer zeitgesteuerten Ausführung im Ein-Kern-Betrieb erfolgen kann, sodass die Aufgaben reibungslos fortgesetzt werden können, um auf kritischen Situationsveränderungen schnell zu reagieren.

Dabei spielen die vom Real-Time Systems Compiler (RTSC) genutzten Abstraktionen eine tragende Rolle, da dieses aus Systemabstraktionen Software für beide Arten der Einplanung generieren kann und Informationen über die Struktur der Anwendungen liefert. Dazu werden die von den Systemdefinitionen verwendeten Primitiven betrachtet, welche Abhängigkeiten zwischen Aufgaben erzeugen, die bei einem Moduswechsel nicht verletzt werden dürfen und eine eigene Behandlung erfordern, um eine fehlerfreie Fortsetzung zu gewährleisten.

Der vorgeschlagene Ansatz zeichnet sich durch einen Transitionsmodus aus, welcher einen Mischbetrieb von Ereignis- und Zeitsteuerung entspricht, bevor ein vollständiger Übergang zu einer sicheren Ausführung durchgeführt wird.

Zunächst folgt die Präsentation der von Echtzeitsystemen gebotenen Funktionalitäten in Abschnitt 2.1 und die Grundlagen des RTSC in Abschnitt 2.2. In Kapitel 3 folgt als erstes eine Darstellung der Funktionsweise des Moduswechsels mit einer anschließenden Untersuchung der Veränderungen der zu erledigenden Aufgaben, sowie die Umsetzung dieser. Anschließend wird ein konkreter Algorithmus für den Moduswechsel in Kapitel 4 beschrieben und seine Anwendbarkeit diskutiert. Abschließend werden verwandte Arbeiten in Kapitel 5 besprochen und im Kapitel 6 findet sich ein Fazit.

[1] Dies können Hard- oder Software-Signale sein. Beispielsweise eine Veränderung eines Sensors, das Beenden einer anderen Aufgabe oder weitere Nutzereingaben, wie ein Bremsvorgang

2 GRUNDLAGEN

Im folgenden wird die Notwendigkeit der Erledigung von Aufgaben in Echtzeitsystemen innerhalb einer vorgegebenen Zeit, die von der Physik gefordert und von der Regelungstechnik ermittelt wird, besprochen. Dazu wird Der zur Umsetzung benötigte Zusammenhang der Ergebnisse der Regelungstechnik mit der Software erläutert sowie die verschiedenen Arten der Ablaufplanung genauer dargestellt. Anschließend werden die verwendeten Abstraktionen des RTSC und die Verwendung dieser beschrieben, sowie die geforderten Funktionen zur Erstellung von Aufgaben und deren Kommunikation aufgestellt.

2.1 Echtzeitsysteme

Echtzeitsysteme, engl. Real-Time Operating Systems (RTOSs), werden meistens zur Steuerung von physischen Komponenten wie beispielsweise Mikrocontrollern in Autos oder Flugzeugen, Produktionsrobotern in der Industrie und vielem mehr verwendet. Hier entsteht die Verbindung aus *physischen* und *virtuellen* Aufgaben – die Software bekommt als Eingabe den Zustand der Hardware und deren Umgebung, führt anhand dieser Berechnungen durch, und sorgt für die Steuerung von Gerätekomponenten. Aus der Verbindung der beider wird fortgeschrittene Technik entwickelt.

2.1.1 Zusammenhang von Echtzeitsystemen und Regelungstechnik

Die Regelungstechnik befasst sich mit der Steuerung eines technischen Systems: Es werden Sensorwerte in festen zeitlichen Abständen gelesen, woraufhin ein Steuerungssystem bestimmt wie sich das System verhalten muss und gibt dies an Aktoren weiter, welche Motoren oder andere Hardware steuern [Kop06]. Aus den physikalischen Gesetzmäßigkeiten ergeben sich für die Größen des Systems beispielsweise bestimmte Werte, Verläufe oder Abhängigkeiten, die trotz externer Einflüsse oder Störungen eingehalten werden müssen. Zur Umsetzung dieser Anforderungen werde *Aktionen* bestimmt, welche für die Einhaltung aller Vorgaben sorgen. Zum Beispiel gibt es Regelungsschritte, welche für die Übereinstimmung von Soll- und Ist-Wert verantwortlich sind, oder Sensormessungen, welche den aktuellen Ist-Wert ermittelt, sodass errechnet werden kann, was die Regelungsschritte leisten müssen.

Diese sind alle an *Ereignisse* gebunden, welche die Notwendigkeit der Ausführung eines Schrittes signalisieren. Beispielsweise leitet eine Abweichung der tatsächlichen Geschwindigkeit eines Autos von der am Tempomat eingestellten eine Anpassung durch die Software ein, damit die Geschwindigkeit konstant gehalten wird. Ein Beispiel für die Notwendigkeit der Reaktion auf ein Ereignis innerhalb einer fest vorgegeben Zeit ist wiederum das Aktivieren der Bremse. Diese wird bezüglich des *Auslösezeitpunkts* als *Termin* des Ereignisses bezeichnet. Somit werden alle Aktionen des Sys-

tems als Reaktion auf Ereignisse, also Umweltveränderungen, durchgeführt. Dabei werden zeitliche Eigenschaften berechnet, welche für die Stabilität und korrekte Funktionsweise eingehalten werden müssen: Wie häufig eine Messung durchgeführt werden muss, damit die Daten aktuell genug sind, wie oft die Größen angepasst werden müssen, damit das System stabil bleibt, oder ob diese Schritte regelmäßig oder sporadisch durchgeführt werden sollen, wird von der Regelungstechnik ermittelt [LH09]. Diese Planung wird für sämtliche Teile des Gesamtsystems vollzogen, welche oft aus einer Vielzahl von Einzelkomponenten besteht. Die Regelungstechnik liefert als Ergebnis alle Ereignisse, deren zeitliche Eigenschaften, und deren Zusammenhang, auf welche das System reagieren muss und welche durch die Software bearbeitet werden müssen.

Nach der Bestimmung der Anforderungen an das System durch die Regelungstechnik werden diese an die Informatik weitergegeben und die Parameter auf die korrespondierenden Softwareelemente übertragen. Es werden also die *Ereignisse* übernommen, wobei zwischen *periodischen* und *nicht-periodischen* Ereignissen unterschieden wird.

Die periodischen Ereignisse werden durch ihre Periode P, zu der sie zyklisch ausgeführt werden, ihre Phase ϕ, welche eine Verschiebung der Periode bezüglich eines gemeinsamen Bezugspunktes aller Aufgaben darstellt, und den Jitter, ein Interval um der Periode, in welchem das Signal ankommen kann, charakterisiert.

Nicht-periodische Ereignisse besitzen nur eine minimale Zwischenankunftszeit (engl. minimum interarrival time). Dies ist die minimale Zeit, welche vergangen sein muss, bevor dieselbe Aufgabe erneut ausgelöst werden kann.

Ein Ereignis erfordert eine Reaktion durch eine *Aufgabe* (engl. task), welche in *Arbeitsaufträgen* (engl. jobs) die nötigen Berechnungen durchführt und die notwendigen Ergebnisse bereitstellt. Da die Berechnungen Einfluss auf die reale Welt haben, ist dabei nicht nur die Korrektheit relevant, sondern auch der Zeitpunkt zu dem diese zur Verfügung stehen. Alle Arbeitsaufträge müssen zu ihrem *Termin* erledigt sein. Allerdings kann dieser nicht ohne Anpassung von der Regelungstechnik übernommen werden, da Latenzen zwischen Auftreten des realen Ereignis und der Ankunft bei der Software sowie bei Übernahme von neuen Stellwerten durch Hardwarekomponenten entstehen. Diese Termine sind in der Regel relativ zum *Auslösezeitpunkt* des Software-Ereignisses. [Sch11]

Eine Aufgabe ist abgeschlossen, wenn ihre Arbeitsaufträge erfüllt worden sind. Falls alle Aufgaben immer vor dem Eintreten ihres Termins abgeschlossen werden können, so nennt man das System *planbar*.

Man unterscheidet abhängig von den Folgen einer Terminverletzung zwischen weichen, festen, und harten Echtzeitsystemen. Bei weichen Systemen sind gelegentliche Überschreitungen der Fristen akzeptabel, wenn gleich nicht erwünscht, da das System dennoch weiterarbeiten kann. Feste Termine tolerieren ebenfalls Verzögerungen, allerdings können die Ergebnisse der Berechnungen nicht weiterverwendet werden und verfallen. Im Gegensatz zu den ersten beiden Fällen hat das Verletzen von zeitlichen Anforderungen in harten Echtzeitsysteme katastrophale Folgen wie der vollständige Ausfall des Systems, weshalb diese unter allen Umständen eingehalten werden müssen [Sch06].

Ein Systems kann zwar mehrere Arten der Termine enthalten, wobei die Typen von der physischen Umwelt bestimmt werden. Diese Arbeit beschränkt sich auf harte Termine, da diese für ein sicheres System relevant sind. Weiterhin beschränkt sich der RTSC, welcher als Grundlage der Arbeit dient, ebenfalls auf diese. [Sch11]

2.1.2 Arten der Ablaufplanung

Diese Aufgaben müssen nun anhand der ihnen auferlegten, zeitlichen Parameter durch das Betriebssystem zur Ausführung gebracht werden. Hierzu wird der Ablaufplaner aufgerufen, der entscheidet,

welche Aufgabe zu welchem Zeitpunkt ausgeführt wird. Wie bereits beschrieben, wird zwischen den Paradigmen der ereignisgesteuerten und der zeitgesteuerten Ablaufplanung unterschieden.

2.1.2.1 Ereignissteuerung

Bei ereignisgesteuerten Systemen wird die Entscheidung, welcher Aufgabe durchgeführt wird, mit dem Eintreffen von Ereignissen verknüpft. Diese Herangehensweise spiegelt die Reaktionen der regelungstechnischen Aktionen auf physikalische Umstandsveränderungen wieder und damit die Realität innerhalb des Softwaresystems, indem die Bereitstellung der Ergebnisse von Aufgaben zeitlich mit dem Eintreten des auslösenden Ereignisses zusammenhängt. Um die Arbeitsaufträge der Aufgabe zu erledigen, müssen diese nun durch den Ablaufplaner zur Ausführung gebracht werden. Zu diesem Zweck wird der aktuell laufende *Aktivitätsträger*[2] unterbrochen (engl. preemption) und der Ablaufplaner kann auf das Ereignis hin reagieren. Dafür wird der erste Arbeitsauftrag der neuen Aufgabe als lauffähig markiert und steht dem Ablaufplaner in der *Bereitliste* zur Verfügung. Die Wahl des aktuell auszuführenden Auftrags hängt dann vollkommen von dem verwendeten Algorithmus und den Parametern der Aufgaben ab. Unabhängig des Ergebnisses der Auswahl liegt der Auslösezeitpunkt der Aufgabe vor und der absolute Termin steht fest.

Die Auslösung eines Ereignisses selbst kann auf verschiedene Weisen geschehen. Eine Möglichkeit sind Hardware-Signale, welche von Komponenten des Gesamtsystems versendet werden, um eine Bearbeitung eines Systemzustands einzuleiten. Diese starten üblicherweise nicht-periodische Aufgaben, wobei diese als periodische Aufgaben nicht ausgeschlossen sind. Durch Software-Signale startet eine von einer Aufgabenbehandlung eingeleitete Aufgabe, beispielsweise um auf verschiedene Situationen unterschiedlich zu reagieren. Andererseits können auch die zeitlichen Eigenschaften der korrespondierenden, regelungstechnischen Aktionen an das mit der Aufgabe verbundenen Ereignis übertragen werden, sodass dies durch einen festen Zeitgeber ausgelöst wird. Dieser Fall tritt bei streng periodischen Aufgaben ein, wie es bei Sensormessungen häufig notwendig ist.

Durch die Natur der Ereignisse, dem Eintreten zur Laufzeit, können alle Ablaufplaner, die diesen Typ implementieren, sämtliche finalen Entscheidungen erst *online* durchführen. Dies sorgt für eine inhärent schwierig vorauszusehende Ausführreihenfolge, sodass Aussagen über die Zuverlässigkeit des Systems nur mit vielen Testfällen möglich sind [Bro06]. Durch die reaktive Ausführung ist vorher nicht klar, welche Arbeitsaufträge zu welcher Zeit ausgeführt werden, sodass ein Wettstreit um gemeinsam genutzte Ressourcen möglicherweise gar nicht eintritt oder aber beliebig viele Konkurrenten haben kann, was die Ausführung der Arbeitsaufträge potentiell verzögert. Zusätzlich können Ereignisse die Präemption von Aufgaben auslösen, womit der Grad der Unsicherheit steigt, was den Jitter noch weiter verstärken kann. Dadurch ensteht eine unterschiedlich schnelle Ausführung desselben Arbeitsaufträge in verschiedenen Iterationen durch verschiedene Hardwareeffekte wie Caches und Pipelines. Durch diese Faktoren wird die Planbarkeitsanalyse erschwert, was zu einer sehr aufwändigen und pessimistischen Einschätzung führt.

Es lässt sich auch keine verlässliche Aussage über den Zustand[3] des System zu einem bestimmten Zeitpunkt der Ausführung treffen, bevor dieser erreicht wird. Nach ausgiebigem Testen und nur mit einer gewissen Fehlerwahrscheinlichkeit kann davon ausgegangen werden, dass das System seine Fristen einhalten wird. Die formale Verifikation der Planbarkeit ist hingegen, wie oben beschrieben, durch viele auftretenden Faktoren schwierig und oftmals gar unmöglich [Bro06].

In dieser Arbeit wird von einem planbaren System, also einer fehlerfreien Funktionsweise in der Ereignissteuerung, ausgegangen. Diese Annahme ist notwendig und gerechtfertigt, da das Ziel ein Wechsel in einen zeitgesteuerten Modus ist, bevor eine Terminverletzung oder ein Fehlschlagen des

[2] Dies kann ein vollständiger Prozess oder nur ein Faden (engl. Thread) sein.

[3] Dieser beinhaltet den Fortschritt aller Aufgaben und deren Status, beispielsweise der Speicherinhalt und Instruktionszähler

Systems durch eine weniger strikte Ausführung mit weniger strengen Parametern eintreten kann. Es wird davon ausgegangen, dass ein bevorstehendes Fehlschlagen rechtzeitig vorausgesagt wurde und ein Wechsel in den sicheren Modus stattfand.

2.1.2.2 Zeitsteuerung

Bei der Zeitsteuerung entsteht kein reaktives System, sondern ein statisch geplantes. Der Start einer Aufgabe findet nicht durch ein vom Betriebssystem unabhängiges Signal einer Hardwareschnittstelle oder einer anderen Aufgabe statt, sondern mithilfe eines Signals durch einen Zeitgeber, bei dessen Ankunft direkt feststeht, welche Aufgabe ausgeführt werden muss. Zur Laufzeit sind keine Auswahlmöglichkeiten mehr vorhanden.

Dies wird erreicht, indem vor dem Systemstart eine spezifische Reihenfolge berechnet wird, sodass jeder Zeitpunkt der Ausführung den Arbeitsauftrag eindeutig bestimmt. Dies nennt man den *Zeitplan* (engl. scheduling table) des Systems, welcher aus einer Menge bzw. Liste von nicht überlappenden, aufeinander folgenden Intervallen besteht. Diese Intervalle werden auch als *Zeitfenster* (engl. time slot) bezeichnet und stellen eine Surjektion zu den Aufgaben dar.[4] Da der Zeitplan zu jedem Zeitpunkt der Ausführung bekannt sein muss und nur eine endliche Länge haben kann, muss dieser streng zyklisch aufgebaut sein. Nachdem der Plan einmal vollständig abgearbeitet worden ist, wird er erneut vom Beginn an ausgeführt. Effektiv wird die Zeit auf den Startpunkt zurückgesetzt.

Bei der Erstellung müssen zeitlichen Bedingungen der Aufgaben erfüllt werden, denn diese sind weiterhin vom Regelungssystem vorgegeben. Um diese in einem einzigen Zeitplan zu platzieren, muss es einen gemeinsamen Start und Endzeitpunkt geben, was allerdings durch die im Allgemeinen unterschiedlichen Perioden der Aktionen erst zu der *Hyperperiode* eintritt. Diese bezeichnet das kleinste gemeinsame Vielfache aller vorkommenden Perioden aus allen Aktionen. Eine weitere Eigenschaft von sämtlichen Arbeitsaufträgen ist die maximale Ausführzeit (engl. Worst-Case Execution Time (WCET)), die ein solcher benötigt. Diese kann auch in ereignisgesteuerten Systemen analysiert werden und wird dort hauptsächlich zur Verifikation verwendet, doch in zeitgesteuerten Systemen ist sie unverzichtbar. Durch das Wissen, wie lange eine Aufgabe maximal benötigt, muss für jede nur genau so viel Zeit wie notwendig eingeplant und das Zeitfenster kann so klein wie möglich gehalten werden. Dadurch steigt die Anzahl der möglichen Zeitpläne, sodass die Anordnungen der Aufgaben nach unterschiedlichen Kriterien erfolgen kann.

Aufgrund der immer komplexeren Hardware ist die WCET nicht exakt oder nur sehr schwierig berechenbar, weshalb sehr pessimistische Schätzungen[5] verwendet werden müssen, sodass es sich garantiert um die *maximale* Ausführzeit handelt, auch wenn eine schnellerer Ablauf wahrscheinlicher ist [Kop06]. Eine bessere Nutzung der Hardware begründet auch die Verwendung eines ereignisgesteuerten Systemes im Normalbetrieb, bevor in die Zeitsteuerung gewechselt wird.

Bisher noch nicht besprochen wurden die nicht-periodischen Aktivitäten. Auch diese müssen in den Zeitplan eingegliedert werden, weshalb ihnen zwangsweise eine Periode zugewiesen werden muss. Diese Transformation geschieht, indem die Aufgabe so häufig eingeplant wird, dass sie zur Laufzeit überprüfen kann, ob ihre Arbeit verrichtet werden muss und diese nur wenn nötig durchführt. Dabei gibt es zwischen zwei aufeinander folgenden Zeitpunkten, zu denen die Aufgabe eingeplant ist, keinen weiteren Zeitpunkt, zu dem die Arbeit notwendig wäre und im nächsten Zeitfenster nicht rechtzeitig beendet werden würde.

[4] Es existieren auch zeitgesteuerte Systeme, welche mehrere Aufgaben innerhalb eines Zeitfensters zulassen, diese werden hier nicht weiter betrachtet, da es keine funktionale Einschränkung darstellt und die Ablaufplanung damit den Fall nicht zusätzlich betrachten muss. Die Unterstützung dieser Art kann bei in einem Zeitfenster unabhängigen Aufgaben problemlos nachgeliefert werden.

[5] Das bedeutet es werden Optimierungen der Rechnerarchitektur wie Caches ignoriert oder als leer angenommen, wenn der genaue Inhalt nicht bekannt ist.

Im Gegensatz zur Ereignissteuerung ist vorab genau bekannt, wann welche Aufgabe durchgeführt wird, womit gemeinsame Ressourcen ohne explizite Synchronisation verwendet werden dürfen, da die Ausführung eines Arbeitsauftrages niemals unterbrochen wird, sodass der Zugriff implizit durch die zeitliche Synchronisation durch den Ablaufplan geschützt ist.

2.1.2.3 Moduswechsel

Für die Entwicklung von Systemmodi gibt es eine Vielfalt an Gründen. Um die verschiedene Operationsmodi eines Regelungssystem darzustellen, können Aufgaben verändert werden, sie können entfallen oder dazukommen. Durch Umweltanpassungen könnte sich ebenfalls die Frequenz oder die Ausführzeit verändern, sodass die Parameter im System entsprechend angepasst werden müssen. Reale Systeme können ausfallen und die Notfalllösung muss eingeschalten werden [TBW92].

Seien m verschiedene Modi vorhanden, dann gibt es in einem solchen System zu jeder Aufgabe τ_j mehrere Parameter. Sei $c_j^i = 1$, falls τ_j im Modus i ausgeführt werden soll, dann gilt

$$\tau_j = \left\{\left(p_j^1, \phi_j^1, D_j^1, x_j^1\right), \cdots, \left(p_j^m, \phi_j^m, D_j^m, x_j^m\right)\right\}. \tag{2.1}$$

An die Aufgabenmenge jedes Modus wird die Anforderung der Planbarkeit gestellt. Falls ein Modus nicht planbar ist, ist das Gesamtsystem nicht nutzbar [RC01]. Weiterhin können die Aufgaben kategorisiert werden in zu beendende Aufgaben und abzubrechende. Das bedeutet, auch wenn die Aufgabe τ_j im neuen Modus nicht benötigt wird, so muss sie in manchen Fällen dennoch abgeschlossen werden. Dadurch enthält der neue Modus in einer Übergangsphase eine Überzahl an Aufgaben, weshalb jede Übergangsphase bei der Planbarkeitsanalyse miteinbezogen werden muss.

Die Moduswechselverfahren können anhand ihres Übergangs charakterisiert werden. Manche arbeiten auf Basis eines vorher berechneten Ablaufplans beziehungsweise erlauben den Wechsel erst zur Hyperperiode aller Aufgaben oder zu anderen festen Zeitpunkten, sodass keine Interferenzen zwischen den alten und den neuen Aufgaben bestehen. Dies ist ein sehr einfaches Protokoll, was zu einer sehr hohen Latenz zwischen der Signalisierung eines Wechsels und dem Vollzug führt. Andere Ansätze verwenden Transitionsphasen, in denen Aufgaben aus beiden Modi durchgeführt werden. Um die Planbarkeit zu garantieren, sind verschiedene Lösungen vorgestellt worden. Eine davon erlaubt eine verzögerte Durchführung des Wechsels, indem die laufenden Aufgaben beendet werden und die neuen nach einem kurzen Versatz beginnen [RC01]. Beispiele für weitere Protokolle können nachgelesen werden in *Mode change protocols for priority-driven preemptive scheduling* [Sha+89] oder *Mode changes in priority preemptively scheduled systems* [TBW92].

Für gewöhnlich handelt es sich bei dem Übergang um dieselbe Art des Ablaufplaners, sodass bei allen Modi beispielsweise ein Raten monotoner, priotitätsgesteuerter oder zeitgesteuerter Ablaufplaner verwendet wird, aber keine Mischung der Ablaufplaner innerhalb eines Systems vorkommt.

Ein besonderes Augenmerk muss beim Moduswechsel auf die Synchronisationsprimitiven gelegt werden. Denn hierbei muss beachtet werden, dass die Synchronisation nicht durch einen Moduswechsel gestört wird. Viele Arbeiten gehen von unabhängigen Komponenten aus. [BD] Stattdessen müssen abhängige Aufgaben, die abgebrochen werden könnten, zu Ende geführt werden, wenn sie sich in einem kritischen Abschnitt befinden. Eine weitere Möglichkeit wäre ein geregelter Abbruch. Dieser wird im weiteren Verlauf der Arbeit dargestellt.

2.1.3 Anforderungen an Echtzeitbetriebssysteme

Die grundlegenden Anforderungen an Echtzeitbetriebssysteme wurden in *A survey of contemporary real-time operating systems* identifiziert als mehrfädige (engl. multi-threaded) Ausführung, ein präemptiver Ablaufplaner, vorhersagbare Synchronisation, verschiedene Prioritätslevel und definierte

Latenzen. [MB05] Viele der Gesichtspunkte sind für die Planbarkeitsanalyse und das Einhalten der zeitlichen Bedingungen relevant. Die Reaktionsgeschwindigkeit der Software auf externe Signale muss bekannt sein, um eine Garantie für die Sicherheit des Systems zu leisten.

Ein weiterer Aspekt ist die Programmierschnittstelle, das Application Programming Interface (API), anhand derer die Anwendungen, welche letztendlich für die Behandlung der physischen Umstandsänderungen verantwortlich sind, entwickelt werden. Eine umfangreiche Funktionalität erleichtert diesen Prozess. Dabei werden oft folgende Schnittstellen bereitgestellt:

- Uni- und multilaterale Synchronisation
- Warten auf die Erfüllung komplexer Bedingungen
- Möglichkeiten für den Informationsaustausch wie Nachrichten oder gemeinsamer Speicher
- Das Verwenden eines Zeitgebers
- Das Vermeiden von Verdrängungen

In *A survey of contemporary real-time operating systems* sind detailliert Angaben zu einer Vielzahl von RTOS vorhanden. Ein erfolgreicher Ablaufplaner mit Moduswechsel muss zu allen Zeitpunkten der Ausführung all diese Punkte ebenfalls unterstützen und eine niedrige Latenz aufweisen.

2.2 Real-Time Systems Compiler

Das Projekt RTSC befasst sich mit der automatischen Migration von ereignis- zu zeitgesteuerten Applikationen anhand einer systemunabhängigen, gemeinsamen Ausgangssprache, welche den Ablauf verschiedener Komponenten mittels Synchronisationsprimitiven aus ereignis-basierten Systemen beschreibt und dafür sorgt, dass sich der Entwickler ohne Beachtung der Zeitplanung auschließlich mit der Funktionalität beschäftigen kann. Obwohl zunächst nur die Migration zwischen einer den beiden Paradigmen entwickelte wurde, wird der RTSC als Basis dieser Arbeit verwendet, da eine Erweiterung zur Generierung von hybriden Anwendungen realisiert werden kann, indem aus der Eingabe kein reines zeitgesteuertes System übersetzt wird, sondern ein Mischsystem, welches den Moduswechsel gezielt durch eine Informationsweitergabe an den Ablaufplaner unterstützt. Diese Daten müssen im Quellcode der Anwendungen platziert werden und sind dem RTSC bereits durch die vorherige Analyse der Eingabe bekannt, sodass eine spätere Implementierung erleichtert wird.

Das im restlichen Kapitel präsentierte Wissen entstammt der Dissertation von Fabian Scheler [Sch11] sowie seiner Papiere [SSP11; SSP10] und kann in diesen Werken genauer nachgelesen werden.

2.2.1 Systemmodell des RTSC

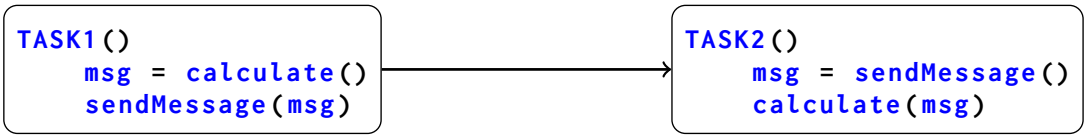
Das RTSC greift auf ereignisgesteuerte Systeme zurück. Der RTSC unterscheidet zwischen den Aufgaben und den Arbeitsaufträgen der Aufgabe, *Unteraufgaben*. Eine Aufgabe besteht aus mindestens einer aber potentiell mehreren Unteraufgaben, wobei in diesem Kontext jede Software Aufgabe eine regelungstechnische Aufgabe repräsentiert und die Unteraufgaben die notwendigen Arbeitsschritte zur Erledigung dieser durchführen. Von Seiten des Betriebssystems sind alle Unteraufgaben eigene Aktivitätsträger mit allen damit verbundenen Eigenschaften. Die Aufgaben werden wiederum durch Ereignisse aktiviert, wobei zwischen *physischen* und *logischen* differenziert wird. Jede Aufgabe besitzt einen *Wurzelarbeitsauftrag*, (engl. rootsubtask), der bei Eintreffen des Ereignisses als lauffähig freigeschaltet wird. Dieser aktiviert in seinem Verlauf weitere Unteraufgaben dieser Aufgabe. Es können

ebenfalls weitere Aufgaben mithilfe eines logischen Ereignisses durch den Wurzelarbeitsauftrag eingeleitet werden. Ereignisse können wiederum periodisch oder nicht-periodisch auftreten und sind, wie bereits besprochen, mit einer Periode, einer Phase, und einem Termin versehen. Unteraufgaben teilen für die Betrachtungen in dieser Arbeit den Termin der Gesamtaufgabe. Dies muss innerhalb des RTSCs nicht zwangsläufig der Fall sein, aber die Annahme hat keine funktionalen Einschränkungen, welche nicht durch eine andere Abbildung aufgelöst werden können. Zwischen Unteraufgaben können mittels Synchronisationsprimitiven Reihenfolgen und geschützte Abschnitte entstehen, welche sich als *Abhängigkeiten* zwischen den Unteraufgaben auswirken.

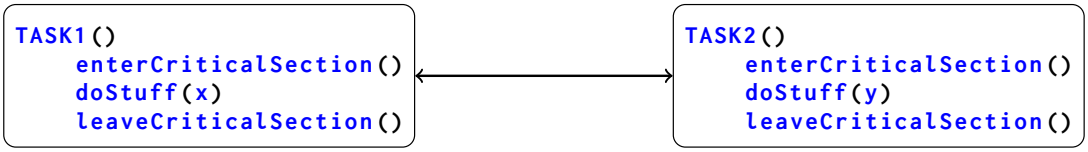
2.2.2 Abhängigkeiten zwischen Aufgaben

Vorzugsweise würde man sich gerne auf einfache Aufgaben (engl. simple tasks) beschränken, bei denen keine Beziehungen mit anderen bestehen, da bei diesen nur die zeitlichen Eigenschaften betrachtet werden müssen, um eine valide Ablaufplanung zu betreiben. In allgemeinen Szenarien ist dies nicht möglich, da zur Erfüllung vieler regelungstechnischer Aufgaben eine Zusammenarbeit von vielen verschiedenen Aktoren und Sensoren erforderlich ist, weshalb auf komplexe Aufgaben (engl. complex tasks) mit beliebigen Interaktionen der Unteraufgaben innerhalb einer als auch zwischen verschiedenen Aufgaben zurückgegriffen wird. Diese werden *Abhängigkeiten* genannt. Bei diesen wird beispielsweise auf die Ergebnisse von Operationen, welche im Normalfall Daten verändern, gewartet. Die Abhängigkeiten beschreiben also eine Vorgänger- beziehungsweise Nachfolger Relation und legt eine Ordnungsreihenfolge zwischen den beteiligten Aufgaben fest. Eine Unterscheidung zwischen *gerichteten* und *ungerichteten* Abhängigkeiten wird durch die verschiedenen Arten der Synchronisation getroffen.

Die gerichtete Variante, also *unilaterale Synchronisation*, tritt auf, wenn eine Unteraufgabe explizit auf das Signal eines anderen wartet, wie es beispielsweise bei blockierenden Nachrichten oder der Aktivierung einer Aufgabe durch logische Events der Fall ist. Wie in Abbildung 2.1a dargestellt wartet beispielsweise $Task2$ auf ein Ergebnis von $Task1$, also ist $Task2$ vom Fortschritt von $Task1$ abhängig. Erst nachdem die Nachricht von $Task1$ gesendet wurde, kann $Task2$ mit dieser weiterarbeiten. Bei dieser Art der Synchronisation erlaubt das RTSC noch eine zeitlich verzögerte Ablieferung des Signals. Diese wird allerdings bei der Erstellung des zeitgesteuerten Systems eliminiert, weshalb zunächst auch keine Verzögerungen in den ereignisgesteuerten Teilen des Mischsystem bei diesen Primitiven unterstützt werden.



(a) Hier hängt Task2 unilateral von Task1 ab.



(b) Die beiden Tasks schließen sich gegenseitig von der zeitgleichen Ausführung von *doStuff* aus.

Abbildung 2.1 – Anwendung von beiden Arten der Synchronisation

Bei *multilateraler Synchronisation*, also ungerichteten Abhängigkeiten, handelt es sich um einen Wettstreit um den Zugriff auf eine limitierte Ressource, welcher nur in *kritischen Abschnitten* verwendet werden kann, um einen konsistenten Zustand dieser zu garantieren. Eine gleichzeitige Veränderung von solchen Ressourcen würde zu unvorhersehbarem Verhalten führen und muss zwangsläufig verhindert werden. Weiterhin wird hier keine statische Reihenfolge festgelegt, sondern der Konflikt erst während der Laufzeit beigelegt. Im Beispiel in Abbildung 2.1b wird eine gemeinsam verwendete Ressource durch einen kritischen Abschnitt geschützt, dabei kann es sich um beliebige Daten handeln. Wenn also eine der beiden Aufgaben diesen Abschnitt betreten hat, muss die jeweils andere Aufgabe auf das Verlassen des kritischen Abschnitts warten, bevor der eigene Abschnitt betreten werden kann.

Die letzte Variante sind Datenabhängigkeiten, welche bei der Verwendung von gemeinsamen Speicher auftreten. Diese Arbeit nimmt an, dass solche Situationen mithilfe von mehrseitiger Synchronisation geschützt sind, damit der Ablaufplaner vom Erreichen sämtlicher Abhängigkeiten informiert wird, was bei der Verwendung von geteiltem Speicher nur durch zusätzliche Informationsweitergabe möglich wäre.

Der RTSC erkennt die Stellen, an denen Abhängigkeiten existieren, um diese Information bei der Transformation in ein zeitgesteuertes System verwenden zu können. Es entfernt alle Abhängigkeiten aus dem Programmcode, da ein zeitgesteuertes System inhärent auf explizite Synchronisation verzichtet und diese stattdessen auf den Zeitplan abbildet, welcher implizit für die Einhaltung der Abhängigkeiten sorgt, indem am Ende eines Zeitfensters statt einer Synchronisationsprimitiven der nächste, richtige Abschnitt aufgerufen wird.

2.2.3 Atomic Basic Blocks

Um die Abhängigkeiten zu verwalten, verwendet der RTSC das Konzept der *atomaren Basisblöcke* (engl. Atomic Basic Blocks (ABBs)). Hierzu wird auf die Basisblöcke des Übersetzerbaus zurückgegriffen. Ein Basisblock besteht aus einander folgenden Instruktionen, wobei dieser im Fall von *maximalen* Basisblöcke so groß wie möglich ist, aber einen rein linearen Ablauf besitzt. Das bedeutet bei Verzweigungen enden die Blöcke. Alle Verzweigungen[6] bilden die Kanten in dem *Kontrollflussgraphen*, dessen Knoten die Basisblöcke sind. Der RTSC analysiert in den Basisblöcken die Abhängigkeiten anhand derer er die Blöcke in ABBs aufteilt. Ein ABB kann auf drei Arten gespalten werden: Er beginnt, falls eine Instruktion auf ein Signal oder einen kritischen Abschnitt wartet, und wird beendet, wenn eine Abhängigkeit durch eine Instruktion erfüllt wird, oder kann beendet beziehungsweise begonnen werden, um die ersten beiden Fälle darzustellen. Letzeres ist notwendig, da die Aufteilung jeweils einen Anfang und ein Ende erzeugt, aber anhand einer Instruktion vorgenommen wird. Zuletzt werden noch weitere künstliche Trennungen anhand des Kontrollflussgraphen vorgenommen. Keine Operation innerhalb eines ABB hat Abhängigkeiten nach außen außer am Anfang oder Ende.

Diese Analyse wird zunächst für jede einzelne Funktion vollzogen und der Graph für diese erstellt. Danach werden die Abhängigkeiten für das gesamte Softwaresystem aufgebaut, bei dem die einzelnen Graphen durch ihre Abhängigkeitskanten verbunden sind. Das Ergebnis ist eine Aufteilung sämtlicher Schritte, die unternommen werden können, wobei jeder Knoten des entstandenen Graphen ausgeführt werden kann, sobald sein Vorgänger und die anderen Abhängigkeiten ausgeführt bzw. erfüllt sind. Im RTSC gibt es noch die Möglichkeit, dass Abhängigkeiten mit booleschen Ausdrücken erweitert werden, sodass beispielsweise nur eines von mehreren Signalen benötigt wird, um einen ABB zu starten, oder eine Konjunktion von vielen Ausdrücken verlangt wird. Die Disjunktionen

[6] Verzweigungen treten auch bei Sprüngen und Schleifen auf und nicht nur bei klassischen Verzweigungen.

können durch das RTSC aufgelöst werden, indem der ABB dupliziert wird und jedes Duplikat nur auf ein bestimmtes Signal wartet. Diese Arbeit geht deshalb davon aus, dass ausschließlich eine bestimmte Menge an Abhängigkeiten existiert, die alle erfüllt sein müssen, und geht nicht weiter auf andere Prädikate ein.

Die vorgestellten atomaren Basisblöcke werden als Verwaltungseinheit für die Ablaufplanung verwendet, da nur an ihren Grenzen Instruktionen mit Auswirkungen auf die Ablaufplanung ausgeführt werden. Damit können Entscheidungen anhand des Fortschritts und des Zusammenhangs mit weiteren Aufgaben über die Kenntnis der Struktur einer Aufgabe getroffen werden.

2.2.4 Transformation in eine Zeittabelle

Mithilfe dieser Aufteilung in atomare Basisblöcke erstellt das RTSC bei der Ablaufplanung die *Zeittabelle* oder den *Zeitplan*, welcher festlegt, welche Aufgaben zu welchem Zeitpunkt ausgeführt werden. Dafür können beliebige mögliche Prioritäten und Reihenfolgen gewählt werden, solange jedes Ereignis vor seinem Termin bearbeitet sein wird. Eine Möglichkeit des RTSC ist ein modifizierter Earliest Deadline First (EDF)[7], bei dem aus der Menge der möglichen lauffähigen ABBs derjenige mit dem frühesten Termin ausgewählt wird, es sei denn, es gibt eine Möglichkeit mit der aufeinander folgende ABBs hintereinander ausgeführt werden. Diese Optimierung sorgt dafür, dass Elemente der Hardware wie die Caches und die Pipeline der CPU so gut wie möglich genutzt werden.

Wie bereits beschrieben, muss der Zeitplan zyklisch sein. Dazu werden die Perioden aller Ereignisse betrachtet und das kleinste gemeinsame Vielfache, die Hyperperiode, berechnet. Alle Aufgaben, die mehrmals in dieser benötigt werden, werden vervielfacht und dann im ABB-Graph eingesetzt. Bei diesem Vorgang muss ein Graph auf eine lineare Zeitachse übertragen werden, wobei es hier verschiedene Kantentypen gibt: Es gibt die Abhängigkeitskanten zwischen verschiedenen Funktionen (engl. inter-function dependencies) und es gibt Abhängigkeitskanten von aufeinander folgenden Teilen einer Funktion, also Kontrollflussabhängigkeiten. Wenn erstere bei dieser Abbildung zunächst vernachlässigt werden würden, bleibt ein Wald von Kontrollflussgraphen [8] übrig, welche auf eine lineare Achse abgebildet werden müssen. Da es bei Verzeigungen nicht notwendig ist, beide Zweige, von denen maximal einer ausgeführt wird, in unterschiedliche Stellen auf der Zeitachse zu platzieren, werden einzelne ABBs anhand ihrer Kontrollflussabhängigkeiten verschmolzen, wobei die ausgehenden Abhängigkeiten auf folgende Knoten im ABB-Graphen und die eingehenden auf vorhergehende Knoten verschoben werden. Es gibt dennoch Situationen in welchen dies so nicht möglich ist, dabei handelt es sich vorerst um eine Aufgabe des RTSCs, weshalb in dieser Arbeit die vollständige Serialisierung des Graphen zur Erstellung der Zeittabelle angenommen wird. Bei dem Vorgang müssen Schleifen beachtet werden. Sollten innerhalb einer Schleife Abhängigkeiten auftreten, das bedeutet, eine Schleife umspannt mehrere ABBs, dann wird weiterhin angenommen, dass die Schleife ausgerollt wird (engl. loop unrolling), und durch die Abhängigkeiten getrennte Blöcke eine eindeutige Identifizierung erhalten, was innerhalb einer Schleife nur mit Einbezug des Schleifenzählers möglich wäre. Mithilfe der Abhängigkeiten zwischen Funktionen können die nun linearen Unteraufgaben auf der Zeitachse angeordnet werden.

Um die Zeitsteuerung zu ermöglichen, werden dann die Ereignissteuerung bezogenen Elemente entfernt und durch ihr äquivalentes Gegenstück ersetzt. Letzteres wird allerdings nur gemacht, wenn laut Zeittabelle nach dem ABB der aktuellen Unteraufgabe der einer anderen laufen wird. Es wird also auf das nächste Zeitfenster gewartet, wenn eine abgearbeitet worden ist.

Für ein Ereignissteuerung müssen stattdessen die entsprechenden Routinen des Betriebssystems für die aus der Beschreibungssprache stammenden äquivalenten Primitiven eingesetzt werden, da die

[7] engl. für frühster Termin als erstes

[8] Jede Unteraufgabe hat dann einen eigenen, nicht mit den anderen verbundenen Graphen oder Baum von ABBs

Abhängigkeiten durch diese bereits enthalten sind und durch den Ablaufplaner implizit eingehalten werden.

2.2.5 Die Programmierschnittstelle des RTSC

Aus einer Betrachtung ausgewählter Eingabesysteme des RTSCs lässt sich entnehmen, welche Operationen mit direktem Einfluss auf die Ablaufplanung vorhanden sind. Zum einen zeigt ein Vergleich mit kommerziellen Echtzeitsystemen eine große Übereinstimmung der API (vgl. [MB05]) und zum anderen bietet dies die Möglichkeit zu untersuchen, welche Funktionalität der Moduswechsel aus Sicht des RTSCs bieten sollte.

Es wurden die Systemunterstützungen *OSEKOS*, *CSystem*, und *POSIXRTSC* des RTSC untersucht. Dabei finden sich zur (Unter-)Aufgabenerzeugung folgende Möglichkeiten:

- Beim *Start durch ein externes Event* wird eine Aufgabe durch ein externes Signal, beispielsweise durch die Zeit bei periodischen Ereignissen oder abweichenden Messwerten, ausgelöst.
- Tritt ein *Start durch eine Unteraufgabe* auf, so aktiviert eine laufende Unteraufgabe eine weitere Unteraufgabe der gleichen Aufgabe[9], um diese parallel zu bearbeiten, oder eine andere Aufgabe[10] durch ein logisches Ereignis.
- Der *verzögerten Start durch eine Unteraufgabe* verläuft wie bei letzterem, allerdings kommt eine statisch bekannter Verzögerung zwischen Signalisierung und Start vor.

Es gibt dabei einige weitere Restriktionen, aber diese sind für diese Arbeit nicht weiter relevant. Es sollte beachtet werden, dass die Erzeugung einer neuen Aufgabe den Startzeitpunkt derer markiert und somit den absoluten Termin festgelegt.

Die Synchronisationmechanismen sind wie folgt:

- Unilateral:
  - Das *Warten auf eine, mehrere oder auf eines von mehreren Signalen* sorgt dafür, dass sich die Unteraufgabe selbst aussetzt bis ihr das Fortsetzen erlaubt wird.
  - Das *Senden eines Signals* ist der Gegensatz zur vorherigen Aktion und wird an alle darauf Wartenden geschickt.
  - Zur Kommunikation gibt es noch das *Senden einer Nachricht*, bei dem Daten übermittelt und in einer Liste gespeichert werden.
  - Durch ein entsprechendes *Empfangen der Nachricht* wird ein Element aus der Liste entfernt, falls möglich, und sonst auf das Eintreffen einer Nachricht gewartet.
- Multilateral:
  - Durch einen Mutex[11] kann eine *Sperroperation* erfolgen. Dabei setzt der aktuelle Aktivitätsträger aus, falls ein anderer bereits den Mutex gesperrt hat.
  - Dieser wieder *freigegeben* werden. Dies erlaubt es anderen den Mutex wieder zu sperren und in dem Fall, dass dies schon versucht wurde und Aktivitätsträger warten, so bekommt einer der Wartenden die Sperre zugewiesen und kann weiterlaufen.

[9] Dies wird als forked subtask im RTSC bezeichnet, dieser behält die zeitlichen Eigenschaften des Erzeugers bei
[10] Und dies ist ein triggered subtask
[11] engl. Abkürzung für mutual exclusion, zu dt. gegenseitiger Ausschluss

Da es eine Dualität zwischen nachrichten-basierter Kommunikation und den anderen Synchronisationsoperationen im Zusammenhang mit gemeinsamen Speicher gibt [LN78], wird auf eine genauere Untersuchung dieser verzichtet und ausschließlich die anderen beiden Primitiven verwendet. Weiterhin wird die üblicherweise verwendete Semantik angenommen, welche besagt, dass derjenige, der eine Sperre belegt, diese auch wieder freigeben muss und der Versuch, dasselbe durch einen anderen Ausführstrang zu tun, zurückgewiesen wird. Diese Bedingung ist notwendig, um Zustandsaussagen anhand der Abhängigkeiten zu treffen und vorherzusagen, wie eine Sperre wieder freigeschaltet werden kann. Weiterhin gibt es die Möglichkeit auf *Bedingungsvariablen* (engl. condition variables) zu warten. Dies ist eine Mischung der beiden Primitiven: Es kann auf ein Signal gewartet werden, wenn man eine Sperre hält, womit diese freigegeben wird, und bei Ankunft des Signals wird der Faden wieder aktiv und bekommt automatisch die Sperre zugewiesen. Diese Art wird nicht weiter betrachtet.

Als letztes Mittel lässt sich noch ein *Alarm* einstellen, welcher wie ein verzögertes Signal agiert. Nach einer vorgegeben Zeit wird ein Signal versendet und eine Aufgabe erfüllt. Dies entspricht den verzögerten Abhängigkeiten, welche im vorherigen Abschnitt besprochen wurden, weshalb darauf in dieser Arbeit nicht näher eingegangen wird.

2.3 Zusammenfassung

Die in diesem Kapitel erläuterten Konzepte der Echtzeitsysteme und der vom dem RTCS gelieferten Abstraktionen bilden die Grundlagw der Arbeit. Die Definition und Differenzierung der beiden Steuerungsarten in Echtzeitsystem, zeit- oder ereignisgesteuert, schaffen das Umfeld, in welchem die zu bewältigenden Herausforderungen der Vereinigung beider Paradigmen in einem hintereinander geschalteten Moduswechsel erarbeitet werden kann. Zur Überführung der Aufgaben und derer unvorhersehbarer Zustände im durch Ereignisse geleiteten Operationsmodus in einen statischen Zeitbetrieb verleihen die atomaren Basisblöcke dem Ablaufplaner eine Struktur, anhand derer er die Auswahl der für den Übergang auszuführenden Aufgaben treffen kann, ohne die Abhängigkeiten zu verletzen. Durch die Identifizierung der erwarteten Operationen eines Echtzeitsystems wird deren Unterstützung bei der Entwicklung des Moduswechsels geplant.

3 ANALYSE DES MODUSWECHSELS

Die grundlegende Annahme hinter dem Moduswechsel ist die Unterteilung der Ausführung in einen normalen und einen kritischen Modus, wobei der erste im Normalfall ohne Gefährdung der Sicherheit des Systems ereignisgesteuert ausgeführt werden kann. Sobald Anzeichen einer Situationsänderung erkennbar sind, welche eine striktere Ausführung erfordert als im normalen Modus, wird dies dem Betriebssystem signalisiert, welches schnellstmöglich den sicheren Ablauf einer Zeitsteuerung erreichen muss. Dieses Signal kommt für das Betriebssystem im Allgemeinen extern und kann zu beliebigen Zeitpunkten eintreten.

Zunächst wird in diesem Kapitel die theoretische Seite betrachtet. Dazu wird eine Übersicht über die Anforderungen an den Wechsels und verschiedene Ansätze der Umsetzung gegeben, woraufhin der Übergang in kleinere Schritte zerlegt und schematisch erklärt wird. Anschließend können durch isolierte Betrachtung der Aufgabenveränderungen die notwendigen Anforderungen und Problemstellen des Wechsels ausgearbeitet werden. Es wird vorgestellt, wie die Strukturinformationen des RTSCs gesammelt und verwertet werden. Zur Vereinigung der grundlegend verschiedenen Paradigmen werden danach die technischen Abbildungmöglichkeiten der Veränderungen von Aufgaben dargelegt, wodurch eine genau Beschreibung eines Lösungsansatzes folgt.

3.1 Möglichkeiten für den Zeitpunkt des Moduswechsels

Der Vollständigkeit halber soll zunächst zwischen den Zeitpunkten unterschieden werden, zu denen ein Wechsel möglich ist. Dabei soll die Latenz zwischen der Signalisierung und des Vollzugs des Wechsels sowie der Zusatzaufwand durch den Ablaufplaner minimal gehalten werden. Es wird zwischen zwei Teilaufgaben des Wechsels unterschieden. Als erstes findet das Aktivieren der notwendigen neuen Aufgaben und als zweites das Umschalten zu einer zeitlich gesteuerten Ausführung dieser statt. Die beiden Zeitpunkte können abhängig von dem bei der Transition eingesetzten Algorithmus übereinstimmen, müssen dies aber nicht. Der Fokus liegt vorerst beim Übernehmen der Aufgabenveränderungen.

Der Wechsel kann mithilfe von zwei bezüglich des Fortschritts der Aufgaben unabhängigen Möglichkeiten umgesetzt werden. Dies sind zum einen der Abbruch des gesamten ereignisgesteuerten Systems und Start des zeitgesteuerten und zum anderen das Beenden einer Hyperperiode der Ereignissteuerung gefolgt vom Start des zweiten Modus. Die beiden Ansätze bieten beide keine zu lösenden Schwierigkeiten, da die Modi jeweils unabhängig voneinander ausgeführt werden. Allerdings gibt es inhärente Probleme, welche den Wechsel in einen kritischen Modus nicht gestatten. Zum einen sind alle Aufgaben beliebig weit fortgeschritten und haben deshalb einen unbekannten und wahrscheinlich inkonsistenten Zustand. Ein jetziges Beenden im ersten Fall führt zu unbekanntem Verhalten und verwirft alle nicht abgeschlossene Aufgaben und damit die bereits vollzogene Arbeit.

Die aktuell laufenden Aufgaben sind notwendig, sonst wären sie nicht durch ein Ereignis ausgelöst worden, weshalb der Verlust der bisherigen Arbeit sämtliche Arbeitsaufträge invalidiert und deren Termine verpasst werden, was in einem harten Echtzeitsystem wie in dieser Arbeit angenommen nicht möglich ist. Zum anderen kann nicht auf die Hyperperiode gewartet werden, da der Moduswechsel auf kurzzeitige Situationsänderungen reagieren soll und die Hyperperiode zeitlich beliebig lange entfernt sein kann, womit kein sicherer Modus verfügbar ist. Die obigen Zeitpunkte stimmen bei den Ansätzen überein.

Dahingegen können die bisherigen Ergebnisse der Aufgaben übernommen werden, wenn der Wechsel die entsprechenden Aufgaben übernimmt. In den beiden Modi können von den Aufgaben unterschiedliche Ergebnisse erwartet werden, weshalb hier zwei Varianten vorkommen. Zum einen kann eine Aufgabe trotz Wechselsignal seine bisherigen Berechnungen vervollständigen und erst zur nächsten Periode die Anpassungen übernehmen. Dies liefert ein definiertes Verhalten, aber die Latenz bis zum vollständigen Wechsel ist abhängig von den vorhanden Aufgaben, deren Parameter und Struktur. Zum anderen kann versucht werden diese Änderungen so gut wie möglich in aktuell laufenden Aufgaben einzubauen. Dies sorgt für eine frühere Übernahme der veränderten, alten Aufgaben, womit der Wechsel zu einer sicheren Zeitsteuerung ebenfalls schneller erfolgt. Beide Verfahren beziehen sich nur auf das Aktivieren von neuen Unteraufgaben, die Teil bisheriger Aufgaben sind und diese verändert. Es können allerdings auch völlig neue hinzukommen, weshalb das Vorgehen bei diesen von der Behandlung von gerichteten Abhängigkeiten bestimmt und genauer untersucht wird

In der vorliegenden Arbeit werden viele Einschränkungen getroffen, da eine Bearbeitung aller Möglichkeiten auf technische Probleme stößt, welche entweder nicht oder nur mit immensen Aufwand gelöst werden können. Viele Änderungen an Aufgaben mit gerichtete Abhängigkeiten können nur synchron oder mit erheblichen Aufwand stattfinden, weshalb auf diese verzichtet wird, um einen ersten Wechsel zwischen den beiden Paradigmen zu ermöglichen.

3.2 Der Moduswechsel im Modell

Die entscheidende Annahme in allen Bereichen ist die Möglichkeit die Bestandteile einer im ereignisgesteuerten Modus ausgeführten Unteraufgabe einem im Zeitplan vorkommenden Intervall zuzuordnen. Dies stützt sich hauptsächlich auf unveränderte Parameter, siehe Abschnitt 3.3.3, sodass alle Teile einer Aufgabe, die im ereignisgesteuerten Modus beim Beginn ihrer Periode zum Zeitpunkt t gestartet wurde, im zeitgesteuerten Bereich innerhalb des Intervalls beginnend bei t und endend bei $t + D$ vorhanden sind, wobei D der relative Termin der Aufgabe ist. In Abbildung 3.1 sieht man anhand von fünf verschieden farbigen Tasks, in welchen Intervallen diese ausgeführt werden können. $Task_2$ wird durch die Aktivierung von $Task_4$ unterbrochen und mitten in der Ausführung von Teilaufgabe $Task_2^2$ unterbrochen. Bei $Task_4$ passiert Selbiges mit $Task_5$, sodass zum Zeitpunkt des Wechsels gerade $Task_5$ abgeschlossen wurde. Weiterhin wurde $Task_1$ nicht ausgeführt, weil dieser erst im zeitgesteuerten Modus eingegliedert wird. $Task_3$ konnte wegen mangelnder Priorität $Task_5$ nicht vor dem Wechsel verdrängen.

Die Reihenfolge und der Fortschritt aller Tasks variieren hierbei von jeder ereignisgesteuerten Ausführung zur nächsten. Die Graphik illustriert, wie jeder Teil einer Aufgabe innerhalb des Intervalle dieser ausgeführt wird. Dies ist in jedem planbaren System der Fall, da die Ausführung einer Aufgabe außerhalb das Verpassen ihres Termins bedeuten würde, was als Annahme ausgeschlossen wurde. Weiterhin werden die entsprechenden Zeitfenster, in denen eine Aufgabe bearbeitet werden soll, innerhalb des Intervalls als vollendet markiert, sobald eine Aufgabe beendet wird. So werden die richtigen Zeitfenster für beispielsweise $Task_2^1$ und $Task_2^2$ gefunden und deren Status eingetragen.

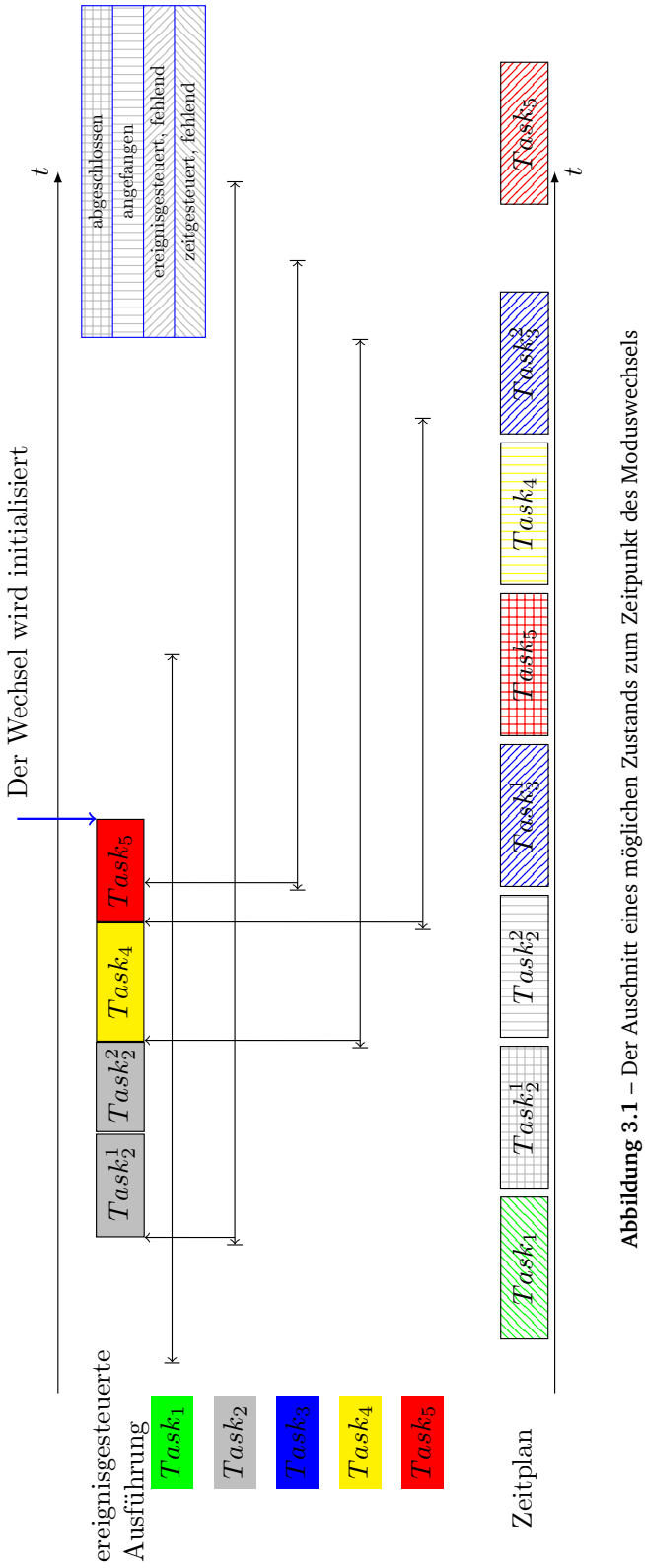


Abbildung 3.1 – Der Auschnitt eines möglichen Zustands zum Zeitpunkt des Moduswechsels

$Task_5$ zeigt wie bei Aufgaben vorgegangen wird, die periodisch ausgeführt werden beziehungsweise mehrmals im Zeitplan vorkommen: Die zugehörigen Zeitfenster können nur innerhalb des Intervalls, in dem das Ereignis bearbeitet werden muss, vorkommen. Deshalb wird $Task_5$ einmal als ausgeführt markiert und das nächste Mal unverändert als fehlend belassen. Bei nicht periodischen Aufgaben wird angenommen, dass diese innerhalb der nächst möglichen Zeitfenster ablaufen sollen und wurden dort vermerkt.

Nachdem der Moduswechsel signalisiert wurde, müssen zum einen zusätzliche neue Aufgaben hinzugefügt und zum anderen eine Zeitsteuerung eingeleitet werden. Dazu muss der Fortschritt aller Aufgaben auf ein Niveau gebracht werden, sodass der Zeitplan keine Lücken noch nicht erfüllter Aufgaben aufweist. Für das Beispiel gilt, dass zunächst alle Lücken bis $Task_4$ im Zeitplan geschlossen werden müssen. Dies sind $Task_1, Task_2^2, Task_3^1, Task_4$. Das sorgt für den Umschwung von einer ereignisgesteuerten zu einer zeitgesteuerten Ausführung. Dieser Ausgleich muss stattfinden, da die Reihenfolge der Intervalle im Zeitplan vorher anhand der Abhängigkeiten der zugehörigen ABBs berechnet wurde. Ein Versuch die Aufgaben sofort in dieser Reihenfolge auszuführen, vermischt potentiell kritische Bereiche: Beispielsweise würde dies in Abbildung 3.1 auftreten, wenn sich $Task_1$ und $Task_2^2$ von der gleichzeitigen Ausführung durch eine Sperre schützen und ohne Ausgleich nach der Signalisierung des Wechsels als erstes $Task_1$ eingeplant wird, bevor $Task_2^2$ beendet wurde. Ein *Nachholen* der Aufgaben wird eingeleitet.

Ein lückenlos erfüllter Zeitplan ist eine sehr starke Voraussetzung und kann etwas abgeschwächt werden, indem alle kritischen Abschnitte abgeschlossen werden. Danach kann die Ausführung der Reihenfolge im Zeitplan folgen, da dieser unter Beachtung der gerichteten Abhängigkeiten erstellt wurde. Nehmen wir wieder an, dass ein kritischer Abschnitt in $Task_2^2$ existiert und alle anderen Aufgaben nicht geschützt sind. Dann reicht es $Task_2$ zu beenden und danach den Rest in der Reihenfolge, wie sie im Zeitplan stehen, auszuführen.

Neue Aufgaben müssen gesondert behandelt werden, da sie vollständig nachgeholt werden müssen und zusätzlich mit weiteren Aufgaben durch Synchronisation zusammenhängen, womit die neuen Aufgaben nicht isoliert, sondern anhand der Abhängigkeiten teilweise gemeinsam hinzugefügt werden müssen.

Ein weiterer Aspekt ist das Einhalten der Termine der Aufgaben. Wenn wie oben beschrieben vorgegangen wird und nur die Aufgaben in kritischen Abschnitten beendet und danach in einer statischen Reihenfolge vorgegangen werden würden, sind Termineinhaltungen nicht mehr garantiert. Da das Nachholen von alten und neuen Aufgaben erforderlich sein kann, beispielsweise sind $Task_1$ und $Task_3^1$ im Zeitplan früh eingeplant, aber in der tatsächlichen Ausführung übersprungen worden, müssen diese vor dem Eintreten der jeweiligen Termine erfüllt sein. Sobald die Reihenfolge der Aufgaben im Zeitplan nicht nach dem frühsten Termin geordnet entstanden ist, muss eventuell eine andere Reihenfolge verwendet werden um die Termine zu wahren.

Durch eine Umordnung der Aufgaben können wiederum kritische Bereiche betreten oder auf Signale gewartet werden, was die Ausführung von anderen Aufgaben verhindert, womit weitere Umordnungen notwendig werden.

Praktischerweise werden letztere Umordnungen durch die Abhängigkeiten eingeleitet und sind damit begrenzt. Es wird also ein abhängigkeitsbewusster Ablaufplaner benötigt, um die Termine einzuhalten, der gleichzeitig versucht nach Zeitplan zu arbeiten, um in eine Lage zu kommen in der die vollständige Zeitsteuerung eintreten darf. Eine Ausführung von Aufgaben ohne die explizite Synchronisation erfordert, dass der Ablaufplaner den Abhängigkeitsgraphen traversiert und selbst auf die Einhaltung der Abhängigkeiten achtet. Dafür muss allerdings ein erheblicher Aufwand geleistet werden , weshalb sich es sich anbietet, Aufgaben mit Code für Ereignissteuerung zu wählen, da durch ihre Synchronisation der Graph entsteht und der Ablaufplaner diese dann sowohl anhand ihres Termins als auch des Zeitplans auswählt.

Daher wird eine *Transitionsphase* zwischen ereignisgesteuerter und zeitgesteuerter Ausführung mit zeitlich gesteuerten Ereignissen eingeführt, welche die Aufgaben durch ihre Einteilung im Zeitplan zu den vorgegebenen Zeitpunkten als ausführbar markiert, doch die Auswahl des nächsten Aktivitätsträgers mithilfe traditioneller ereignisgesteuerter Algorithmen trifft.

Eine genau Beschreibung des Algorithmus findet sich in Kapitel 4.

3.3 Aufgabenänderung

Es sollen nun die möglichen auftretenden Änderungen der zu erledigenden Aufgabenmengen erarbeitet werden.

In Übereinstimmung mit bisherigen Moduswechseln, wie in Abschnitt 2.1.2.3 präsentiert, betrachten wir für beide Modi jeweils eine Menge von Aufgaben mit eigenen Parametern. Wir gehen hierbei von einer Überschneidung der Aufgabenmengen aus, sodass die prinzipiell die gleichen Aufgaben erfüllt werden müssen. Das bedeutet, dass dieselben Regelungsschritte und Sensormessungen benötigt werden, aber deren Bestandteile und Parameter variieren dürfen. Es bezeichnet E die Menge der Aufgaben im ereignisgesteuerten Modus und T die Menge der Aufgaben im zeitgesteuerten Modus. Für diese Mengen gilt:

- Keine der beiden Mengen ist leer.
- Die Schnittmenge ist nicht leer.

Gilt eine der Annahmen nicht, entfällt mit dem dadurch unnötigen Übernehmen der Aufgaben im neuen Modus der Schwerpunkt dieser Arbeit und das von anderen Moduswechseln bekannte Problem des Beendens der alten Aufgaben vor oder nach dem Start der neuen muss gelöst werden.

Die Änderung der Bestandteile einer Aufgabe sowie das potentielle Entfallen von alten und Hinzukommen von neuen Aufgaben kann anhand der Auswirkungen im ABB-Abhängigkeitsgraphen dargestellt werden. Systeme können ohne diese Änderungen auskommen, wodurch auf die sonst notwendige Behandlung von Sonderfällen verzichtet wird, doch der geregelte Übergang bleibt unverändert und bezieht die vorhandenen Abhängigkeiten und Aufgaben genauso mit ein.

Die Art und Weise auf die Aufgaben verändert werden können, zeichnet sämtliche weitere Überlegungen aus. Es soll angenommen werden, dass Änderungen von Aufgaben nur die von ihnen durchgeführten Unteraufgaben wechselt, aber die Unteraufgaben selbst nicht verändert. Dies bedeutet, dass alle Unteraufgaben vor und nach dem Wechsel für dieselben Berechnungen zuständig sind. Benötigt eine Aufgabe mehr oder weniger Berechnungen im sicheren Modus, so soll dies durch wegfallende oder hinzukommende Unteraufgaben umgesetzt werden, also innerhalb einer Unteraufgabe keine neuen Berechnungen hinzukommen und keine alten entfallen. Dies sorgt dafür, dass die Änderungen innerhalb einer Aufgabe gering sind, womit sich die Problematik auf die veränderten Abhängigkeiten beschränkt. Sollten solche *lokalen* Veränderungen ermöglicht werden, so kann nicht garantiert werden, dass diese in der zum Moduswechsel aktuell laufenden Periode der Aufgabe wirksam sind, da jede Unteraufgabe beliebig weit fortgeschritten sein kann und Veränderungen von Berechnungen, welche bereits geschehen sind, retroaktiv nicht beliebig nachgereicht werden können. Dies würde eine inkonsistente Mischung der Unteraufgaben erzeugen, bei der jede Möglichkeit zugelassen sein muss.

Indem die Änderungen der Abhängigkeiten auf Basisfälle im ABB-Graphen abgebildet werden, kann das Verhalten des Moduswechsels rekursiv bezüglich dieses Aspekts genau definiert und die Umsetzbarkeit aller Fälle evaluiert werden. Um diese Betrachtung durchführen zu können, müssen

wir von den zwei ereignisgesteuerten Systemen ausgehen, dem normalen System und dem der Transitionsphase, da in der Zeitsteuerung die Abhängigkeiten verschwinden und nur implizit durch die Reihenfolge abgebildet werden. Dabei werden die Abhängigkeiten nicht am Ende eines Knoten erfüllt, sondern nur bei der Erstellung umgesetzt. Außerdem müssen die Aufgabenänderungen eingesetzt und die notwendigen Aufgaben wie bereits beschrieben nachgeholt werden, bevor die Zeitsteuerung aktiv wird.

Hierzu betrachten wir die Knotenpaare, an deren Abhängigkeiten Änderungen vollzogen werden, wobei mindestens ein Knoten in E vorhanden sein soll. Falls beide Knoten in T liegen, gilt durch das neuen Hinzukommen von beiden, dass sich keiner in Ausführung befinden kann, womit nach einem erfolgreichen Übernehmen der restlichen Aufgaben diese zwei nur im neuen Modus gestartet werden müssen. Übrig bleiben noch die einzelnen Knoten ohne Partner. Diese stammen entweder aus T und sind demnach neu, womit das gleiche Argument anwendbar ist, oder es handelt sich um potentiell bereits begonnene Abschnitte, welche nun verschwinden. Doch durch die fehlenden Abhängigkeiten - es handelt sich ja ausschließlich um einzelne Knoten - können diese ohne Auswirkungen abgesetzt werden.

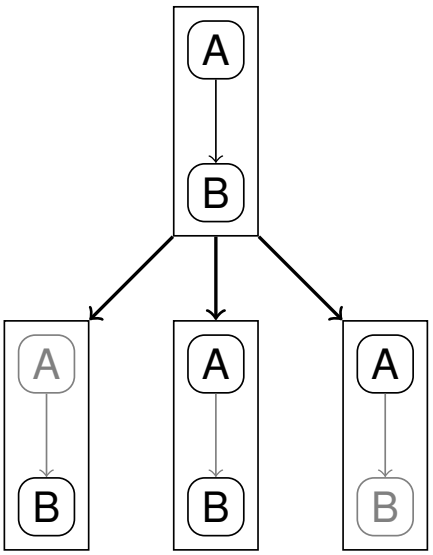
3.3.1 Unilaterale Synchronisation

In Abbildung 3.2 wird in den Rechtecken die Veränderung der Abhängigkeiten von ereignis- zu zeitgesteuerten Ausführung dargestellt. Die abgerundeten Quadrate repräsentieren die ABBs, wobei in grauer Farbe wegfallende und in roter hinzukommende Einheiten abgebildet werden. Wenn Abhängigkeiten, die dünnen, schwarzen Pfeile, verschwinden, muss es sich zum einen um Knoten aus E handeln, was bedeutet, dass diese - jeder umfasst mindestes eine Anweisung - beliebig weit fortgeschritten sein können. Bei grauen ABBs könnte es sich entweder um einzelne ABBs handeln, welche in einer Unteraufgabe nicht weiter benötigt werden, es kann aber auch eine vollständige Unteraufgabe sein, welche komplett entfernt wird.

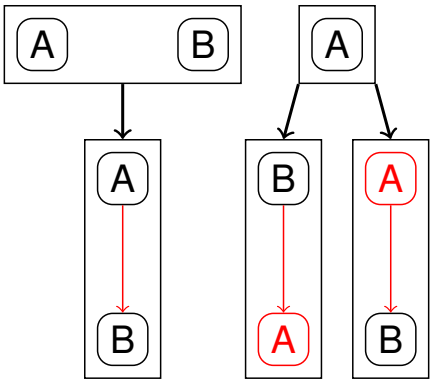
Auf das Entfernen einzelner ABBs sollte wenn möglich verzichtet werden, da diese innerhalb des Ausführstrangs eines Unteraufgabe liegen. Die Inkonsistenz der Aufgabe und Mischung der Berechnungen, wie oben beschrieben, würde genau dann eintreten, wenn ABBs mitten in der Ausführung einer Unteraufgabe verändert werden dürfen. Indem die Aufgaben vollständig beendet oder keine einzelnen ABBs innerhalb einer Unteraufgabe angefasst werden, wird dies vermieden. Damit sind keine Änderungen des Kontrollflusses erlaubt. Die Ausnahme bildet die Abbildung der Abhängigkeitsänderungen.

Einzelnen ABB abzubrechen bedeutet die Korrektur des Status der Unteraufgabe, wobei diese entweder die Bedeutung haben muss, dass der ABB vollständig oder überhaupt nicht ausgeführt worden ist. Bei ersterem wird entweder ein vorher berechneter Status verwendet oder der ABB zu Ende berechnet. Dies hat dieselbe Semantik wie eine Änderung zu einem Zeitpunkt, zu dem sich die Aufgabe nicht in Ausführung befindet. Dieser Ansatz hat Vorteile und bietet mehr Freiheiten, aber wird nicht weiter verfolgt.

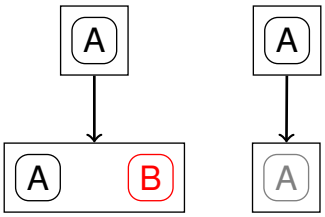
Stattdessen wird eine alternative Lösung verwendet, um zusätzliche Verwaltungsinformationen zu reduzieren. Durch das Ausgliedern des zu entfernenden Teils in eine einzelne Unteraufgabe kann diese entweder mit oder ohne Abhängigkeitsänderungen beendet oder abgebrochen werden. Das Abbrechen erfordert potentiell ein aufwändiges Zurücksetzen, das zu jeder Zeit des regulären, ereignisgesteuerten Ablaufs aktiv unterstützt werden muss, um die Konsistenz gemeinsamer Daten zu gewährleisten. Dies ist allerdings sehr viel einfacher als einen ABB innerhalb einer Unteraufgabe zurückzusetzen, indem für jede Unteraufgabe durch RTSC eine entsprechende Funktion generiert wird. Beim Fortsetzen muss allerdings die Interaktion mit anderen Aufgaben beachtet werden.



(a) Abhängigkeiten können fallen gelassen werden



(b) Es können neue entstehen



(c) Unabhängige Daten sind ebenfalls möglich

Abbildung 3.2 – Mögliche Änderungen unilateraler Natur

Somit wird ausschließlich das Entfallen gesamter Unteraufgaben erlaubt. Zu welchen Zeitpunkt Unteraufgaben entfallen dürfen wird später geklärt, da es dieselben beziehungsweise sehr ähnliche Informationen wie für das Beenden gesamter Aufgaben benötigt.

Die wegfallenden Abhängigkeiten sind in Abbildung 3.2a dargestellt.

Im den mittleren Teil wird eine Abhängigkeit zwischen zwei Knoten entfernt, deren Aufgaben in beiden Modi benötigt werden. Damit ist die Reihenfolge nicht länger fest vorgegeben.

Angenommen der Block A wurde bereits erfüllt, dann wurde B ebenfalls bereits aktiviert und es können keine weiteren Schwierigkeiten an dieser Abhängigkeit entstehen, da die Interaktion mit dem Folgeknoten abgeschlossen ist. Falls A allerdings noch nicht beendet worden ist, kann er sich entweder mitten in Ausführung befinden oder der Knoten noch nicht gestartet worden sein. Ist A schon in Ausführung, wird ein Entfernen der Abhängigkeit innerhalb des Programmcodes technisch nicht verlässlich möglich, sondern muss auf Ebene des Systemaufrufs stattfinden oder zugelassen werden und erst im Nachhinein angepasst werden. Wird der Knoten A noch gestartet, muss die Abhängigkeit gar nicht ausgeführt werden.

Allerdings darf B in keinem Fall auf eine entfernte Abhängigkeit warten, weshalb der Status von B relevant ist. Dafür wird unterschieden zwischen dem Auslösen eines logischen Ereignisses, das die Aufgabe mit ABB B startet, und dem Senden eines Signales, welches den weiteren Fortschritt von B zulässt.

Falls die Unteraufgabe von B noch nicht läuft und durch A gestartet wird, also ein logisches Ereignis eintritt, dann muss B im neuen Modus von einer anderen Aufgabe gestartet werden. Dazu wird entweder der neue Auslöser noch aktiviert oder die Aufgabe B wegen des Fortschritts der auslösenden sofort lauffähig. Im ersten Fall muss der Auslöser wie jede andere neue Abhängigkeit eingebaut werden und im zweiten, wenn es schon zu spät für den Einbau des neuen Auslöser ist, die Aufgabe B automatisch freigeschaltet werden. Falls es sich um ein Signal handelt, kann B entweder noch nicht ausgeführt oder schon begonnen worden sein. Im letzteren Fall wartet B eventuell bereits auf einem Signal, weshalb das Warten entweder vorher entfernt oder das Signal simuliert werden muss, womit die Unteraufgabe von B sofort geweckt und lauffähig wird. Eine noch nicht wartende Aufgabe kann auf das Signal verzichten.

Bisher wurden nur Punkt-zu-Punkt Signale besprochen, bei denen pro Senden maximal eine Aufgabe erreicht wird. Falls ein Signal mehrere Aufgaben erreichen soll, darf nur das Warten bei dem Empfänger entfernt werden, welcher die Abhängigkeiten nicht mehr benötigt, und nicht das Senden, welches von den anderen Aufgaben noch benötigt wird. Ebenfalls gibt es Signale, welche mehrere erreichen sollen, aber nur Auswirkungen auf einen der Wartenden haben. Beispielsweise werden Nachrichten nur von einer Aufgabe konsumiert oder es wird nur genau eine Aufgabe von Semaphoren aufgeweckt, auch wenn sie in der API noch nicht vorkommen, aufgeweckt wird. In diesen Fällen entsteht durch das Entfernen eines Konsumenten eine Überzahl an Signalen, sodass jeweils ein Sender und Empfänger entfernt werden muss. Dafür soll vor der Laufzeit festgelegt werden, welche Kombinationen zusammenhängen, sodass jeweils Paare gemeinsam entfernt werden.

Im linken Teil des Beispiels wird der ABB A nicht weiter benötigt. Es muss sich bei A und B um unterschiedliche Unteraufgaben handeln, da A komplett entfällt und keine lokalen Modifikationen erlaubt sind. Nun soll die Unteraufgabe von A vollständig entfernt werden, wozu für alle Abhängigkeiten genauso vorgegangen werden muss wie im vorherigen Fall. Das bedeutet, dass keine andere Aufgabe, hier die von B, an irgendeiner Stelle auf Aufgabe A warten darf, nachdem A entweder durch eine spezielle Funktion abgebrochen oder beendet wurde. Dazu müssen wie oben entweder die Warteoperationen entfernt oder das Aufwecken simuliert werden.

Der Fall, in dem ein abhängiger Knoten entfernt wird, wie in Abbildung 3.2a rechts, ist einfacher zu handhaben. Hierbei wurde die Abhängigkeit entweder bereits erfüllt oder noch nicht. Das Erfüllen sorgt entweder für das Auslösen eines logischen Ereignisses, welches die Aufgabe von B freischaltet, oder es setzt eine unterbrochene Ausführung durch ein Signal fort. In beiden Fällen kann B bereits gestartet worden sein und muss beendet werden. Falls ABB B noch auf seine Abhängigkeit wartet oder beziehungsweise schon gestartet wurde, so muss er kontrolliert beendet werden. Dies erfordert zum einen eventuell ein Aufwecken von B und zum anderen entweder eine Abbruchsfunktion oder ein reguläres Beenden der Aufgaben.

Nun können Aufgaben alle drei Fälle beinhalten und müssen auf all diese reagieren. Für einen reibungslosen Übergang stützt sich eine einfache Umsetzung, welche keine Veränderung innerhalb der Anwendung benötigt, auf modifizierte Signalbehandlung im Betriebssystem. Dafür wird mit einem weiteren Parameter angegeben, in welchen Modi die Abhängigkeit vorhanden ist. Sobald der Moduswechsels stattfindet, wird bei entfernten Abhängigkeiten zum einen jede wartende Unteraufgabe aufgeweckt und zum anderen die alte Synchronisation ignoriert. Wenn ein logisches Ereignis entfernt werden soll, wird dieses wie oben entweder automatisch freigegeben oder in einer neuen Aufgabe eingebaut.

Das Verfallen von gerichteten Abhängigkeiten erfordert eine sehr intensive Behandlung und Laufzeitunterstützung und sollte deshalb nur sparsam angewendet werden. Die Ausnahme bildet das Entfernen aller Aufgaben eines zusammenhängenden Gebietes der gerichteten Abhängigkeiten. Solange die durch im nächsten Abschnitt besprochenen ungerichteten Abhängigkeiten entstehenden kritischen Abschnitte kontrolliert beendet werden und einen sicheren Zustand hinterlassen, kann die gesamte Zusammenhangskomponente abgebrochen werden, da keine weiteren Interaktionen mit dem restlichen System stattfinden.

Als nächstes werden die Möglichkeiten für hinzukommende Abhängigkeiten betrachtet. Im linken Teil der Abbildung 3.2b wird vom ABB A eine Kante zum Knoten B gebildet. Falls A bereits beendet wurde, ist dieses Signal nicht versendet worden oder B wartet nicht darauf und kann ebenfalls entweder bereits begonnen worden sein oder nicht. Unabhängig davon welcher Fall eintritt, funktioniert das System im normalen Modus ohne diese Abhängigkeit. Der einzige Unterschied ist eine induzierte Reihenfolge, welche vorerst ignoriert werden kann, um schneller in den zeitgesteuerten Modus zu wechseln. Deshalb wird die Annahme getroffen, dass die Ausführung von B grundsätzlich ohne Voraussetzung an A möglich ist. Um diese Abhängigkeit dennoch zu übertragen, müssen für Signale optional zusätzliche Operationen an den ABB-Grenzen ermöglicht werden. Diese können dann erst zu einen gemeinsamen Zeitpunkt aktiviert werden, sodass die Operation von beiden durchgeführt, also weder von Sender noch Empfänger übersprungen wird. Falls es sich um den Start der zu B zugehörigen Aufgabe handelt, also ein logisches Ereignis, gilt die gleiche Behandlung wie für Signale: Die Umschaltung der Auslösung von B muss mit A synchronisiert werden.

Diese Arbeit geht davon aus, dass das System im ersten Modus ohne das Signal funktioniert hat und im folgenden ebenso abläuft. Beim Entfernen eines Ereignisses wurde bereits erklärt, wie das neue Ereignis von einem anderen oder dem Ablaufplaner ausgelöst werden muss.

Bei den nächsten beiden Fällen in Abbildung 3.2b, bei denen eine der beiden ABBs nur im kritischen Modus ausgeführt werden muss, kann nur der rechte, bei dem die Abhängigkeit von einem neuen ABB ausgeht, analog behandelt werden. Dabei wird wie oben davon ausgegangen, dass B ohne die Abhängigkeit von A im ereignisgesteuerten Betrieb umgesetzt worden ist und dies auch weiterhin möglich ist, also wird diese weiterhin als optional angenommen und nur nach dem vollständigen Wechsel aktiv.

Allerdings entstehen Schwierigkeiten im mittleren Fall der Abbildung 3.2b. Es muss von B ein zusätzliches Signal gesendet werden, indem es entweder wie oben durch eine Möglichkeit hinzugeschalten werden kann oder permanent vorhanden ist. Es darf allerdings nur Auswirkungen nach dem Wechsel in die Transitionsphase haben. Unabhängig davon kann B wieder entweder beendet worden sein oder nicht. Im ersten Fall wird A durch das Erfüllen der Abhängigkeit nicht ausgelöst, womit A auf die nächste Periode von B warten muss. Wenn B eine Aufgabe aktiviert, kann diese abgewartet werden. Wenn allerdings A bereits gestartet wurde, kann die Aufgabe auf die Abhängigkeit warten, obwohl B bereits vor dem Wechsel beendet wurde und daher nie das Signal sendet.

Dies wird momentan in Kapitel 4 akzeptiert, aber sollte in weiteren Entwicklungen angepasst werden. Denn das genaue Verhalten ist vollkommen vom Zustand zum Zeitpunkt des Wechsels abhängig und sollte durch einen vorhersehbareren Verlauf ersetzt werden. Beispielsweise kann ein Zeitraum nach dem Abschluss der Abhängigkeit eingesetzt werden, in dem diese weitergereicht werden kann. Also wird die Abhängigkeit, sollte sie vor dem Wechsel ausgelöst werden, für eine festgelegten Zeit als aktiv behandelt und in dem Fall, dass ein Wechsel initiiert wird, an A weitergegeben werden, sodass sie nicht verloren geht. Die Semantik würde sich damit mehr dem Sperren eines Mutex oder der einer Semaphore anpassen.

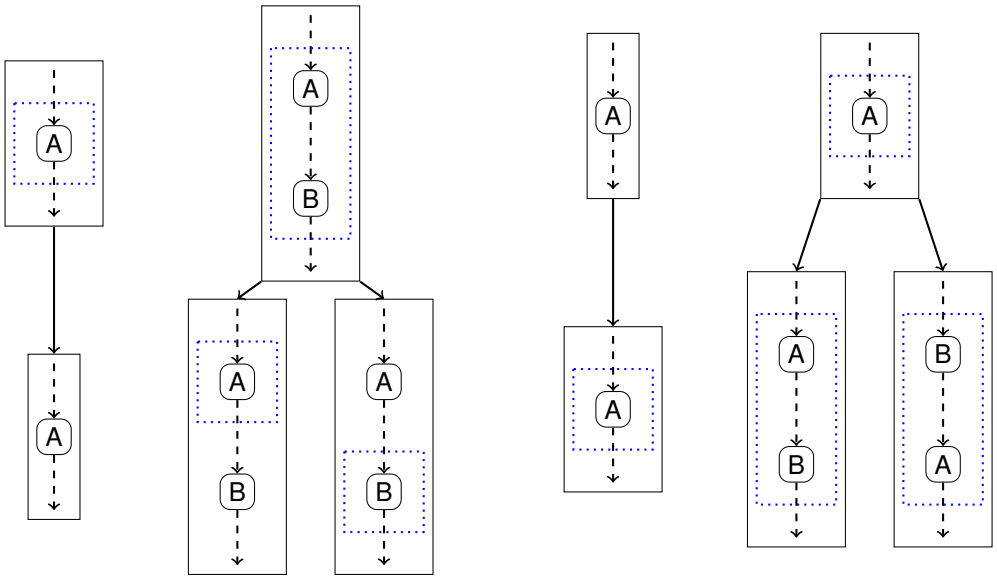
Abschließend ist noch die Manipulation von unabhängigen Knoten in Abbildung 3.2c zu sehen. Durch die fehlenden Abhängigkeiten können wie bei dem bereits erwähnten Entfernen einer starken Zusammenhangskomponente keine Auswirkungen auf andere Aufgaben entstehen, sodass diese problemlos eingefügt beziehungsweise entfernt werden können. Im letzten Absatz wurde die Abhängigkeit der Ausführung von bestimmmten ABBs vom genauen Zeitpunkt des Wechsels erläutert. Es gilt zu beachten, dass dies für sämtliche neu hinzukommenden Knoten gilt, weshalb hierfür eine Lösung gefunden werden muss. In dieser Arbeit wird der Ansatz verfolgt, bei dem sämtliche Aufgaben, welche noch erfüllt werden könnten, zum Zeitpunkt des Wechsels bereit werden. Der Vorteil dieses Systems ist es, dass möglichst viel ausgeführt wird, und der Wechsel in einem sicheren Zustand schneller vollzogen werden kann. Allerdings gefährdet dies die Planbarkeit und muss noch weiter untersucht werden.

3.3.2 Multilaterale Synchronisation

Im Bezug auf ungerichtete Abhängigkeiten können zum einen die kritischen Bereiche erweitert oder verkleinert werden, sodass die Menge der geschützten ABBs angepasst wird. Zum anderen sind Veränderungen innerhalb eines kritischen Bereichs möglich. Dabei können Abhängigkeiten beider Type entweder entstehen oder verfallen.

In Abbildung 3.3 und Abbildung 3.4 werden kritische Bereiche nach dem Vorbild der RTSC-Literatur [Sch11] in blaue gepunkteten Umrandungen dargestellt, wobei die gestrichelten schwarzen Pfeile den Kontrollfluss abbilden, welcher wie zu Beginn erläutert, nicht weiter behandelt werden muss.

Die Verkleinerung eines kritischen Abschnittes, deren Möglichkeiten in Abbildung 3.3a dargestellt werden, kann stattfinden, bevor dieser betreten wurde, währenddessen oder nach seinem Abschluss.



(a) Kritische Bereiche können kleiner werden

(b) Kritische Bereiche können größer werden

Abbildung 3.3 – Mögliche Änderungen multilateraler Natur

Im ersten Fall kann mithilfe einer Laufzeitunterstützung, wie bei hinzukommenden und wegfallenden unilateralen Primitiven erforderlich[12], das Betreten verzögert werden oder gar nicht stattfinden. In Abbildung 3.3a bedeutet dies, A wurde noch nicht ausgeführt und es muss entweder der Systemaufruf am Ende von B im mittleren respektive der am Beginn von A im rechten verschoben, oder im linken Falle beide vom ABB A entfernt werden. Letztere Behandlung trifft auch auf die Entfernung von kritischen Abschnitten mit mehreren ABBs zu.

Wurde der Abschnitt bereits abgeschlossen, so tritt die Anpassung frühestens zum nächsten Aufruf ein und benötigt außer der Verschiebung der Aufrufe der Synchronisationsprimitiven wie im vorherigen Fall keine weitere Unterstützung zum Moduswechsel.

Befindet sich der Aktivitätsträger allerdings innerhalb eines geschützten Bereiches, so wird gerade ein Block ausgeführt, welcher im neuen Modus noch Teil des kritischen Abschnittes ist. In der Abbildung 3.3a ist dies im mittleren bei angefangener aber nicht abgeschlossener Bearbeitung von A und im rechten bei der von B möglich. Letzteres wird behandelt, indem der Block zu Ende geführt und danach wie bei einem abgeschlossenen kritischen Abschnitt vorgegangen wird. Soll der Mutex allerdings vor dem geplanten Ende freigegeben werden (Mitte), so kann diese Freigabe entweder eingeschoben werden wie bisher, oder erst zu der nächsten Periode ermöglicht werden. Weiterhin kann ein Block in Ausführung sein, der im neuen Modus kein Teil des Abschnittes ist. Dann wird dieser entweder erst später betreten oder hätte bereits verlassen werden sollen. Beide Fälle werden am besten behandelt, indem dieser Schritt erst zur nächsten Iteration genommen wird, da sonst die Sperre manuell zum aktuellen Zeitpunkt freigegeben werden müsste.

All diese Fälle könnten unterstützt werden, allerdings erweist sich eine tatsächliche Umsetzung durch ein Betriebssystem als schwierig, da das häufig notwendige Verschieben der Grenzen aufwändig und die Verifikation der Auswirkungen durch das Grundproblem des Wechsels, der unvorhersehbare Zustand der Aufgaben, ebenfalls nur schwierig ist. Erfreulicherweise ist die Implementation dieser überflüssig, weil ein Wegfallen eines kritischen Abschnittes nur möglich ist, wenn kein weiterer Zugriff auf vor dem Wechsel geschützte Daten mehr stattfindet. Dies kann durch die Annahme, dass der Kontrollfluss innerhalb einer Unteraufgabe durch einen Wechsel nicht weiter verändert wird, also der Zugriff bei dem verkleinerten kritischen Bereich immer noch vorhanden ist, nur stattfinden, wenn alle anderen Zugriffe auf diese Daten im zeitgesteuerten Modus nicht mehr auftreten, also alle Unteraufgaben mit Zugriffen entfallen. Somit gelingt die Sperrung durch die verbleibende Unteraufgabe, welche ihren kritischen Bereich anpasst, immer. Der Zusatzaufwand durch die nicht entfernten Synchronisationen sind auf die Systemaufrufe beschränkt und damit gering, sodass das Entfernen diese Kosten sogar übersteigen könnte.

Beim Erweitern von kritischen Abschnitten kann sehr ähnlich vorgegangen werden: Befindet sich der Aktivitätsträger bereits in diesem, so muss maximal im mittleren Falle der Abbildung Abbildung 3.3b die Freigabe verschoben werden. Rechts ist die Verschiebung erst in der nächsten Periode relevant. Wurde der kritische Abschnitt bereits verlassen, dann ist wiederum nur der mittlere Fall relevant, bei dem ein Teil von B potentiell ungeschützt berechnet wird. Bei der Erzeugung eines neuen kritischen Abschnitts, gelten dieselben Aussagen wie bei den Erweiterung: Eventuell wurde ein Teil des nun zu schützenden Abschnitts bereits ausgeführt, er kann noch nicht betreten worden sein, oder wurde schon beendet.

Auch in diesen Fällen ist die Implementierung sehr schwierig und ebenfalls überflüssig. Sowohl das Erweitern als auch das Erzeugen von neuen kritischen Abschnitten ist nur notwendig, falls durch den Wechsel zusätzliche Zugriffe auf gemeinsamen Speicher entstehen, welcher in ereignisgesteuerter Ausführung von genau einer Unteraufgabe benötigt wurde. Wäre dieser Speicher von mehreren verwendet worden, so wäre er bereits geschützt. Damit kann die neue Sperre im ereignisgesteuerten

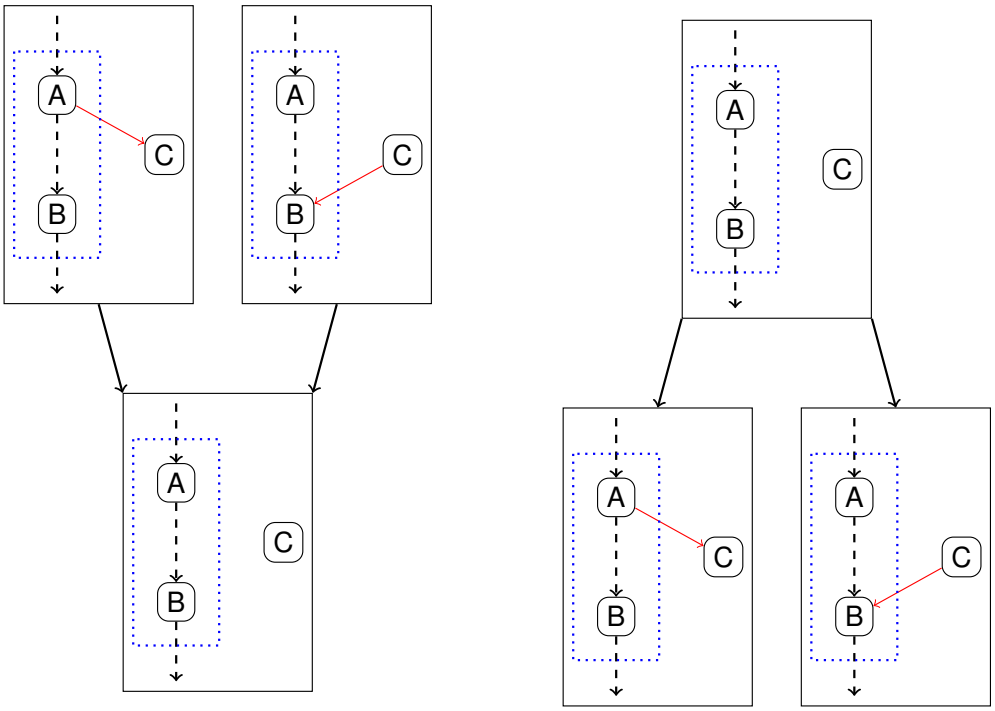
[12]Die entsprechenden Aufrufe müssen im neuen Modus eingefügt bzw. entfernt werden.

Modus immer beim ersten Versuch erlangt werden, da diese in dem Modus nur einen Nutzer hat, womit der Zusatzaufwand, diese immer zu verwenden, genauso gering wie oben ist. Deshalb wird auf eine Vergrößerung von kritischen Abschnitten in ereignisgesteuerten Modus vorkommenden Aufgaben verzichtet.

Bisher wurden nur kritische Abschnitte in einer Unteraufgabe beachtet. Es können allerdings auch gesamte Unteraufgaben mit diesen Bereichen entfallen oder entstehen. Bei letzterem können keine Komplikationen auftreten, da die Aufgabe erst gestartet wird und kein Mutex gesperrt worden sein kann. Das Entfallen von Aufgaben weist dieselben Probleme auf wie die lokalen Veränderungen an kritischen Abschnitten, weshalb die Aufgaben regulär oder durch eine spezielle Abbruchfunktion beendet werden müssen. Erst danach können sie entfallen.

Zuletzt sind die Veränderungen der unilateralen Synchronisation innerhalb geschützter Bereiche in Abbildung 3.4 zu sehen. Außerhalb tritt die exakt gleiche Behandlung wie ohne Sperren ein, da Signale eine lokale Bedeutung für eine Instruktion besitzen und keine Auswirkungen auf den Bereich haben. Solange Unteraufgaben im Zuge der Veränderungen von gerichteten Abhängigkeiten nicht entfallen sollen, wie oben angenommen, werden Veränderungen innerhalb geschützter Bereiche genauso behandelt wie außerhalb, da eine Aufgabe maximal länger wartet, eine andere Aufgabe freigibt, oder dies jeweils entfällt und kein Einfluss auf den kritischen Bereich selbst hat.

Für geschachtelte Sperren kann genauso vorgegangen werden wie für nicht geschachtelte, da sämtliche Änderungen ausgeschlossen wurden.



(a) Abhängigkeiten können fallen gelassen werden

(b) Es können neue entstehen

Abbildung 3.4 – Mögliche unilateraler Natur innerhalb geschützter Bereiche

3.3.3 Parameteränderungen

Innerhalb eines laufenden Systems müssen Parameteränderungen generell zu festen Zeitpunkten stattfinden. Es kann nicht erlaubt werden, dass diese zu beliebigen Zeitpunkten durch einen Wechsel initiiert werden, da ein Zusammenhang des Systems durch Synchronisationen besteht, welches durch eine Verschiebung einzelner gestört werden würde. Es kann die Periode und der Termin jeweils verkürzt oder verlängert, die Phase verschoben, oder die WCET angepasst werden. Abhängig vom verwendeten Einplanungsalgorithmus kann jede Veränderung eines Parameters während der Ausführung einiger Aufgaben zu einem nicht planbarem System führen, wenn beispielsweise einige Periodenverkürzungen durchgeführt werden, aber Verlängerungen der Perioden anderer Aufgaben erst später stattfinden würden. Die Verkürzung von Periode, Deadline, Phase oder WCET kann bewirken, dass eine Aufgabe noch nicht vollendet wurde, und schon beendet sein beziehungsweise wieder gestartet werden sollte, sich aber noch in Ausführung befindet. Das jeweilige Verlängern ist in diesem Aspekt weniger problematisch. Allerdings muss auf die Abhängigkeitsänderungen geachtet werden.

Aufgrund gerichtete Abhängigkeiten müssen alle zusammenhängenden Aufgaben zeitgleich ihre Parameter anpassen, da sonst entweder zu viele beziehungsweise zu wenige Ereignisse oder Signale ausgelöst werden könnten und auf die falsche Anzahl gewartet wird, womit ein endloses oder sehr langes Warten initiiert wird. In Abbildung 3.5 ist die Verschiebung der Intervalle, in welchen eine Aufgabe ausgeführt werden kann, zu sehen, wie sie bei nicht synchronisierter Anpassung der Parameter entsteht. Beispielsweise hat $Task_1$ die doppelte Frequenz von $Task_2$, wobei die beiden von einander abhängen. Dann kann $Task_1$ nicht nach einer ungeraden Anzahl an Zyklen seine Periode verändern, da der Zusammenhang mit $Task_2$ gestört werden würde. Ein Verdoppeln der Periode sorgt dafür dass kein Anfang und Ende der beiden Aufgaben zum gleichen Zeitpunkt stattfindet und ein Halbieren dafür, dass im selben Zeitraum $Task_1$ einmal öfters ausgeführt wird als $Task_2$. Die Auswirkungen auf das technische System dahinter sind vollkommen unbekannt und anwendungsabhängig. Durch die Verschiebung werden die Signale zu falschen Zeitpunkten gesendet beziehungsweise erwartet und verzögern gerade bei der Verlängerung der Periode die wartende Aufgabe, oder es werden zu viele Signale gesendet, was eine Synchronisation mittels Semaphoren oder das Versenden von Nachrichten stört. Dies kann nicht bei ungerichteten Abhängigkeiten auftreten, da der Wettstreit um Sperren zwar potentiell erhöht ist, aber die Sperren genauso oft freigegeben wie gesperrt werden.

Diese zeitliche Parameteränderung muss zur Hyperperiode der zusammenhängenden Aufgaben stattfinden, denn nur zu diesem Zeitpunkt sind alle diese Aufgaben mit Sicherheit abgeschlossen und obige Situationen können nicht auftreten. Diese Hyperperiode entspricht nicht der aller Aufgaben, sondern kann pro Zusammenhangskomponente einmal vorkommen.

Dadurch würde sich allerdings eine noch stärkere Mischung von Aufgaben ergeben, da für verschiedene Aufgaben entweder ein alter oder ein neuer Parameter verwendet werden kann, was die Verifikationen des Systems erschwert. Ein zeitgleiches Anpassen der Parameter aller Aufgaben kann hingegen erst zur Hyperperiode stattfinden. Weiterhin muss in der Transitionsphase grundsätzlich durch die Überlappung der beiden Aufgabenmengen bereits mehr Arbeit erfüllt werden, als in den einzelnen Modi, sodass eine Erhöhung der Last durch Reduktion der Perioden den Wechsel weiter erschwert. Um die Planbarkeit zu gewährleisten, wird ein 2-Phasen System eingesetzt, welches zunächst den Übergang in ein zeitgesteuertes System ohne Parameteränderungen sondern ausschließlich mit Aufgabenänderungen vollzieht und danach zu festen, vorberechneten Zeitpunkten in ein zweites zeitgesteuertes mit beliebigen Parameteränderungen übergeht. Der Nachteil dieses Systems ist eine verzögerter Wechsel in den sicheren Modus mit allen gewünschten Änderungen.

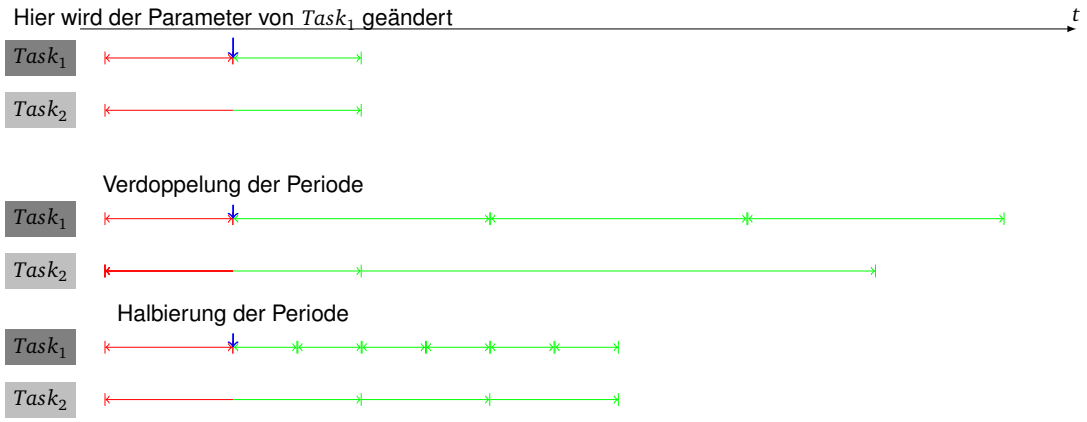


Abbildung 3.5 – Die roten Intervalle wurde bereits ausgeführt und die grünen zeigen die folgenden Intervalle

Doch es ist bereits vor der Laufzeit bekannt, welchen Zustand alle Aufgaben zu den bestimmten Zeitpunkten der Parameteränderungen besitzen, womit der Vorgang im Voraus geplant werden kann.

Wenn das System nicht ohne den sofortigen Einsatz neuer Parameter auskommt, gibt es noch die Möglichkeit die aktuell laufenden Aufgaben, bei denen dieser Fall eintritt, entweder abzubrechen, falls ein eimaliges Verpassen der Termine gestattet ist, oder diese zu beenden, aber zeitgleich die Aufgaben neu zu starten mit den angepassten Parametern. Die Möglichkeit verlangt geringe Perioden, um die Latenz gering zu halten und kann deshalb nur selten eingesetzt werden.

Vorerst werden Parameteränderungen zum Wechsel nicht akzeptiert und höchstens verzögert eingebracht.

3.3.4 Nachplanen von Aufgaben

Es wurde beschrieben wie neue Aufgaben hinzukommen können und wie Änderungen der Abhängigkeiten zwischen alten und neuen Aufgaben auftreten. Es wurde erklärt welche Bedeutungen diese haben und wie mit ihnen umgegangen kann, aber es wurde noch nicht darauf eingegangen, ob und wann die Aufgabenänderungen aktiv werden.

Zum einen müssen die neu hinzukommenden innerhalb ihres Termins abgeschlossen werden können und andererseits dürfen sie keine Abhängigkeiten zu anderen Aufgaben besitzen, welche noch nicht hinzukommen. Würde eine Aufgabe A hinzugefügt werden und abhängig von einer nicht nicht hinzugefügten Aufgabe B sein, so würde A warten, bis B zur nächsten Periode gestartet wird. Da A auf B wartet, sollte B entweder der Periode von A oder innerhalb des Intervalls zwischen Periode und Termin gestartet werden. Um dies zu gewährleisten müssen alle durch gerichtete Abhängigkeiten zusammenhängenden Aufgaben entweder nachgeholt werden oder erst später eingeplant werden. Wenn in diesem Beispiel der Termin des letzten Intervalls von B zum Zeitpunkt des Wechsels bereits abgelaufen ist und das nächste Intervall noch nicht gestartet hat, so kann A ebenfalls nicht gestartet werden, selbst wenn dessen Interval noch gültig ist.

Diesbezüglich wird vorerst angenommen, dass entweder kein Zusammenhang zwischen verschiedenen Aufgaben oder nur zwischen Aufgaben gleicher Parameter besteht, aber ein beliebiger Zusammenhang von Unteraufgaben einer Aufgabe wird zugelassen.

3.4 Informationsgewinnung zur Laufzeit

Im letzten Abschnitt haben wir die möglichen Änderungen der Aufgaben betrachtet, um die der Wechsel berücksichtigen soll. Dabei wurden Behandlungen für sämtliche Fälle vorgeschlagen, welche nun durchgeführt werden soll. Dazu muss allerdings der eintretende Fall zur Laufzeit bekannt sein, da jede Unteraufgabe mehrere Behandlungen und somit eine dynamische Anpassung erfordern kann. In Verknüpfung mit dem Abhängigkeitsgraphen kann diese Entscheidung anhand des Fortschritt einer Unteraufgabe mithilfe der Untergliederung in ABBs stattfinden. Dies ist der Punkt, an dem die Laufzeitunterstützung bei der Übersetzung durch das RTSC entsteht, indem die ABB-Informationen an den richtigen Stellen an das Betriebssystem weitergeben werden.

In Abbildung 3.1 wurde deutlich, wie der Status von Aufgaben und Unteraufgaben mithilfe der Intervalle, in welcher diese ausgeführt werden können, innerhalb des Zeitplans markiert wird. Dasselbe wird nun mit ABBs gemacht, da Unteraufgaben aufgeteilt im Zeitplan eingetragen werden können und die Unterteilung an ABB-Grenzen stattfindet.

Weiterhin kann der zeitgesteuerte Modus durch die Art und Weise der Zeitplanerstellung auch ABB-gesteuert genannt werden, da die Ausführung einer Unteraufgabe stets an ABB Anfängen beginnt und an deren Enden stoppt. Der Wechsel muss versuchen die ungeordnete Reihenfolge der auszuführenden, begonnenen, und ausgeführten ABBs in Einklang mit der fest vorgegebenen Reihenfolge der Zeitsteuerung zu bringen. Eine Visualisierung des Problems ist in Abbildung 3.6 zu sehen. Es gibt eine Menge von geplanten ABBs beliebiger Aufgaben mit beliebigen Abhängigkeiten in der Zeitsteuerung. Zum Zeitpunkt des Wechsels hat jeder dieser ABBs einen Status: Ausgeführt, begonnen, nicht begonnen. Um in den sicheren Zustand zu gelangen, den die Zeitsteuerung bietet, müssen die fehlenden ABBs nachgeholt werden. Falls ABBs der Ereignissteuerung (ABB_3,ABB_6)fehlen, werden diese garantiert ausgeführt, aber bei neu hinzukommenden, wie den ABB_0,ABB_7 muss dies nicht der Fall sein. Hier können entweder Aufgaben, welche vor dem Zeitpunkt des Wechsels begonnen werden sollten wie ABB_0 retroaktiv gestartet und eingeplant werden oder entfallen. Ob eine Aufgabe noch rechtzeitig beendet werden kann, sollte bei der Entscheidung erwogen werden. Auch beide Varianten können gleichzeitig auf einer pro Aufgaben Basis zum Zuge kommen. Für alle Aufgaben einer Zusammenhangskomponente bezüglich gerichteter Synchronisation sollte gleich vorgegangen werden. Das Einplanen von neuen Aufgaben wird stark erleichtert, indem der Zusammenhang von alten und neuen sowie zwischen neuen Aufgaben so wenig wie möglich durch Signale bedingt ist, und hauptsächlich durch das Auslösen von Ereignissen entsteht. Dies bezieht sich wohlgemerkt auf gesamte Aufgaben. Unteraufgaben einer Aufgabe können hingegen einen so hohen Zusammenhang haben wie sie benötigen, da mit dem Einplanen einer Aufgabe ihre Unteraufgaben ebenfalls hinzugefügt werden.

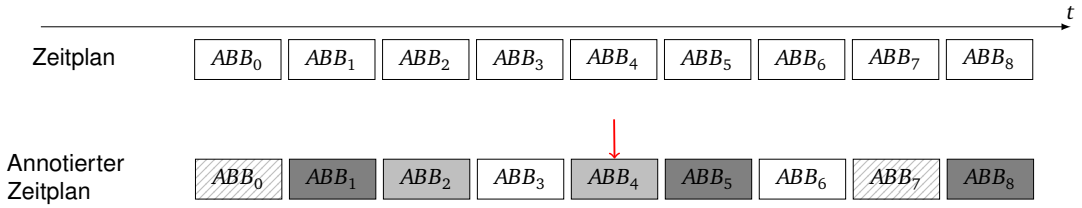


Abbildung 3.6 – Zum Zeitpunkt des Wechsels kann ein beliebiger Teil der geplanten ABBs entweder beendet (grau), begonnen (hellgrau) oder noch zu beginnen (weiß, weiß gestreift) sein

Deshalb wird der Fortschritt sämtlicher Aufgaben und Unteraufgaben mithilfe ihrer ABBs gemessen. Nachdem ein ABB abgeschlossen wurde, wird dies vom Ablaufplaner registriert und für einen Wechsel hinterlegt. Wird eine Aufgabe beendet, so verfallen alle Informationen über den Fortschritt und müssen für das nächste Intervall gesammelt werden.

Die Informationsübertragung an den Ablaufplaner kann auf mehrere Arten geschehen und basiert auf dem Fakt, dass ABBs durch die Synchronisation entweder mit Systemaufrufen beginnen oder mit diesen enden. Mit dem Beginn eines neuen ABBs wird der vorherige automatisch abgeschlossen und mit dem Ende eines ABBs natürlich ebenfalls. Daher bieten sich zwei Varianten für die Vermittlung des Fortschritts an:

- Mit jedem Systemaufruf wird als zusätzlicher Parameter eine eindeutige ABB Identifikationsnummer an den Ablaufplaner übergeben, welcher bei der Synchronisation grundsätzlich aufgerufen werden muss und dann die Daten in einer Verwaltungsstruktur eingeträgt.

- Diese ID kann auch über mit dem Ablaufplaner geteiltem Speicher vollzogen werden. Dabei schreibt der Aktivitätsträger vor dem Systemaufruf an eine vorgegebene Speicherstelle die ABB ID und der Ablaufplaner liest diese aus, wenn er durch den folgenden Systemaufruf aktiviert wird. Hierbei ist zu beachten, dass es sich noch um eine präemptive Ausführung handelt und eine Wettlaufssituation zwischen dem Schreiben der ID und der Synchronisation entstehen kann, weshalb die ID erst bei letzterer Operation und nicht bei jedem Aufruf des Ablaufplaner verwertet werden darf.

Aus Perspektive der Funktionalität unterscheiden sich die beiden Ansätze nicht, aber bei ersterem müssen im Betriebssystem spezialisierte Systemaufrufe implementiert werden, welche ein zusätzliches Argument verarbeiten können.

Das Speichern der Fortschrittsinformationen ist ein weiterer Grund, weshalb die Parameteränderungen der Aufgaben vermieden werden sollte. Denn danach sind ausgeführte Blöcke nicht mehr einfach im Zeitplan zu orten, da sich die Intervalle, in denen die Aufgabe ausgeführt werden kann, verschieben. Bei der zeitgleichen Änderungen der Parameter für zusammenhängende Aufgaben kann die Informationsverwaltung noch verwendet werden, ein freies Anpassen ist hingegen nicht möglich. Zur Verwirklichung werden die Informationen dann anhand der Intervalle zur Ereignissteuerung gespeichert und werden erst zum Änderungszeitpunkt mit dem Zeitplan vereint.

3.5 Abbildung auf ein technisches System

Um den Aufgabenwechsel in einem laufenden Betriebssystem zu unterstützen müssen die Anwendungen der drei Modi, Ereignis-, Zeit-, und Transitionsphase, in ein gemeinsames System übertragen werden. Dabei bekommt jede Unteraufgabe ihren eigenen Aktivitätsträger, die falls nötig über gemeinsamen Speicher verfügen. Weiterhin sollen auch im zeitgesteuerten Bereich einzelne Unteraufgaben nicht vermischt werden, was durch die Serialisierung möglich wäre. Es gibt mehrere Möglichkeiten, wie die Modi denkbar sind:

- Pro Modus gibt es einen einzelnen Aktivitätsträger, sodass die Aufgaben strikt getrennt sind.

- Die zwei Modi nach Ereignissteuerung sind in einem Aktivitätsträger und die Zeitsteuerung ihren eigenen.

- Alle Modi sind in einem Aktivitätsträger vereint.

Das Problem bei einer Trennung ist, dass der jeweilige Status der Unteraufgabe nicht einfach in den nächsten Modus übertragen werden kann, da es sich um unterschiedlichen Quellcode handelt und der Aufrufstapel im Speicher an den Aktiviätsträger gebunden ist. Eine Übertragung des Speichers wird notwendig, um keine Berechnungen zu verlieren. Weiterhin wären Sperren nicht verwendbar, da diese nur von dem sie haltenden Aktivitätsträger freigegeben werden können. Modifikationen daran können möglich sein, allerdings muss dies auf Protokollbasis entschieden werden. Wenn nur die Zeitsteuerung ihren eigenen Faden bekommt, kann der vollständige Wechsel frühestens zur Periode jeder Aufgabe durchgeführt werden, die Gründe bezüglich des Speichers treffen ebenfalls zu. Als letztes bleibt die Verwendung eines einzelnen Aktivitätsträger übrig, der die notwendigen Operationen für jeden Modus unterstützen muss. Dies sind zum einen die veränderten Synchronisationen in der Transitionsphase und zum anderen das Entfernen aller Synchronisation für einen ausschließlich zeitgesteuerten Modus.

Für die erste Aufgabe müssen sowohl die erlaubten Synchronisationsänderungen aus Abschnitte 3.3.1 bis 3.3.2 und die neuen Aufgaben bearbeitet werden. Bei beiden kann dies im Allgemeinen nicht zu beliebigen Zeitpunkten erfolgen, da sonst nur Teile von Abhängigkeitsänderungen aktiv sind, wie für neue Aufgaben beschrieben eventuell Abhängigkeiten fehlen und die Aufgabe damit warten muss und ihren Termin verpasst. Dabei wird ebenfalls das Verwalten der Fortschrittsinformationen gestört, da die Zuordnung des ausgeführten Blocks nicht mehr in dem Intervall stattfindet und zu keinem Zeitfenster passen wird. Dafür müssen für neue und alte Aufgaben die Zusammenhangskomponenten berechnet werden, sodass jede Komponente im gesamten ihre Änderungen zu einem Zeitpunkt aktiviert.

Vorerst werden die Synchronisationsänderungen auf hinzukommende Signalisierung, hinzukommende und entfallende Ereignisse und zusätzlicher kritische Bereiche in den neuen Unteraufgaben beschränkt. Aufgaben dürfen nur beendet werden, wenn durch Signale zusammenhängender Aufgaben entfallen. Es können also nur Unteraufgaben zu einer alten oder komplett neuen Aufgabe hinzugefügt werden. Um den Wechseln zwischen der Aufgabenmenge in der ereignisgesteuerten Phase und der Transitionsphase zu ermöglichen, müssen zusätzliche Signale eingebaut werden können. Es können nur Signale von alten zu neuen (Unter-)Aufgaben entstehen, weshalb zwei Möglichkeiten für den Einbau zur Verfügung stehen:

- Wie in Abschnitt 3.3.1 beschrieben, wird jedem Signal ein zusätzlicher Parameter übergeben, zur Unterscheidung, ob das Signal nur im zeitgesteuerten oder in beiden Modi benötigt wird. Damit können alle Systemaufrufe für die Signalisierung in jeder Unteraufgabe für Modi vorhanden sein und müssen nicht extra aktiviert werden.

- Da nur neue Synchronisationen hinzukommen und keine entfällen, kann eine Verzweigung eingebaut werden, welche den Status des Ablaufplaner in einer festen Speicheradresse liest, und im Falle der Transition die zusätzlichen Aufrufe aktiviert.

Bei letzterem Ansatz können Wettlaufsituationen (engl. race conditions) bei der Entscheidung für einen Zweig auftreten, weshalb hier auch nach dem Wechsel noch Signale fehlen können. Da neue Abhängigkeiten bereits zu einem Zeitpunkt vor dem Wechsel im Verlauf der Unteraufgabe ausgelöst werden müssten, sollen diese auf Aufgabenbasis gespeichert und nachgereicht werden, sodass alle Unteraufgaben innerhalb des Intervalls ihrer Aufgabe abgeschlossen werden. Dies ist durch den ersten der beiden Ansätze einfacher realisierbar, weshalb dieser gewählt wird. Für Synchronisation zwischen alten und neuen Aufgaben kann eine ähnliche Lösung verwendet werden, allerdings muss für das Aktivieren neuer Aufgaben grundsätzlich noch das Problem des Zusammenhangs mit anderen Aufgaben gelöst werden, weshalb nicht weiter darauf eingegangen wird. Gleiches gilt für den Zusammenhang von neuen Unteraufgaben alter Aufgaben, welche von neuen Aufgaben

abhängig sind. Nachdem die Systemaufrufe bereits verändert werden, wird die ABB-ID im gleichen Schritt mit eingebaut.

Nun muss noch der Wechsel in eine zeitgesteuerte Ausführung erfolgen. Die ABBs jeder Unteraufgabe wurden dabei in die Zeitfenster untergliedert, sodass für den letzten ABB in einem Zeitfenster eine Operation eingefügt werden muss, welche auf das nächste Zeitfenster wartet, beziehungsweise die Aufgabe im letzten Fenster beendet. Hierzu können wie für die neue Synchronisation entweder die Systemaufrufe angepasst oder mittels Verzweigungen zwischen Synchronisationsprimitiven und der Warteoperation gewechselt werden. Das Hauptproblem bei Ersterem ist die unnötig hohe Anzahl an Systemaufrufen, die, auch wenn keine Arbeit geleistet wird, eine größere Latenz als Verzweigungen hat und welche in zeitgesteuerten Systemen üblicherweise möglichst selten vorkommen sollen. In einer Mischform der beiden wird die erste Lösung bis zum Übergang und bis zur ersten Beendigung im zeitgesteurten Modus verwendet, woraufhin in der nächsten Ausführung eine Variante ohne unnötige Systemaufrufe gestartet wird. Dies kann entweder mittels einer Verzweigung mit einer am Anfang der Periode evaluierten Bedingung oder mit einem zweiten Aktivitätsträger mit angepasstem Programmcode geschehen.

3.6 Zusammenfassung

In diesem Kapitel wurden verschiedene Aspekte des Moduswechsel besprochen. Die Unterteilung des Wechsels bezüglich Aufgaben und Steuerungsart ermöglicht ein strukturiertes Vorgehen für zwei getrennte Bereiche. Es wurden die Veränderungen der Abhängigkeiten beim Hinzufügen von weiteren eingeschränkt, da entfernte nur für eine freiere Reihenfolge sorgen, welche eigentlich in einem normalen anstatt einem kritischen Modus erwartet wird. Die gerichteten Abhängigkeiten bedingen weiterhin die Notwendigkeit der zeitgleichen Änderung von Parameter und des koordinierten Starts neuer zusammenhängender Aufgaben, um keine Verschiebung zu erzeugen. Die Struktur der Anwendung und der Fortschritt dieser wird mittels der Unterteilung in ABBs gemessen und durch das RTSC ausgegeben. Abschließend wurden Möglichkeiten zur Übertragung der theoretischen Änderungen auf ein tatsächliches Programm vorgestellt.

4 ALGORITHMUS FÜR EINEN MODUSWECHSEL

In diesem Kapitel wird ein Algorithmus zum Moduswechsel erläutert. Es wird der EDF (erster Termin zuerst) Algorithmus in der ereignisgesteuerte/ Phase, eine Anpassung dessen in der Transitionsphase und ein Zeitplan mit beliebiger Reihenfolge vorausgesetzt. Keine Aufgabe verändert ihre Parameter und es werden die vorher beschriebenen Bedingungen gestellt. Es wird hauptsächlich der zweite Teil des Moduswechsels - der Übergang von einem ereignisgesteuerten zu einem zeitgesteuerten Modus - untersucht und das Hinzufügen neuer Aufgaben vorerst ohne Beachtung ihrer Abhängigkeiten durchgeführt.

4.1 Algorithmusbeschreibung

Grundlegend wird der Zeitplan der Zeitsteuerung zyklisch durchlaufen und in diesem der Fortschritt der Ereignissteuerung markiert. Die Aufgaben selbst werden durch ihre Ereignisse gestartet und nach einem EDF eingeplant. Der Plan dient also nur der Informationsverwaltung im ersten Modus. Periodische und nicht-periodische Aufgaben werden wie in oben beschrieben in der Tabelle markiert.

Der Wechsel soll nun zu einen beliebigen Zeitpunkt τ eintreten. Nachdem der Zeitplan zyklisch abgearbeitet wird, betrachten wir τ Modulo der Länge des Zeitplans, sodass τ stets ein Punkt im Zeitplan ist. Während der gesamten Ausführung wird ein genauer Zeitgeber angenommen, anhand dessen zum einen die periodischen Ereignisse mit minimalem Jitter ausgelöst werden und zum anderen der aktuelle Zeitpunkt t im Plan gemessen wird.

Es werden folgende Mengen benötigt:

- Fehlende alte Aktivitäten vor dem aktuellen Zeitpunkt $A_v(t)$.
- Fehlende alte Aktivitäten nach dem aktuellen Zeitpunkt $A_n(t)$.
- Fehlende neue Aktivitäten vor dem aktuellen Zeitpunkt $N_v(t)$.
- Fehlende neue Aktivitäten nach dem aktuellen Zeitpunkt $N_n(t)$.

Diese werden in der Phase I berechnet und kommen in der Transitionsphase zum Einsatz. Die Mengen beziehen sich auf den Fortschritt im Zeitplan.

In Phase I wird ein EDF ausgeführt, sodass alle lauffähigen Aktivitätsträger, also die Unteraufgaben, nach Termin aufsteigend in der Bereitliste sortiert sind. Wenn ein Ereignis ausgelöst, beziehungsweise eine blockierte Aufgabe lauffähig wird, werden die obigen Mengen konzeptionell zusammen mit der Bereitliste aktualisiert. Aufgaben des ereignisgesteuerten Modus werden dabei

anhand ihres Fortschritts im Zeitplan in eine der Mengen eingeteilt. Beispielsweise ist eine Unteraufgabe τ_i, deren zuletzt angefangener ABB zum aktuellen Zeitpunkt t zur Zeit t_1 gestartet wird, also ist $\tau_i \in A_v(t)$, falls $t_1 \leq t$ respektive $\tau_i \in A_n(t)$, falls $t_1 > t$. Selbiges wird für die neu hinzukommenden Aktivitäten wiederholt, wobei vor der Transition jede Aufgabe in $N_v(t)$ ist, falls deren Termin noch nicht abgelaufen ist. Vom Einplaner wird damit in Phase I nicht mehr als von einem normalen, unveränderten EDF erwartet, weshalb die Aufgaben unter Beachtung der erhöhten Kosten durch der zusätzlichen Informationsverwaltung ohne Terminverletzung ausführbar sind, solange das System planbar ist. Die Optimalität des EDFs im Ein-Kern-Betrieb garantiert dies[Lóp+00].

Beim nächsten Schritt, das Hinzufügen der neuen Aufgaben, geht momentan von einem geringen Zusammenhang der Aufgaben aus und muss eigentlich, wie besprochen, unter Einbezug aller zusammenhängender Aufgaben stattfinden. Nach Signalisierung des Wechsels werden in Phase II zum Zeitpunkt τ alle Aufgaben in $N_v(\tau)$ lauffähig und zusammen mit allen lauffähigen Aufgaben $A_v(\tau)$ in eine neue Bereitliste eingetragen. Mithilfe dieser Bereitliste wird nun ebenfalls ein EDF ausgeführt. Der Unterschied zum bisherigen Algorithmus ist der Inhalt der Bereitliste und wann diese verändert wird. Aufgaben werden nicht mehr anhand ihrer Ereignisse in die Liste eingetragen, sondern erst sobald die aktuelle Zeit den Start des ersten Intervals der Aufgabe im Zeitplan überschreitet. Also werden die Aufgaben, sobald sie von $N_n(t)$ nach $N_v(t)$ beziehungsweise von $A_n(t)$ nach $A_v(t)$ mit Fortschritt der Zeit t gelangen, in die Bereitliste eingetragen. Damit sind die zur Auswahl stehenden Aufgaben stets diejenigen, welche in einer Zeitsteuerung weiter bearbeitet worden wären, als sie es bisher wurden, womit alle Aufgaben nach und nach mit dem Zeitplan aufholen. Sobald keine Aktivitätsträger mehr in der Bereitliste übrig sind und keine Aufgabe weiter ausgeführt worden ist als der aktuelle Zeitpunkt, kann in einen vollständig zeitgesteuerten Betrieb übergangen werden.

Dabei muss allerdings noch auf die wartenden Unteraufgaben geachtet werden, da diese nicht in der Bereitliste vorkommen. Eine wichtige Beobachtung ist, dass alle Wartenden in dem Fall an einer Sperre warten müssen beziehungsweise auf ein Signal warten, das entweder auf einen Mutex wartet oder rekursiv wieder auf ein Signal wartet, welches selbst auf entweder einen Mutex oder rekursiv auf ein Signal und so weiter wartet. Grundsätzlich gilt allerdings, dass als letztes in der Kette auf eine Sperre gewartet werden muss. Angenommen die letzte Warteoperation bezieht sich auf ein Signal, dann muss der auslösende ABB vorher im Zeitplan vorkommen, womit er in der Bereitliste wäre. Falls allerdings auf eine Sperre gewartet wird, dann kann diese von einer Unteraufgabe gehalten werden, welche im Zeitplan später drankommt. Wenn obiger Zeitpunkt erreicht ist, zu dem die Bereitliste leer und keine Aufgabe im Zeitplan nach dem aktuellen Interval ausgeführt worden ist, so kann die Sperre erst in einem späteren freigegeben werden. Daher müssen alle Aufgaben, welche eine Sperre halten immer in der Bereitliste vorkommen, egal wie weit sie fortgeschritten sind, außer sie sind selbst blockiert. Solange kein Deadlock vorkommt, wird die Bereitliste erst leer sein, wenn keine Sperren mehr gehalten sind und daher keine weiteren Aufgaben blockiert sind, die vor dem aktuellen Zeitpunkt hätten bearbeitet werden sollen. Das Vorkommen der Aktivitäten mit Sperren in der Bereitliste kann abgeschwächt werden, indem nur Aufgaben hinzugefügt werden, deren Sperre von einer weiteren Aufgabe benötigt wird, welche selbst im Zeitplan noch nicht so weit fortgeschritten ist, wie die Sperre haltende Unteraufgabe. Denn alle Aufgaben, welche auf die Sperre warten, die weiter fortgeschritten und von denen weitere Aufgaben abhängig, also blockiert, sind, verzögern nur Aufgaben, welche nach dem Interval, nach dem die Sperre wieder freigegeben wird, eingeplant worden sind.

Der Algorithmus bietet einen schnelle und sicheren Übergang in eine Zeitsteuerung. Indem nur die notwendigen, fehlenden Aufgaben zur Auswahl stehen, wird eine weitere Abweichung von der geplanten Reihenfolge vermieden und aktiv das Nachholen unterstützt.

Eine mögliche Anpassung wäre das Vergrößern der Bereitliste bis zum am weitesten fortgeschrittenen Interval. Dadurch wird ein dynamischerer Ablauf in der Transition erlaubt, welcher

dem ursprünglichen EDF stärker entspricht und den Laufzeitaufwand reduziert. Aus der Bedingung, dass keine Lücken im Zeitplan vor dem Wechseln in den zeitgesteuerten Modus vorhanden sind, folgt, dass diese zusätzlichen Aufgaben in der ersten Variante des Algorithmus ebenfalls ausgeführt werden würden, bevor die Phase beendet werden kann. Die genauen Auswirkungen auf die Dauer des Übergangs und der Latenz sollten allerdings in einem realen System getestet und evaluiert werden.

4.2 Planbarkeit

Der Algorithmus kann nur angewandt werden, wenn die Termineinhaltung garantiert werden kann. Dazu greifen wir auf die Optimalität des EDF zurück. Diese besagt, dass der EDF, falls das System planbar ist, eine Lösung findet. Da Aufgaben nun nicht mehr ab ihres Auslösezeitpunkts lauffähig sind, sondern erst nur schrittweise ab Intervalbeginn, garantiert die Planbarkeit der Aufgaben nicht mehr die Optimalität.

Um dies zu umgehen, wird der modifizierte EDF auf einen normalen abgebildet, indem nicht mehr Aufgaben mit ihrem Termin betrachtet, sondern neue Aufgaben bestehend aus dem Intervalbeginn mit dem Termins der korrespondierenden ursprünglichen Aufgabe konstruiert werden. Dazu werden alle Intervalle $[b, e)$ des Zeitplans betrachtet, wobei diese paarweise nicht überlappen und jeweils einen Teil einer Unteraufgabe ausführen. Zu jedem Interval wird der Termin d der Unteraufgabe zugewiesen, womit der Zeitraum $n := [b, d)$ entsteht, in welchem der Teil ausgeführt werden muss. Indem alle neuen n als periodische Ereignisse mit Phase b, relativen Termin $d - b$, und Periode gleich der Länge des Zeitplans betrachtet werden, kann statt des modifizierten EDFs ein normaler ausgeführt werden, wobei zu allen Zeitpunkten die Bereitlisten beider Algorithmen dieselben Intervalle des Zeitplan enthalten, womit die Entscheidungen ebenfalls übereinstimmen. Damit garantiert die Optimalität des EDF auch die Optimalität der modifizierten Variante und durch die Existenz des Zeitplans ist die Planbarkeit gegeben, womit alle Aufgaben vor ihrem Termin beendet werden.

Bisher wurde der Startzustand ignoriert. Doch dieser kann ignoriert werden, solange keine neuen Aufgaben nachgeholt werden müssen, da die Bereitliste beim Wechsel dann ausschließlich kleiner werden kann und durch der Optimalität des EDFs die bisherigen Aufgaben in einer Reihenfolge ausgeführt wurden, in der sie termingerecht beendet werden, falls es möglich ist. Der modifizierte EDF führt diese Auswahl nun optimal fort. Die Sonderbehandlung der Sperren kann vernachlässigt werden, da die entsprechenden Aufgaben immer Teil der Bereitliste im normalen EDF sind und somit eine geringere Abweichung von diesem darstellen als die restlichen Aufgaben.

Wenn neue Aufgaben beim Wechsel nachgeholt werden, kann es notwendig sein, dass der Zeitplan nicht vollständig gefüllt ist, sondern spezielle Lücken zum Aufholen der neuen Aufgaben besitzt. Da für einen kontrollierten Ablauf wegen der Abhängigkeiten feste Zeitpunkte berechnet werden müssen, wann die neuen Aufgaben jeweils lauffähig werden dürfen, ist zusätzliche Last zu jedem Zeitpunkt genau bekannt. Dies wird vereinfacht, indem der Wechsel nur zu festen Zeitpunkten w_i zugelassen wird. Zu jedem w_i kann jetzt der schlechteste Zustand abgeschätzt werden, zu dem die bezüglich des Zeitplans noch nicht berechneten Intervalle vor dem Wechsel die maximale Arbeitszeit max und die bereits berechneten Intervalle nach dem Wechsel die minimale Arbeitszeit min benötigen. Die Zeit $max - min$ muss nun eingeplant werden, um das Nachholen zu ermöglichen, indem entweder eigene Zeitfenster eingebaut oder die bestehenden verlängert werden.

4.3 Aufwand des Algorithmus zur Laufzeit

Um den Algorithmus in einem Echtzeitbetrieb einzusetzen, muss die maximale Latenz berechenbar und gering sein. Grundsätzlich wird der bereits analysierte EDF verwendet, weshalb nur die Veränderungen genauer betrachtet werden. Die Fortschritte müssen jederzeit markiert werden, daher entsteht bei jeder Blockade und jeder Freigabe eines Signals beziehungsweise einer Sperre ein zusätzlicher aber konstanter Aufwand. Die Verwaltung der Bereitliste in der Transitionsphase ist insgesamt aufwändiger als in Phase I, da bei jedem Interval eine Aufgabe eingefügt werden muss, doch der Aufwand für das Einfügen pro Aufgabe bleibt unverändert. Durch das Verwenden eines balancierten Baumes ist die Operation bezüglich der Länge n der Bereitliste in $O(\log n)$ machbar. Die Länge der Bereitliste ist sogar kleiner als in Phase I, da nicht alle Aufgaben in dieser vorkommen, und wird umso geringer umso stärker sich der Fortschritt der Aufgaben dem Zeitplan anpasst.

Das einzige Problem ist die initiale Bereitliste der Transitionsphase. Diese kann entweder kontinuierlich in Phase I bereitgehalten werden, falls der Wechsel eintreten sollte, womit sich der Aufwand verteilt, oder erst zum Wechsel berechnet werden. Für die erste Variante kann die Bearbeitung aus der Transitionsphase übernommen werden, was den Aufwand zu jedem Aufruf des Einplaners ungefähr verdoppeln würde. Im Kontrast dazu steht die Berechnung zum Zeitpunkt des Wechsels, bei der einmal über die Bereitliste iteriert werden muss und jede Aufgabe entfernt wird, welche weiter gerechnet hat als der Zeitplan für aktuellen Zeitpunkt vorsieht und keine Sperre hält. Der maximale Inhalt der Bereitliste müsste daher abgeschätzt werden, sodass Wechsel nicht zu Zeitpunkten stattfinden, bei welchem der Wechsel die Ausführung soweit verzögert, dass Aufgaben ihre Termine verpassen. Falls für neue Aufgaben wie oben beschrieben der schlimmste Zustand bezüglich des Fortschritts berechnet wird, kann dies ebenfalls im gleichen Schritt erarbeitet werden. Falls darauf verzichtet werden sollte bietet sich die erste Lösung wegen ihres fest definierten Verhaltens an.

4.4 Zusammenfassung

Es wurde ein konkreter Algorithmus vorgestellt, mit dem von einer ereignisgesteuerten zu einer zeitgesteuerten Ausführung übergegangen werden kann. Als Basis wurde der bekannte EDF gewählt, um die Optimalität zu garantieren und keine Termine zu verletzen. Durch die reduzierten Auswahlmöglichkeiten wird zum einen die Anpassung an den Zeitplan ermöglicht, sowie die Optimalität in der Transitionsphase erhalten. Für neue Aufgaben muss zusätzliche Arbeit vor dem Systemstart vollzogen werden, um alle Abhängigkeiten erfüllen zu können, was in der bisherigen Revision vermieden wird. Der Aufwand des gesamten Algorithmus kann angegeben und zur Planung des Ablaufs verwendet werden, um die Sicherheitsforderungen des Echtzeitbetriebs zu erfüllen.

5 VERWANDTE ARBEITEN

Im Gegensatz zum binären Moduswechsel dieser Arbeit, bei dem alle Aufgaben dieselbe *Kritikalität* besitzen und in einer kritischen Situation im Gesamten verändert werden, sind unter dem Begriff *gemischte Kritikalität* (engl. mixed-criticality) Systeme entstanden, die Aufgaben unterschiedlicher Wichtigkeit enthalten, sodass zur selben Zeit verschiedene Kategorien abgearbeitet werden können. Eine in den meisten Arbeiten getroffene Annahme ist die Bearbeitung von unabhängigen Komponenten. [BD] Daraus wird eine Einzelbehandlung der korrespondierenden Aufgaben erlaubt, was dazu führt, dass Aufgaben beliebig ausgesetzt werden dürfen. Generell wird Aufgaben geringerer Kritikalität nur dann Rechenzeit zugeordnet, wenn alle Aufgaben höherer Kritikalität ihre WCET an Rechenzeit vor ihrem Termin bekommen können. Eine optimale Lösung für das Einplanungsproblem mit mehr als einer Kritikalitätsstufe wurde als NP-Schwer gezeigt. [Bar+12]

Bisher verwendete Algorithmen wie beispielsweise der EDF oder RMS[13] wurden in einigen Papieren bereits untersucht und nicht länger als optimal in Umgebungen mit unterschiedlichen Kritikalitäten befunden und daher optimiert. Dazu wurde ein fester Prioritäts-Algorithmus (engl. fixed-priority) von Vestal mittels der Verwendung von Audsleys Prioritätszuweisungsalgorithmus [Aud01] vorgestellt und später als optimal bewiesen. [BV08; BD]

Weiterhin wurden Möglichkeiten zur Feststellung der Planbarkeit einer gegeben Aufgabenmenge gemischter Kritikalität vorgestellt, damit der Algorithmus in dieser Umgebung ebenfalls weiterhin eingesetzt werden kann. Indem der Rechenbedarf der weniger kritikalen Aufgaben abgeschätzt und eine obere Grenze angegeben werden kann, ermöglichen verschiedene, relative Termine in den Modi die Anwendbarkeit des EDF mit nicht-periodischen Aufgaben. [EY14]

Ein weitere Ansatz für gemischte Kritikalitätssysteme ist MC^2, welcher auf Mehr-Kern-Systemen mit vier verschiedenen Kritikalitätsleveln operiert, in denen unterschiedliche Aufgaben ausgeführt werden. Es unterscheidet sich von anderen durch die Verwendung unterschiedlicher Einplanungsalgorithmen für die verschiedenen Stufen, wobei eine strikte Priorisierung von Aufgaben höherer Kritikalität stattfindet, wodurch das Problem nicht optimal gelöst wird. Weiterhin sind die Aufgaben statisch in eine der Stufen eingeteilt und können diese nicht verlassen. [Her+12]

Abhängig von den genauen Anforderungen des technischen Systems, welches durch das Echtzeitsystem gesteuert wird, sind nie alle Annahmen gültig und erfordern spezielle Lösungen. Die obigen Beispiele machen beispielsweise Annahmen über die Art der Aufgaben, sodass diese keinen gemeinsamen Speicher nutzen oder keinen Zusammenhang durch Synchronisation besitzen dürfen, um eine korrekte Funktionsweise zu gewährleisten. Es werden teilweise keine Aufgaben übernommen, sodass deren Ergebnisse verfallen oder zumindest die Termineinhaltung nicht garantiert wird. Oder der sichere Zustand wird unverändert mittels einer Ereignissteuerung hergestellt und nicht durch eine statisch geplante Reihenfolge erreicht.

[13] Rate-monotonic Scheduling

Ein ausführliches Nachschlagewerk über den Fortschritt der Echtzeitsysteme im gemischten Kritikalitätsbereich wurde von Alan Burns und Robert I. Davis unter *Mixed Criticality Systems - A Review* veröffentlicht. [BD]

6 FAZIT

Obwohl starke Einschränkungen notwendig sind, wurde eine Möglichkeit vorgestellt, wie ein Moduswechsel ohne die Unabhängigkeitsbedingung an die Aufgaben vollzogen werden kann. Die Untersuchung der Abhängigkeiten führt zu Anforderungen an die Abbildung auf ein technisches System, welche das Ein- und Ausschalten der Instruktionen unterstützen muss, um die Kommunikation mit neu hinzukommenden Aufgaben zu erlauben. Dazu sind alle Abhängigkeiten zur Zeit der Ereignissteuerung vorhanden und werden innerhalb des Kernes geschützt vor der Verdrängung durch den Scheduler nur im jeweilig richtigen Modus ausgeführt.

Im gleichen Zug kann die Verwaltung der Fortschrittsinformationen stattfinden, welche allen Synchronisationsprimitiven die aktuell ausgeführten ABBs der Ablaufplanung mitteilt, womit dieser den Fortschritt innerhalb des Zeitplans messen kann. Mithilfe dieser Informationen können gezielte Entscheidungen getroffen werden, um den Übergang in den Zeitgesteuerten Modus schnellstmöglich zu verrichten.

Der Ablaufplaner wählt dann die fehlenden Aufgaben im Zeitplan aus, sodass der Ausgleich des bisher unterschiedlichen Fortschritts stattfindet, bis alle Aufgaben den gleichen Stand besitzen und der Wechsel in die Zeitsteuerung keine Abhängigkeitsverletzungen nach sich ziehen kann. Der Aufwand des Algorithmus ist vergleichbar mit dem regulärer ereignisgesteuerter Ablaufplaner, weshalb die Nutzung in realen Systemen möglich sein sollte.

Daraus ergeben sich neue zu lösende Fragestellungen, wie das Starten der neuen Aufgaben. Bisher wurde dies nur grob skizziert, da es den Rahmen der Arbeit übersteigen würde, doch der Zeitpunkt zu welchem neue Aufgabe hinzugefügt werden können, steht im Zusammenhang mit allen anderen Aufgaben und muss unter Betrachtung aller Abhängigkeiten bestimmt werden. Das kann allerdings für das System vor der Ausführung stattfinden und der Ablaufplanung mitgeteilt werden, sodass kein zusätzlicher Laufzeitaufwand betrieben werden muss.

Sobald diese Frage gelöst ist, kann das gesamte System implementiert und getestet werden, um die bisher theoretischen Ergebnisse zu evaluieren. Die Skalierbarkeit mit der Anzahl der Aufgaben und des Grades des Zusammenhangs sollte dabei mit der Funktionsfähigkeit überprüft werden, sodass Aussagen über die Nutzbarkeit in realen Systemen getroffen werden können.

Wenn sich das bisher vorgestellte System als funktionsfähig erweist, kann begonnen werden Restriktionen zu schwächen. Es kann eine größere Menge an Aufgabenänderungen zugelassen werden, wenn beispielsweise Parameteränderungen synchron stattfinden, oder sogar der Ein-Kern-Betrieb auf einen Mehr-Kern-Betrieb erweitert werden.

ABKÜRZUNGSVERZEICHNIS

API	Application Programming Interface
RTSC	Real-Time Systems Compiler
ABB	Atomic Basic Block
EDF	Earliest Deadline First
RTOS	Real-Time Operating System
WCET	Worst-Case Execution Time

ABBILDUNGSVERZEICHNIS

LITERATUR

[Aud01] N. C. Audsley. “On Priority Asignment in Fixed Priority Scheduling”. In: *Inf. Process. Lett.* 79.1 (Mai 2001), S. 39–44. ISSN: 0020-0190. DOI: 10.1016/S0020-0190(00)00165-4. URL: http://dx.doi.org/10.1016/S0020-0190(00)00165-4.

[Bar+12] Sanjoy Baruah u. a. “Scheduling real-time mixed-criticality jobs”. In: *IEEE Transactions on Computers* 61.8 (2012), S. 1140–1152.

[BD] Alan Burns und Robert Davis. “Mixed criticality systems-a review”. In: ().

[Bro05] M. Broy. “Automotive software and systems engineering”. In: *Proceedings. Second ACM and IEEE International Conference on Formal Methods and Models for Co-Design, 2005. MEMOCODE ’05.* 2005, S. 143–149. DOI: 10.1109/MEMCOD.2005.1487905.

[Bro06] Manfred Broy. “Challenges in Automotive Software Engineering”. In: *Proceedings of the 28th International Conference on Software Engineering*. ICSE ’06. Shanghai, China: ACM, 2006, S. 33–42. ISBN: 1-59593-375-1. DOI: 10.1145/1134285.1134292. URL: http://doi.acm.org/10.1145/1134285.1134292.

[BV08] Sanjoy Baruah und Steve Vestal. “Schedulability analysis of sporadic tasks with multiple criticality specifications”. In: *Real-Time Systems, 2008. ECRTS’08. Euromicro Conference on*. IEEE. 2008, S. 147–155.

[EY14] Pontus Ekberg und Wang Yi. “Bounding and shaping the demand of generalized mixed-criticality sporadic task systems”. In: *Real-time systems* 50.1 (2014), S. 48–86.

[Her+12] Jonathan L Herman u. a. “RTOS support for multicore mixed-criticality systems”. In: *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2012 IEEE 18th*. IEEE. 2012, S. 197–208.

[Kop06] Hermann Kopetz. *Real-Time Systems - Design Principles for Distributed Embedded Applications*. Berlin Heidelberg: Springer Science & Business Media, 2006. ISBN: 978-0-306-47055-4.

[Kop91] H. Kopetz. “Event-triggered versus time-triggered real-time systems”. In: *Operating Systems of the 90s and Beyond: International Workshop Dagstuhl Castle, Germany, July 8–12 1991 Proceedings*. Hrsg. von Arthur Karshmer und Jürgen Nehmer. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, S. 86–101. ISBN: 978-3-540-46630-7. DOI: 10.1007/BFb0024530. URL: https://doi.org/10.1007/BFb0024530.

[LH09] Merz Ludwig und Jaschek Hilmar. *Grundkurs der Regelungstechnik, Einführung in die praktischen und theoretischen Methoden*. 2009 PB. ISBN: 978-3-486-25960-5. URL: https://www.degruyter.com/view/product/226120.

[LN78] H.C. Lauer und R.M. Needham. “On the Duality of Operating Systems Structures”. In: *Second International Symposium on Operating Systems*. reprinted in Operating Systems Review, 13,2 April 1979. 1978, pp. 3–19.

[Lóp+00] Jose Maria López u. a. “Worst-case utilization bound for EDF scheduling on real-time multiprocessor systems”. In: *Real-Time Systems, 2000. Euromicro RTS 2000. 12th Euromicro Conference on*. IEEE. 2000, S. 25–33.

[MB05] N. Meghanathan und S. Baskiyar. “A survey of contemporary real-time operating systems”. In: *Informatica* 29.2 (2005).

[RC01] J. Real und A. Crespo. “Offsets for scheduling mode changes”. In: *Proceedings 13th Euromicro Conference on Real-Time Systems*. 2001, S. 3–10. DOI: 10.1109/EMRTS.2001.933989.

[Sch06] Peter Scholz. *Softwareentwicklung eingebetteter Systeme - Grundlagen, Modellierung, Qualitätssicherung*. Berlin Heidelberg New York: Springer-Verlag, 2006. ISBN: 978-3-540-27522-0.

[Sch11] Fabian Scheler. “Atomic Basic Blocks: Eine Abstraktion für die gezielte Manipulation der Echtzeitsystemarchitektur”. Diss. Erlangen: Friedrich-Alexander-Universität Erlangen-Nürnberg, 2011. URL: http://opus4.kobv.de/opus4-fau/files/1740/FabianSchelerDissertation.pdf.

[Sha+89] Lui Sha u. a. “Mode change protocols for priority-driven preemptive scheduling”. In: *Real-Time Systems* 1.3 (1989), S. 243–264. ISSN: 1573-1383. DOI: 10.1007/BF00365439. URL: https://doi.org/10.1007/BF00365439.

[SSP10] Fabian Scheler und Wolfgang Schröder-Preikschat. “The RTSC: Leveraging the Migration from Event-Triggered to Time-Triggered Systems”. In: *Proceedings of the 13th IEEE International Symposium on Object/Component/Service-oriented Real-time Distributed Computing (ISORC ’10)*. Hrsg. von Roman Obermaisser. Carmona, 2010, S. 34–41. ISBN: 978-0-7695-4037-5. DOI: 10.1109/ISORC.2010.11. URL: http://www4.informatik.uni-erlangen.de/Publications/2010/scheler_isorc2010_RTSC.pdf.

[SSP11] Fabian Scheler und Wolfgang Schröder-Preikschat. “The Real-Time Systems Compiler: migrating event-triggered systems to time-triggered systems”. In: *Software: Practice and Experience* 41.12 (2011), S. 1491–1515. ISSN: 1097-024X. DOI: 10.1002/spe.1099. URL: http://dx.doi.org/10.1002/spe.1099.

[TBW92] K. W. Tindell, A. Burns und A. J. Wellings. “Mode changes in priority preemptively scheduled systems”. In: *[1992] Proceedings Real-Time Systems Symposium*. 1992, S. 100–109. DOI: 10.1109/REAL.1992.242672.